



**Univerzitet u Beogradu
Elektrotehnički fakultet**

SISTEM ZA VIZUELNU REPREZENTACIJU BLOKČEJN TEHNOLOGIJE

DIPLOMSKI RAD

Kandidat:

Dimitrije Knežević 244/2017

Profesor:

prof. dr Žarko Stanisavljević

Beograd
mart 2023.

SADRŽAJ

1. UVOD.....	5
2. TEORIJSKA OSNOVA BLOCKCHAIN-A	6
2.1. Kao Struktura Podataka	6
2.1.1. Kriptografske Heš Funkcije	6
2.1.2. Magični Broj.....	9
2.1.3. Vremenski Žig.....	9
2.1.4. Sadržaj Bloka	9
2.1.5. Digitalni Potpis.....	10
2.1.5.1. ECDSA.....	10
2.1.6. Nonce i Težina	12
2.1.7. Coinbase	13
2.2. Kao Mrežni Protokol	13
2.2.1. Tipovi Blockchain-a	14
2.2.1.1. Javni.....	14
2.2.1.2. Privatni	15
2.2.1.3. Konzorcijum	15
2.2.2. Konsenzus Algoritmi.....	15
2.2.2.1. Proof of Work	15
2.2.3. Blockchain Forking	16
3. IMPLEMENTACIJA	18
3.1. Prednosti i mane u odnosu na druga moguća rešenja.....	18
3.2. Korišćene Tehnologije.....	19
3.2.1. HTML.....	19
3.2.1.1. Pug.....	19
3.2.2. CSS	20
3.2.2.1. SASS.....	20
3.2.2.2. Bootstrap.....	20
3.2.3. Javascript	20
3.2.3.1. Node.js	20
3.2.3.2. jQuery.....	21
3.2.4. Camtasia 9	21

3.3. Arhitektura Sistema	21
3.4. Organizacija Fajlova	22
3.4.1. Metadata Fajlovi	22
3.4.2. Index.js	23
3.4.3. Layout.pug	23
3.4.4. Bs-config.js	23
3.4.5. Globalne Funkcije	23
4. PRIMER RADA SISTEMA	24
4.1. Pokretanje Aplikacije	25
4.2. Index.pug	25
4.3. CHF.pug	26
4.4. Block.pug	29
4.5. Blockchain.pug	30
4.6. Transactions.pug	33
4.7. Coinbase.pug	34
4.8. Keys.pug	36
4.9. Signatures.pug	36
4.10. Signed_Txs_and_lds.pug	39
4.11. MiningDiff.pug	41
4.12. Mining.pug	42
4.13. Distributed.pug	44
4.14. Network.pug	47
4.15. References.pug	50
4.16. About.pug	51
5. ZAKLJUČAK	52
6. LITERATURA	54
7. TABELA SLIKA	57

8. TABELA BLOKOVA KODA	59
9. PRILOG	60
9.1. Ostali Konsenzus Algoritmi.....	60
9.1.1. Proof of Stake.....	60
9.1.2. Delegated Proof of Stake	60
9.1.3. Proof of Burn	61
9.1.4. Proof of Capacity	61
9.1.5. Proof of Elapsed Time.....	62
9.2. Napadi na Blockchain.....	62
9.2.1. Peer-to-Peer Mrežni Napadi.....	63
9.2.1.1. Eclipse Attack.....	63
9.2.1.2. Sybil Attack	63
9.2.2. Napadi na Konsenzus Mehanizam i Proces Rudarenja.....	63
9.2.2.1. Selfish-Mining Attack	63
9.2.2.2. Rudarski Malware	63
9.2.2.3. 51% Attack	64
9.2.2.4. Timejack Attack.....	64
9.2.2.5. Finney Attack	64
9.2.2.6. Race Attack.....	64
9.2.3. Napadi Korišćenjem Pametnih Ugovora	65
9.2.3.1. The DAO Attack	65
9.2.4. Napadi na Novčanike	65
9.3. Istorija	66
9.4. Blokovi Koda	69

1. UVOD

Blokčejn (eng. *blockchain*) je trenutno jedan od najaktuelnijih izraza (eng. *buzzword*) u IT svetu. Iako sama tehnologija postoji već 30-ak godina, u poslednje vreme je došla u pažnju šire javnosti najviše zahvaljujući kriptovalutama (eng. *cryptocurrency*). Zbog toga će fokus ovog rada i aplikacije biti na primeru građenja *blockchain*-a koji će predstavljati pozadinski mehanizam imaginarne kriptovalute. Bitno je napomenuti da su kriptovalute trenutno najšira primena *blockchain*-a, ali ne i jedina. Kako se svet sve više digitalizuje tako će i moguće primene *blockchain*-a rasti, i već postoje pokušaji primene ove tehnologije u različim sferama – od poljoprivrede do politike.

Međutim, zahvaljujući prevelikoj pažnji koju su i klasični mediji i društvene mreže poklonili *blockchain*-u, a pritom bez razumevanja načina na koji on funkcioniše i pozadinskih mehanizama, entuzijasti pomenute tehnologije precenjuju trenutni domen njene korisnosti. Stoga, cilj ovog rada je da se demistifikuje princip rada *blockchain*-a i složenih mehanizama na kojima se on zasniva, objasni koji problemi se time rešavaju, a koji se stvaraju i tako doprinese boljem razumevanju mogućih primena ove tehnologije.

Postoji puno različitih implementacija *blockchain*-a, tako da je autor odlučio da vizuelno prikaže glavne karakteristike i mehanizame koje su zajedničke za većinu verzija *blockchain*-a, i na nekim mestima pomene druge moguće verzije implementacije i pokaže njihove prednosti i mane. Ovo sve je urađeno u vidu web aplikacije na primeru imaginarne kriptovalute i njenih transakcija, ali je naravno prenosivo i na sve druge oblasti u kojima bi *blockchain* mogao da se koristi, jer se radi sa alfanumeričkim podacima (imajući u vidu da se i multimedijalni sadržaji takođe mogu pretvoriti u alfanumeričke podatke).

U drugom poglavlju data je teorijska osnova na kojoj se *blockchain* zasniva. Posebno će se razmatrati *blockchain* kao struktura podataka, a posebno mrežni deo, tj. *blockchain* kao mrežni protokol. U ovom poglavlju je takođe data kratka istorija *blockchain*-a. Detaljniji vizuelni prikaz i objašnjenje svih mehanizama koji se koriste u implementaciji *blockchain*-a će biti provučeni kroz ceo rad.

Treće poglavlje opisuje jezike i okvire (eng. *frameworks*) koji su korišćeni u realizaciji sistema za vizuelnu reprezentaciju *blockchain*-a.

Četvrto poglavlje služi za prolazak kroz ceo sistem web aplikacije, gde će se posebno razmatrati svaka komponenta *blockchain*-a kao strukture podataka i njena implementacija. Struktura će biti takva da se kreće od osnovnih pojmova i onda se daljim rešavanjem mogućih problema postepeno uvode ostali mehanizmi.

U petom poglavlju se daje zaključak u vidu rezimea celog sistema i daljih koraka za njegovo unapređenje, kao i kratak osvrt na *blockchain* kao samu tehnologiju.

2. TEORIJSKA OSNOVA BLOCKCHAIN-A

Blockchain je struktura podataka (i protokol) koji predstavlja digitalnu implementaciju distribuirane glavne knjige (eng. *ledger*)[1]. *Ledger* je trajna kolekcija svih novčanih transakcija (mogu se pratiti i druge stvari) u hronološkom redosledu i koristi se već hiljadama godina, još od kada su ljudi prvobitno počeli da trguju. Ove transakcije su se kroz istoriju beležile na svemu od glinenih tablica do papira, a jedini značajniji pomak u ovoj oblasti pre *blockchain*-a je bila kompjuterizacija koja je prvobitno samo značila prenos sa papira u digitalnu formu.

Blockchain tehnologija je između ostaloga omogućila kreiranje elektronskog sistema za transfer novca bez potrebe za trećim licem koje bi proveravalo i odobravallo transakcije[2] (npr. banke). Ovaj rad se bazira na jednom takvom imaginarnom sistemu. Krajnji korisnici mogu da prebacuju novac iz svojih digitalnih novčanika (eng. *wallet*), a da istovremeno ostanu anonimni i imaju pristup celom *ledger*-u gde su zabeležene sve njihove transakcije i transakcije svih drugih korisnika koji koriste istu kriptovalutu.

U ovom odeljku ćemo proći kroz teoriju mehanizama koje *blockchain* koristi, a detaljnije ćemo videti kako oni funkcionišu i kako se kombinuju u poglavlju 4.

2.1. KAO STRUKTURA PODATAKA

Svaki korisnik *blockchain* sistema ima ličnu kopiju čitavog lanca na svom uređaju. U nastavku ćemo proći kroz bitne elemente i tehnike koje *blockchain* lokalna struktura podataka mora ili može da koristi unutar svakog bloka.

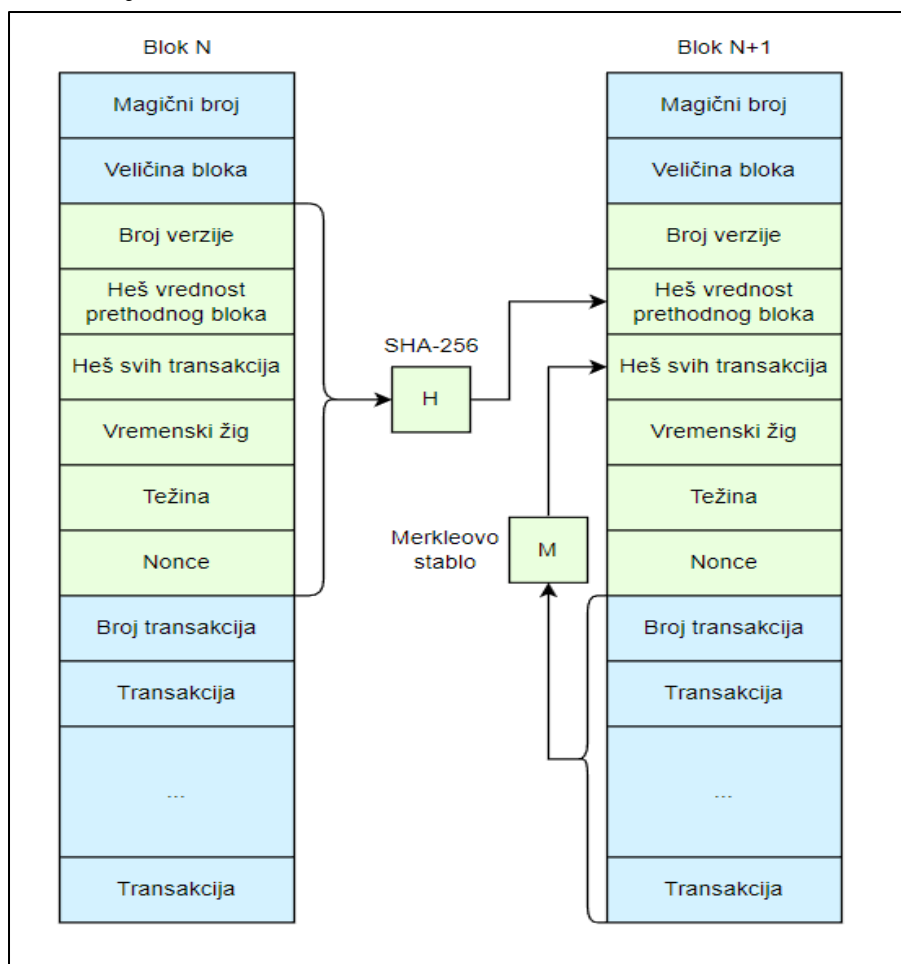
2.1.1. KRIPTOGRAFSKE HEŠ FUNKCIJE

Neophodan element svake verzije implementacije *blockchain*-a je kriptografska heš funkcija (eng. *cryptographic hash function* – *CHF*). *CHF* je jednosmerna funkcija (ne može se izračunati ulaz funkcije na osnovu izlaza) koja od ulaza proizvoljne dužine daje izlaz fiksne dužine[3]. Ulaz može biti bilo šta, pa čak i multimedijalni sadržaj (iako neefikasan), jer se ulaz prvo pretvara u binarni oblik dok je izlaz uvek binarni. Vrednost koju dobijamo na izlazu heš funkcije se zove heš vrednost (ili heš kod).

Jednosmernost je najbitnija odlika *CHF*. Ove funkcije imaju veoma široku primenu u

kriptografiji, internet bezbednosti, kompresiji i optimizaciji, i telekomunikacijama. Jednostavan primer je čuvanje lozinki na nekom web sajtu: Lozinke ne bi trebalo da se čuvaju u originalnom obliku (eng. *plain text*) u bazi podataka već da se čuvaju samo njihove heš vrednosti. Na taj način, čak i ako neautorizovano lice uspe da dođe do baze podataka, sa njom nikad neće moći da pristupi nalogu korisnika, jer ne postoji način da od heširanih vrednosti dođe do originalnih lozinki koje su potrebne da se unesu na ulazu sistema. Sa druge strane, autorizovan korisnik uvek može da pristupi svom nalogu sa validnom ulaznom lozinkom, jer će heširana vrednost lozinke uvek biti ista za isti ulaz.

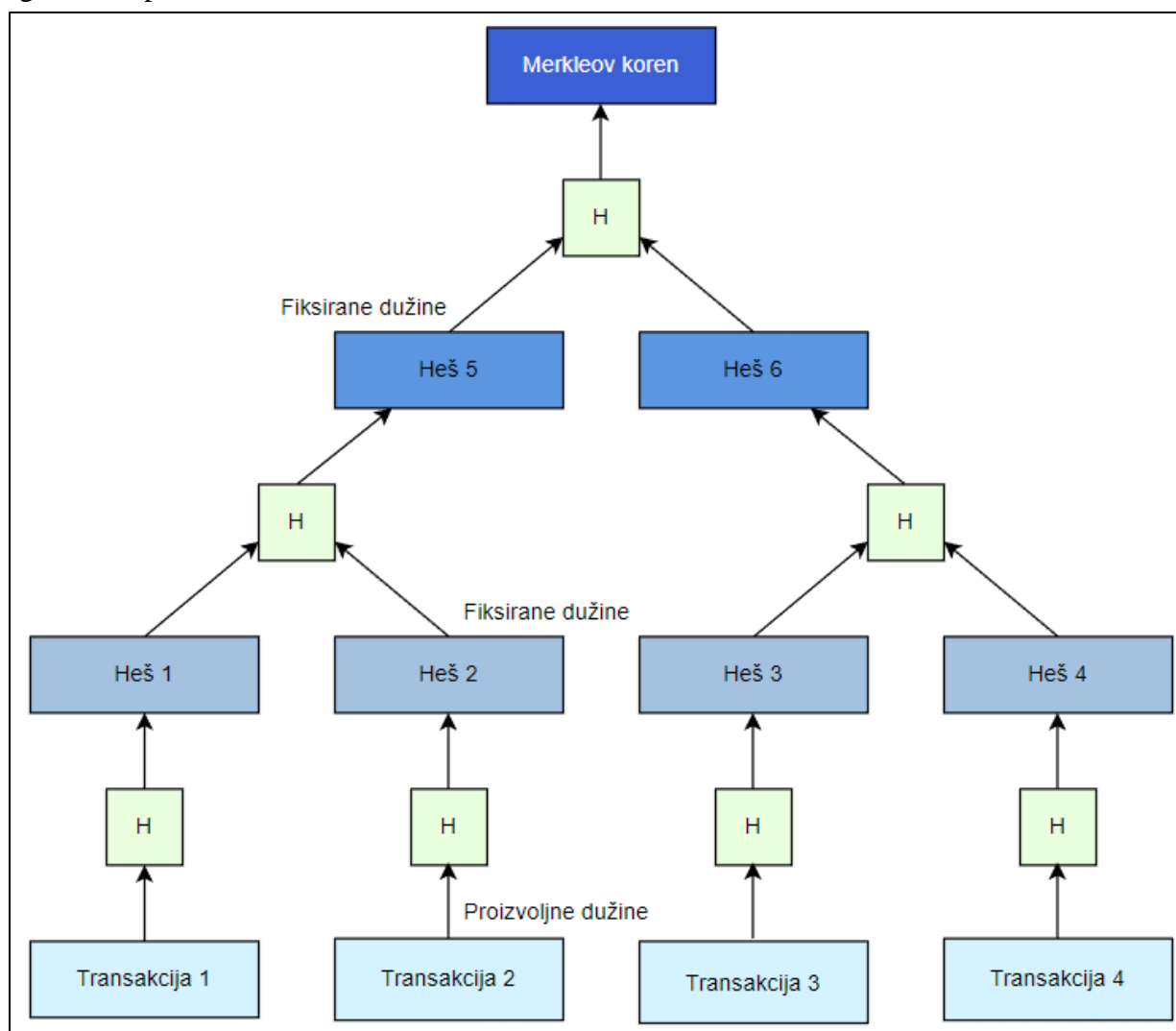
Ovo nije savršen bezbednosni mehanizam, zbog toga što je izlaz uvek identične veličine, a ulaz proizvoljne, uvek će se više ulaza preslikavati u jedan isti izlaz. Na primer, to znači da bi korisnici tehnički mogli da pristupe svom nalogu i sa pogrešnom lozinkom koja daje istu heš vrednost kao i originalna lozinka, međutim šanse za ovo su matematički zanemarljive. Napomena: u većini sistema se dodaju i drugi bezbednosni mehanizmi kako bi smanjili šansu za gorepomenuti problem (npr. dodavanje *salt* vrednosti).



Slika 1. Jedan od mogućih načina korišćenja CHF-a za povezivanje blokova unutar blockchain-a

Upotreba *CHF* kod *blockchain*-a je sledeća: 1) Podaci iz trenutnog bloka se heširaju i ta heš vrednost se čuva u posebnom polju. 2) Heš vrednost iz prethodnog bloka je jedan od ulaznih podataka sledećeg bloka i na taj način se zapravo blokovi ulančavaju, stoga i naziv *blockchain* (lanac blokova)[1]. Pošto svaki blok ukazuje na svog prethodnika baš pomoću ove heš vrednosti, ukoliko bi se neki blok unutar lanca izmenio, ili ukoliko bi se dodao potpuno novi blok između već dva postojeća, morale bi ponovo da se računaju sve heš vrednosti blokova od tog mesta pa do najnovijeg bloka u lancu.

U stvarnim implementacijama dosta često postoje dva različita polja u koja se zasebno smeštaju ostali heširani podaci. Na primer može se iskoristiti Merkleovo stablo heševa za heširanje transakcija unutar bloka, a da se iz prethodnog bloka koristi samo finalna heš vrednost njegovog Merkleovog stabla umesto heša svih transakcija jednostavno konkateneranih zajedno radi dodatne sigurnosti i provere.



Slika 2. Primer načina funkcionisanja Merkleovog stabla

U autorovoj implementaciji iskorišćen je idejno sličan, ali jednostavniji mehanizam heširanja svih podataka prethodnog bloka u samo jedno polje, tj. ne postoji Merkleovo stablo za dodatno heširanje transakcija. Izbačeni su i veličina bloka, broj verzije i broj transakcija, jer se oni u glavnom koriste zbog provere pristiglih podataka nakon prolaska kroz mrežu kod stvarnih implementacija i radi dodatnih jednostavnih provera (da li navedena veličina bloka odgovara stvarnoj veličini, da li postoje sva potrebna polja u ovoj verziji lanca, da li navedeni broj transakcija odgovara stvarnom broju) za koje je autor smatrao da samo otežavaju razumevanje ključnih karakteristika *blockchain*-a. Struktura autorovog bloka se može videti na slici 5.

2.1.2. MAGIČNI BROJ

Magični broj (eng. *magic number*) predstavlja jedinstven identifikator *blockchain*-a[1]. Svaki blok unutar jednog lanca mora da ima isti magični broj, nezavisno od toga na kojoj mašini se nalazi. Na primer *Bitcoin*-ov magični broj je 0xD9B4BEF9 (tj. 3652501241 u decimalnom obliku), dok je nasumično izabran magični broj autorovog izmišljenog *blockchain*-a 74567.

2.1.3. VREMENSKI ŽIG

Vremenski žig (eng. *timestamp*) je još jedan veoma bitan deo *blockchain*-a koji služi za vremensko datiranje stvaranja bloka i potvrdu da sadržaj bloka (transakcije, potpisi dokumenata, itd.) nije nastao nakon naznačene vremenske oznake unutar bloka. Pruža i dodatnu sigurnost u vidu potvrđivanja redosleda blokova (ukoliko se krećemo od najnovijeg ka najstarijem bloku unutar lanca, sve vremenske oznake moraju biti u opadajućem redosledu). U autorovoj implementaciji radi jednostavnosti je korišćen autoinkrementirajući identifikacioni broj bloka koji ima sličnu svrhu.

2.1.4. SADRŽAJ BLOKA

Svaki blok (osim u nekim slučajevima prvog – *genesis* bloka) pored svih ostalih pomoćnih podataka mora sadržati i neku informaciju u sebi. U autorovom primeru sadržaj bloka su transakcije između korisnika koji koriste izmišljenu kriptovalutu.

U stvarnim implementacijama najčešći sadržaj i jesu transakcije kriptovaluta, ali postoje i druge vrste sadržaja: ostale ekonomske radnje (npr. novčane transakcije, kupovina i prodaja deonice, osiguranje, ...), identifikovanje i autentifikacija objekata ili korisnika, praćenje trenutnog vlasništva objekata, decentralizovani registri podataka, brojevi glasova, itd. [1]. Međutim, bitno je napomenuti da je veliki broj *blockchain* aplikacija koje implementiraju prethodne vrste sadržaja još uvek u eksperimentalnoj fazi ili fazi razvoja.

U opštem slučaju, sadržaj bloka može biti bilo kakva informacija ili skup informacija, ali za većinu sistema nema potrebe čuvati podatke u *blockchain* strukturi, jer je neefikasna za

pretraživanje i nije potreban (ili dozvoljen) javni decentralizovani uvid u podatke.

2.1.5. DIGITALNI POTPIS

Digitalni potpisi (eng. *digital signatures*) predstavljaju način za dokazivanje autentičnosti poruke (informacije) i njenog autora. Mogu biti sa i bez trećeg lica (arbitra) koji verifikuje potpis i poruku i mogu se koristiti za kreiranje elektronskih potpisa (smatraju se najsigurnijom formom elektronskih potpisa)[4]. Digitalni potpisi se implementiraju korišćenjem kriptografski asimetričnih algoritama. Osim što je nemoguće lažno reprodukovati digitalni potpis, on takođe može da potvrdi da je autor zaista potpisao dokument čak i kada autor negira da je to uradio. Neke od popularnih primena gde se koriste ovi potpisi su: *online* transakcije, *email*, *FTP* servisi, *VPN*.

U Srbiji zakonski kvalifikovani elektronski potpisi se formiraju korišćenjem nekog od sledeća tri algoritma: *RSA* (*Rivest Shamir Adleman*), *DSA* (*Digital Signature Algorithm*), *ECDSA* (*Elliptic Curve Digital Signature Algorithm*); sa propisanim neophodnim minimalnim parametrima[5].

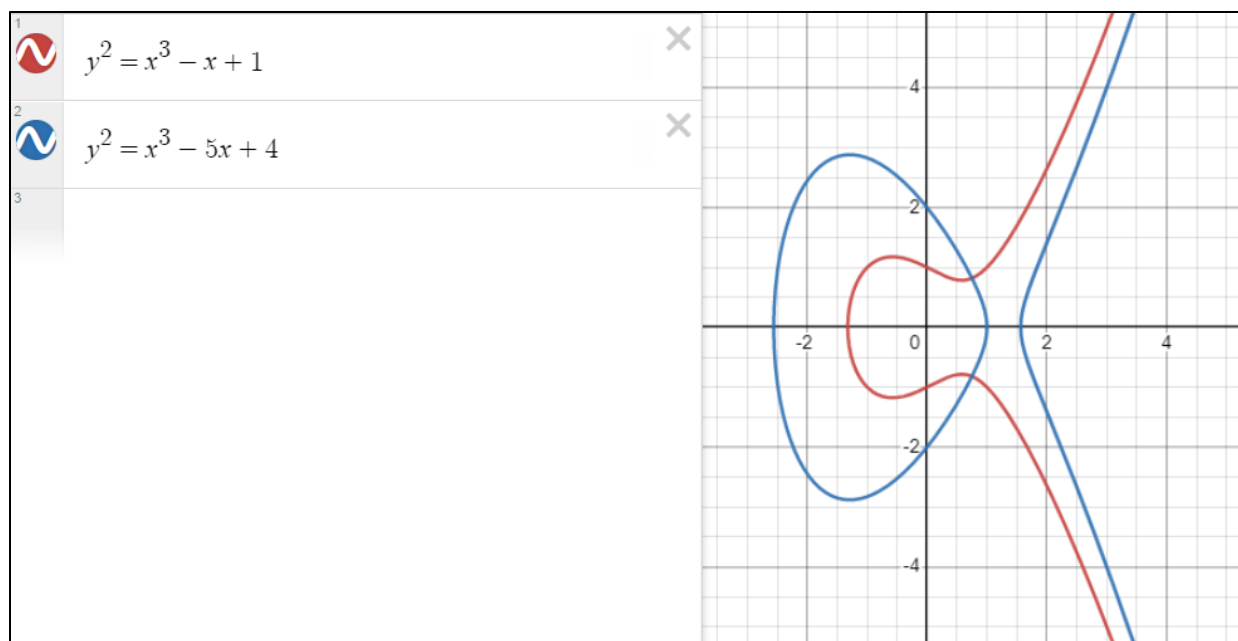
Pošiljalac i primalac poseduju svoje privatne i javne ključeve (eng. *private and public keys*). Javni ključ se izvodi matematički na osnovu privatnog, ali je nemoguće dobiti privatni poznavajući javni (ideja slična heširanju, mada je matematički pristup drugačiji – koristi se teorija brojeva i prosti brojevi).

Ukoliko primalac poseduje javni ključ pošiljaoca on može garantovano tvrditi da je poruka potpisana privatnim ključem pošiljaoca. Pošiljalac može dodatno enkriptovati poruku javnim ključem primaoca i time zagarantovati da samo autentični primalac može da pročitati poruku dekriptovajući je svojim privatnim ključem. Ovime je omogućeno obostrano poverenje čak i kada se poruka šalje kroz nepoznatu i nebezbednu sredinu (internet).

Autor u svojoj implementaciji koristi *ECDSA* algoritam za digitalno potpisivanje transakcija.

2.1.5.1. ECDSA

ECDSA spada u grupu algoritama koji se često sreću u kriptografiji, a zasnovani su na jednačinama eliptičke krive (oblika $y^2 = x^3 + ax + b$). *ECDSA* je često korišćen kao algoritam za enkripciju *online* komunikacije između servera i web browsera zbog svoje brzine u poređenju sa drugim sličnim algoritmima, jer koristi ključ od samo 160bitova[6]. Koristi se i kod kriptovaluta gde su korisničke adrese u stvari javni ključevi (npr. *Bitcoin* i autorova implementacija).



Slika 3. Dva primera izgleda eliptičke krive

Sam algoritam je matematički relativno komplikovan, i pošto nije tema ovog rada, nećemo previše duboko zalaziti u njega. Ispod je dat površan primer generisanja ključeva i provere validnosti poruke na osnovu njih pomoću ovog algoritma[7]:

- 1) Recimo da Ana želi da pošalje potpisanu poruku Branku. Pre same komunikacije oni moraju da ustanove tri unapred poznata parametra: jednačinu krive i polje nad kojim je definisana C , baznu tačku krive u okviru grupe reda nekog prostog broja G i n koje predstavlja multiplikativni red od G i koje na osnovu Bezuove teoreme mora biti prost broj.
- 2) Ana kreira privatni ključ d_A nasumičnim izborom celog broja između 1 i $n - 1$ i javni ključ izborom tačke na krivoj $Q_A = d_A \times G$, gde \times predstavlja skalarno množenje tačke na krivoj.
- 3) Ana potpisuje poruku m prateći sledeće korake:
 - 3.1) Računa $h = \text{HASH}(m)$, gde je HASH unapred poznata heš funkcija i pretvara izlaz u decimalni ceo broj.
 - 3.2) Uzima L_n najviših bitova od h i naziva ih l , gde L_n predstavlja dužinu grupe reda n u bitima.
 - 3.3) Generiše kriptografski sigurno i nasumično k koje predstavlja ceo broj između 1 i $n - 1$.
 - 3.4) Računa tačku na krivi $(x_l, y_l) = k \times G$.
 - 3.5) Računa $r = x_l \bmod n$. Ukoliko dobije da je $r = 0$, vraća se na korak 3.3.
 - 3.6) Računa $s = k^{-1} \cdot (l + r \cdot d_A) \bmod n$. Ukoliko dobije da je $s = 0$, vraća se na korak

3.3.

3.7) Potpis je uređeni par (r, s) . (Validan potpis je i uređen par $(r, -s \bmod n)$ zbog geometrije izabrane krive).

Napomena: Veoma je bitno da se za svaki potpis generiše različito k u tački 3.3, jer u suprotnom je moguće rešiti jednačinu 3.6 po d_A , ukoliko napadač ima pristup više od jednoj poruci. U decembru 2010. god. je zbog ovog propusta Sony-jev *PlayStation3* bio hakovan, omogućavajući napadačima kompletnu kontrolu nad operativnim sistemom, a avgusta 2013. nekoliko *Android* aplikacija za *Bitcoin* transakcije zbog nepravilne implementacije unutar *Java* klase *SecureRandom* koja je ponekad generisala koliziju za k vrednosti, što je omogućilo napadačima da se lažno predstavljaju ukradenim privatnim ključevima.

4) Branko dobija poruku m , Anin javni ključ Q_A i potpis (r, s) . Proverava validnost potpisa za tu poruku prateći sledeće korake:

4.1) Proverava da Q_A nije jednako jediničnom elementu O , da Q_A zaista leži na krivoj i da $n \times Q_A = O$. Ukoliko nešto od prethodnog nije ispunjeno zna da Anin potpis nije validan i ne može da veruje da je poruku zaista dobio od nje.

4.2) Proverava da su r i s celi brojevi između 1 i $n - 1$. Ukoliko nisu potpis nije validan.

4.3) Računa h i l isto kao u 3.1 i 3.2.

4.4) Računa $u_1 = l \cdot s^{-1} \bmod n$ i $u_2 = r \cdot s^{-1} \bmod n$.

4.5) Računa tačku na krivoj $(x_2, y_2) = u_1 \times G + u_2 \times Q_A$. Ukoliko je $(x_2, y_2) = O$ potpis nije validan.

4.6) Potpis je validan ako i samo ako je $r \bmod n = x_2 \bmod n$. U suprotnom nije validan i ne može verovati da je poruku zaista dobio od Ane.

2.1.6. NONCE I TEŽINA

Nonce (*number once*), u prevodu broj sa jednom svrhom, i težinu (eng. *difficulty*) imaju blokovi u *blockchain* sistemima koji kao metod validacije blokova koriste dokaz radom (eng. *proof of work*)[1]. Najpoznatiji takav sistem jeste *Bitcoin*, i autor je u svojoj implementaciji takođe koristio tu tehniku dokaza validnosti blokova.

Težina predstavlja broj na osnovu kog se računa neophodan broj prefiksni nula heš vrednosti celog bloka[2]. Autor je radi jednostavnosti i lakšeg razumevanja implementirao jedan na jedan preslikavanje između težine i broja prefiksni nula, ali u opštem slučaju postoji funkcija zavisnosti i težina može biti predstavljena čak brojevima sa decimalom. Menjanjem težine se u zavisnosti od raspoložive procesorske snage svih rudara eksplicitno određuje očekivana vremenska razlika između dodavanja novih blokova.

Nonce je broj čija je jedina svrha da kad se hešira zajedno sa ostatkom bloka proizvede

validnu heš vrednost, tj. heš vrednost koja započinje sa onoliko nula koliko je odredila težina bloka[2]. Novi blok se dodaje tek kada se pronade *nonce* koji ovo omogućava, a pronalaženje takvog *nonce*-a se naziva rudarenje (eng. *mining*) u *proof of work* sistemima.

2.1.7. COINBASE

Poslednji deo *blockchain*-a kao strukture podataka koji je bitan za kriptovalute jeste *coinbase*. *Coinbase* predstavlja transakciju koja nema pošiljaoca, već samo primaoca koji je u stvari čvor koji je validirao ovaj blok i dodao ga u lanac. To je zapravo nagrada koju validator dobija za uspešnu validaciju bloka i svaki validator prilikom validacije stavlja samoga sebe kao primaoca[2].

Novi novčići jedino ovako ulaze u sistem. Većina kriptovaluta ima limiran broj tokena koji može da bude u sistemu, i ta ograničenost im daje vrednost i sprečava inflaciju (za razliku od tradicionalnog novca koji nema gornju količinsku granicu). Ograničenost se postiže tako što je na svakih N blokova *coinbase* vrednost smanjena.

Primer *Bitcoin*-a: *Coinbase* nagrada se prepolovi na svakih 210,000 blokova[11]. Nagrada na početku *Bitcoin*-a za validaciju bloka je bila čak 50 *BTC*. Sada možemo da izračunamo ukupnu količinu tokena koji može da uđe u sistem preko beskonačne sume:

$$\begin{aligned} 210,000 \cdot 50 \text{ BTC} + 210,000 \cdot 25 \text{ BTC} + 210,000 \cdot 12.5 \text{ BTC} + \dots = \\ 210,000 \cdot 50 \text{ BTC} \cdot \left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right) = \\ 210,000 \cdot 50 \text{ BTC} \cdot \sum_{n=0}^{\infty} \left(\frac{1}{2}\right)^n = \\ 210,000 \cdot 50 \text{ BTC} \cdot 2 = 21,000,000 \text{ BTC} \end{aligned}$$

Do januara 2023. godine je u cirkulaciju ušlo 19,300,000 *BTC*, a prognozira se da će se do maksimalnog kapaciteta doći oko 2140. godine, jer su procesori naravno limitirani u pogledu najmanjih vrednosti sa kojima mogu da rade, te suma naravno nije beskonačna. Nakon što *Bitcoin* dođe do gornje granice količine tokena, plan je da se izbací *coinbase* nagrada, a da rudari budu nagrađivani transakcijskim troškovima koji se uzimaju od pošiljaoca/primaoca[8].

2.2. KAO MREŽNI PROTOKOL

Ledger-i su neophodna stavka za finansijsku reviziju u svakoj organizaciji. U tradicionalnom digitalnom formatu *ledger* se čuva na centralnom serveru dok su ostalim čvorovima (korisnicima na sopstvenim računarima) (eng. *nodes*) u mreži dodeljene privilegije za čitanje/pisanje (eng. *read/write privilege*)[1]. Ovo naravno podrazumeva da postoji čitav bezbednosni mehanizam unutar sistema koji bi korisnike autentifikovao i autorizovao i zatim

obezbedio siguran pristup samom centralnom serveru koji bi zapisavao dobijene transakcije i čitao zatražene u/iz *ledger*-a.

Mane ovako opisanog centralizovanog sistema su: ne tako dobra skalabilnost (sa rastom broja čvorova u mreži opada svima brzina pristupa *ledger*-u), *single point of failure* (ako centralni server postane nedostupan iz bilo kog razloga onda niko neće imati pristup *ledger*-u), zavisnost od centralnog autoriteta (sadržaj *ledger*-a i pristup njemu zavise od „dobre volje“ onoga ko kontroliše centralni server).

Alternativan sistem se naziva siguran distribuirani *ledger* (eng. *secure distributed ledger* - *SDL*)[1]. Siguran znači da se podaci ne smeštaju u jedinstvenu bazu podataka već u nadogradivu *blockchain* strukturu koja onemogućava kasnija menjanja već postojećih podataka za razliku od centralnog servera. Distribuiran znači da ne postoji centralni autoritet koji nadgleda transakcije i određuje ko ima kakvu vrstu pristupa *ledger*-u već se transakcije i njihova validnost verifikuju postizanjem konsenzusa u mreži, tj. potrebno je da se većina čvorova složi da je transakcija validna pre nego što se ona izvrši. Kao što je već spomenuto svaki korisnik *SDL* sistema (u nastavku teksta se podrazumeva da se misli na *SDL* sistem kad se kaže *blockchain* protokol) ima zasebnu kopiju čitavog lanca na svom lokalnom uređaju (ovo naravno ne važi baš u svakom trenutku zbog slanja podataka kroz mrežu).

Blockchain tako rešava sve tri mane centralizovanog sistema, ali kreira i neke nove: neefikasno korišćenje ukupne računarske memorije i snage (otprilike broj čvorova puta više nego kod centralizovanog sistema), pretraga podataka je spora, jer mora da se ide unazad kroz lanac od bloka do bloka (morale bi da se koriste dodatne strukture podataka za poboljšanje pretrage, što bi dovelo do još veće neefikasnosti korišćenja ukupne memorije), novi korisnici moraju da skinu celu kopiju lanca prilikom prvog pristupa u *blockchain* mrežu (npr. trenutno 435GB kod *Bitcoin*-a i nastaviće da se povećava).

Kasnije u ovom poglavlju ćemo videti kako je moguće da anonimna lica unutar decentralizovanog sistema dođu do konsenzusa i koji su to mogući načini validacije ispravnosti lanaca.

2.2.1. TIPOVI BLOCKCHAIN-A

Postoje različiti tipovi *blockchain*-a, tako da ćemo u ovom poglavlju proći kroz grubu podelu na tri najosnovnija[1].

2.2.1.1. JAVNI

Javni *blockchain* je ubedljivo najčešća verzija *blockchain*-a. Njemu može da pristupi bilo ko preko interneta. Smatraju se da su zaista u potpunosti decentralizovani, jer ne postoji centralni autoritet koji dodeljuje dozvole čvorovima za pristupanje *ledger*-u ili za njegovo dopunjavanje

novim transakcijama. Prednosti su transparentnost podataka i laka revizija, međutim u to ime se žrtvuje privatnost podataka i generalno imaju veću ukupnu potrošnju energije za održavanje ovakvog sistema. *Bitcoin* je najpoznatiji primer javnog *blockchain*-a. Autorova implementacija se takođe zasniva na javnoj verziji *blockchain*-a.

2.2.1.2. PRIVATNI

Privatni *blockchain* dozvoljava samo entitetima sa autoritetom ili vlasnicima da dodaju nove transakcije u okviru lanca (npr. jedna kompanija). Dozvole za čitanje podataka mogu biti javne (dostupno svima na internetu) ili privatne (vlasnik dodeljuje dozvole čvorovima kojima on želi). Privatni *blockchain* je druga najpopularnija verzija *blockchain*-a ali dosta zaostaje za javnim. Ne smatraju se decentralizovanim sistemima.

2.2.1.3. KONZORCIJUM

Kod konzorcijuma unapred izabrani skup čvorova odobrava transakcije. Na primer 9 različitih firmi učestvuju u istoj *blockchain* mreži od kojih transakciju mora da validira barem 6 da bi se ona dodala u lanac. Postoje različite varijacije gde čitanje i pisanje mogu biti javni ili privatni. Ne smatraju se potpuno decentralizovanim sistemima.

2.2.2. KONSENZUS ALGORITMI

Najbitnije pitanje kod distribuiranog decentralizovanog sistema jeste kako zagarantovati da svi korisnici mogu da veruju u tačnost informacija koje dobijaju bez centralnog autoriteta koji bi verifikovao transakcije.

Sa tačke gledišta jednog novog korisnika nekog *blockchain* sistema, on je ušao u mrežu gde ne zna sa koliko drugih čvorova komunicira, niti ko su, a zapravo mu je zagarantovano da može da veruje informacijama koje dobija od ostatka mreže. To se postiže uz pomoć konsenzus algoritama, tj. algoritama kojima se umreženi čvorovi usaglašavaju oko validnosti informacija unutar sistema.

Postoji dosta različitih konsenzus algoritama, a u ovoj glavi ćemo proći kroz najrasprostranjeniji koji je i implementiran u autorovoj aplikaciji, dok se ostali mogu pronaći u prilogu u poglavlju 9.1[12].

2.2.2.1. PROOF OF WORK

Najpoznatiji, najkorišćeniji i najstariji konsenzus mehanizam (algoritam) jeste dokaz radom (eng. *Proof of Work – PoW*). Njega koristi *Bitcoin*, kao i autorova implementacija, a teorija iza njega postoji već tridesetak godina.

Kod ovog mehanizma posebni čvorovi unutar mreže koji validiraju blokove se zovu rudari

(eng. *miners*). Njihov zadatak jeste da pronađu *nonce* vrednost koja daje odgovarajuću heš vrednost kad se hešira zajedno sa ostatkom bloka. Pronalaženje te vrednosti se naziva rudarenje (eng. *minning*).

Rudar koji prvi pronađe jedno moguće *nonce* rešenje kao nagradu dobija *coinbase* svotu kriptovalute. Rudari imaju inicijativu da ulože u što više hardverske opreme za rudarenje (npr. grafičke kartice – ovo je razlog zašto su dosta poskupele od kada je kripto rudarenje postalo popularno), jer time povećavaju šanse da baš oni budu ti koji su pronašli odgovarajuću *nonce* vrednost.

Zbog atributa koje poseduje *CHF*, *nonce* vrednost koju je rudar teško pronašao se lako validira od strane ostatka mreže.

Vremenom kako ukupna procesorska snaga mreže raste tako raste i težina matematičkog problema (*difficulty* raste), tj. smanjuje se šansa pronalaska validnog *nonce*-a i tako se srednje vreme dodavanja novog bloka održava konstantnim[8].

Rudari se mogu udružiti u veće grupe zvane bazeni (eng. *pools*) i da se ravnomerno svakom članu srazmerno njegovoj procesorskoj snazi dodeljuje *coinbase* ukoliko neki rudar iz tog *pool*-a reši blok.

Mane *PoW*-a:

- 51% napad: Ukoliko jedan entitet poseduje više od polovine procesorske snage u mreži, onda je moguće da zloupotrebi algoritam u svoju korist, tako što će matematički on biti taj koji najviše transakcija validira i može da u njih unese, npr. nepostojeće transakcije zarad lične koristi. Ovaj napad postoji i kod većine drugih konsenzus algoritama, samo sa drugačijim preduslovima.
- Vremenski i resursno zahtevno: Rudari moraju da potroše dosta vremena i resursa (električne energije, hardver) da bi validirali transakcije (tj. izrudarili blok). Ovo je glavni razlog zašto novije kriptovalute ne koriste ovaj algoritam, jer sistem postaje ekonomski neisplativ tokom vremena (težine blokova su sve veće i to zahteva sve više resursa, a nagrade za njihovo rešavanje su sve manje).
- Transakcije su spore: Zbog spore validacije transakcije nisu instantne. Proces potvrđivanja transakcije u glavnom traje između 10 i 60 minuta trenutno za *Bitcoin*[9].

2.2.3. BLOCKCHAIN FORKING

Decentralizovanost sistema znači da korisnici moraju da dođu do konsenzusa oko trenutnog stanja lanca. Ukoliko se svi čvorovi slože onda postoji samo jedna verzija lanca, međutim to često nije slučaj.

Zbog toga što je potrebno vreme da informacije prođu kroz mrežu moguće je da se čvorovi ne slažu jednoglasno oko toga kako lanac trenutno izgleda. Tada dolazi do razdvajanja *blockchain*-a u više paralelnih lanaca istovremeno i ovo se naziva živo slučajno/trenutno razdvajanje (eng. *live accidental/temporary fork*). Ovo se veoma često dešava u *blockchain* sistemima, ali se lako razrešava tako što čvorovi preuzimaju tok lanca od sledećeg rudara.

Živo namerno razdvajanje (eng. *live intentional fork*). može da bude meko (eng. *soft*) i tvrdo (eng. *hard*)[10]. Meko označava da je nova verzija *blockchain*-a kompatibilna unazad (eng. *backwards compatible*), tj. da su uvedena nova strožija pravila u sistemu i da su novi blokovi smatrani validnim od strane starijih verzija softvera (npr. uvedene nove adrese). Tvrdo razdvajanje označava suprotnu stvar, tj. uvedena su nova pravila koja su u suprotnosti sa prethodnim pravilima, i čvorovi koji pređu na novu verziju sistema ne mogu da interaguju sa čvorovima koji su na staroj (npr. kada je Ethereum prešao sa *PoW*-a na *PoS*).

Postoji i *codebase fork* koji predstavlja mogućnost da dođe do razdvajanja lanca prilikom izlaska potpuno nove verzije softvera zasnovane na nekom već postojećem *blockchain* sistemu (npr. nova kriptovaluta zasnovana na *Bitcoin*-u, sa minimalnim izmenama sistema), ali to nećemo detaljno razmatrati.

3. IMPLEMENTACIJA

Autor je odlučio da implementira sistem za vizuelizaciju *blockchain* tehnologije koristeći popularne web tehnologije, jezike i njihove okvire (eng. *frameworks*) – *HTML*, *CSS* i *Javascript* zbog toga što pružaju jednostavnost i automatizaciju kreiranja tekstualnih formi koje čine srž sistema.

3.1. PREDNOSTI I MANE U ODNOSU NA DRUGA MOGUĆA REŠENJA

Tekstualne forme u vidu web aplikacije su izabrane pošto je glavni cilj rada granularna vizuelna reprezentacija i autor je smatrao da je najbolje prikazati rad *blockchain*-a pomoću interaktivnog sadržaja kojim su predstavljeni podaci unutar svakog polja izmišljenog *blockchain* sistema sa kojim korisnici web aplikacije mogu da interaguju i da vide u realnom vremenu kako izmene u jednom bloku utiču na ostale podatke unutar istog bloka kao i čitavog lanca.

Tehnologije bazirane na webu su izabrane zbog lakog pristupa mnoštvu javno-dostupnih biblioteka (na primer za računanje heš funkcija i digitalnih potpisa), jednostavnosti korišćenja aplikacije i veoma širokog spektra mogućeg dizajna korisničkog interfejsa koji će se kasnije lako uniformno proširiti na svaki deo sistema, što je od izuzetne važnosti za intuitivno korišćenje aplikacije od strane korisnika bez prethodnog predznanja o *blockchain* tehnologiji.

Ovakva struktura je takođe omogućila inkrementalno građenje *blockchain*-a od nule i lak pristup aplikaciji preko web čitača, što znači da nisu potrebne nikakve dodatne instalacije alata ili programa da bi korisnici aplikacije mogli da joj pristupe.

Dakle glavne prednosti ovog pristupa su prosta i uniformna automatizacija kreiranja gradivnih blokova koji čine aplikaciju, jednostavan, ali moderan i intuitivan korisnički interfejs, lakoća pristupa aplikaciji od strane korisnika (ne postoje alati ili programi koje je prethodno potrebno instalirati osim web čitača i moguć je pristup aplikaciji čak i preko mobilnog uređaja), granularnost i interaktivnost sa elementima koji čine *blockchain*.

Mana izabranog tehničkog rešenja jeste slaba proširivost aplikacije, mada ona više ima veze sa izabranim pristupom organizaciji koda, nego sa izabranim tehnologijama.

Druga moguća razmatrana rešenja su (od najmanje do najviše idealnog prema autoru): konzolna aplikacija simulacije *blockchain*-a napisana u nekom od klasičnih OOP jezika, konzolna aplikacija sa interaktivnim tekstualnim prikazom *blockchain* polja i blokova napisana u nekom od klasičnih OOP jezika, web aplikacija simulacije *blockchain*-a napisana koristeći moderne okvire poput *React*-a ili *Vue*-a, vizuelna aplikacija sa interaktivnim vizuelnim prikazom *blockchain* polja i blokova u vidu formi napisana u *C#*-u ili *Java*-i (*WinForms* ili *JavaFX*).

Konzolne aplikacije pored očiglednog nedostatka vizuelizacije i težine korišćenja za

korisnike imaju i drugu manu, a to je manjak fleksibilnosti u radu, tj. previše su linearne. Zbog toga je autor smatrao da ne predstavljaju dobar osnov za implementaciju prikaza rada *blockchain*-a. Njihova jedina prednost je laka implementacija osnovnih elemenata, međutim manjak dostupnih biblioteka bi otežao implementaciju komplikovanijih algoritama koji se koriste (heš funkcije, a pogotovo ECDSA).

Web aplikacija simulacije *blockchain*-a za glavnu manu ima težinu same implementacije, podrazumevajući korišćenje veoma velikog broja sporih i kompleksnih biblioteka i kompleksnih okvira za koje je autor smatrao da nisu potrebni za implementaciju izabrane vrste rešenja. Prednosti bi bile veća fleksibilnost nego trenutno rešenje, sa dodatnim mogućim modulima – npr. mrežna simulacija napada na *blockchain* sistem ili mrežna simulacija prenosa i validiranja blokova među čvorovima. Autor smatra da su ovo mogući sledeći koraci za doradu aplikacije.

Aplikacije u vidu formi predstavljaju veoma sličnu ideju kao i implementirano rešenje, ali su takođe manje fleksibilne, sa manje dostupnih biblioteka i gorim korisničkim interfejsom. Njihova prednost bi bila lakša nadogradivost koda, međutim autor nije smatrao da je to od krucijalnog značaja za unapred jasno definisanu veličinu i odlike projekta.

3.2. KORIŠĆENE TEHNOLOGIJE

3.2.1. HTML

HyperText Markup Language – *HTML* je jednostavan platformski nezavisan jezik za označavanje (eng. *markup language*) dokumenata koji su namenjeni za prikazivanje u web čitačima (eng. *browser*). Ti dokumenti se nazivaju web stranice.

Markup language je sistem za šifriranje teksta (eng. *text-encoding*) koji se sastoji od simbola koji su ubačeni u tekstualni dokument kako bi se kontrolisala njegova struktura, formatiranje i odnos između njegovih različitih delova. Ovi simboli se ne prikazuju kao deo teksta.

HTML semantički opisuje strukturu web stranice pomoću *HTML* elemenata zvanih tagovi koji se navode između karaktera < i >. Web *browser*-i ne prikazuju tagove u obliku teksta već pomoću njih struktuiraju web stranice. *HTML* fajlovi se završavaju ekstenzijom .htm ili .html.

Pomoću *HTML* elemenata se umeću ostale tehnologije poput *CSS*-a i *Javascript*-a u web stranice.

3.2.1.1. PUG

Pug (prethodno poznat kao *Jade*) je *Node*-ov generator šablona (eng. *template engine*). Kompajlira se u *HTML* i ima jednostavniju sintaksu. *Pug* fajlovi se završavaju ekstenzijom .pug. Slično jeziku Python, mora se obratiti pažnja na razmake jer ne postoje zatvarajući tagovi kao u *HTML* već se sadržaj koji je više puta tabuliran nalazi unutar sadržaja koji ga okružuje, a koji je

manje puta tabuliran.

Bitna odlika *Pug*-a jeste mogućnost da se u zasebnim fajlovima implementira jedna funkcionalost i onda lančano ubacuje sa dinamičkim parametrima u roditeljsku stranicu. Ovo je autor veoma često koristio prilikom implementacije web aplikacije.

3.2.2. CSS

Cascading Style Sheets – CSS je jezik koji služi za opis prezentacije dokumenta koji je napisan u *markup* jeziku (najčešće *HTML*). Dizajniran je tako da može da omogući razdvajanje sadržaja stranice od izgleda stranice tako što se smešta u posebne fajlove sa ekstenzijom *.css*.

Zbog veoma česte pojave standardizovan je među web čitačima. Određeni stil se odnosi samo na selektore koji su naznačeni u okviru CSS fajla. Ti selektori se mogu odnositi na: sve elemente naznačene posebnim tagom iz markup jezika, određene elemente označene specifičnim atributom (npr. *id* ili *class*), određene elemente u zavisnosti od toga gde su smešteni u odnosu na druge u okviru stabla dokumenta.

3.2.2.1. SASS

Syntactically Awesome Stylesheet – SASS je CSS preprocesor koji omogućava korišćenje odlika koje ne postoje u čistom CSS-u poput promenljivih, nasleđivanja, funkcija, ugnežđenih pravila, ...

SASS fajlovi se završavaju ekstenzijom *.scss*.

3.2.2.2. BOOTSTRAP

Bootstrap je *front-end CSS framework* koji prema statistici koristi svaki peti web sajt. Napravljen je pomoću SASS-a, a inicijalno je bio razvijen u okviru kompanije *Twitter* radi lakšeg osiguravanja konzistencije stilova između različitih alata. Autor je koristio *Bootstrap* za uređivanje prezentacije web aplikacije.

3.2.3. JAVASCRIPT

Kao glavni programski jezik prilikom razvijanja aplikacije korišćen je *Javascript*. *Javascript*-ovo radno okruženje (eng. *runtime environment*) *Node.js* je korišćen za uspostavljanje servera na kom se pokreće aplikacija, čist *Javascript* i *jQuery* okvir su korišćeni za *front-end* validaciju, raznolika računanja i upravljanje *HTML* elementima. *Node packet manager* je korišćen za upravljanje bibliotekama.

3.2.3.1. NODE.JS

Node.js predstavlja open-source serversko okruženje koje može da se izvršava na različitim platformama i operativnim sistemima. Omogućava *JavaScript*-u da se izvršava izvan web čitača i

dozvoljava programerima da pišu alate za komandnu liniju i serverske skripte u njemu. Ovo znači da i za front-end i back-end može da se koristi identičan jezik.

Iako je *JavaScript* jednonitni (eng. *single-threaded*) jezik, tj. nije moguće konkurentno izvršavanje više tokova kontrole programa, *Node.js* omogućava pravljenje skalabilnih i brzih servera koristeći pojednostavljeni model programiranja vođenog događajima (eng. *event-driven programming*) koji koristi pozive unazad (eng. *callbacks*) za označavanje završetka zadataka (eng. *tasks*). *Callback*-ovi su funkcije koje se prosleđuju drugim funkcijama kao argument i u njima pozivaju.

3.2.3.2. JQUERY

jQuery je *JavaScript* biblioteka koja je osmišljena da omogući lako kretanje i manipulisanje *HTML DOM* (*Document Object Model*) stabilom, i upravljanje događajima, *CSS*-om i *Ajax*-om. Predstavlja ubedljivo najšire korišćenu *JavaScript* biblioteku i nalazi se na više od 77% web stranica.

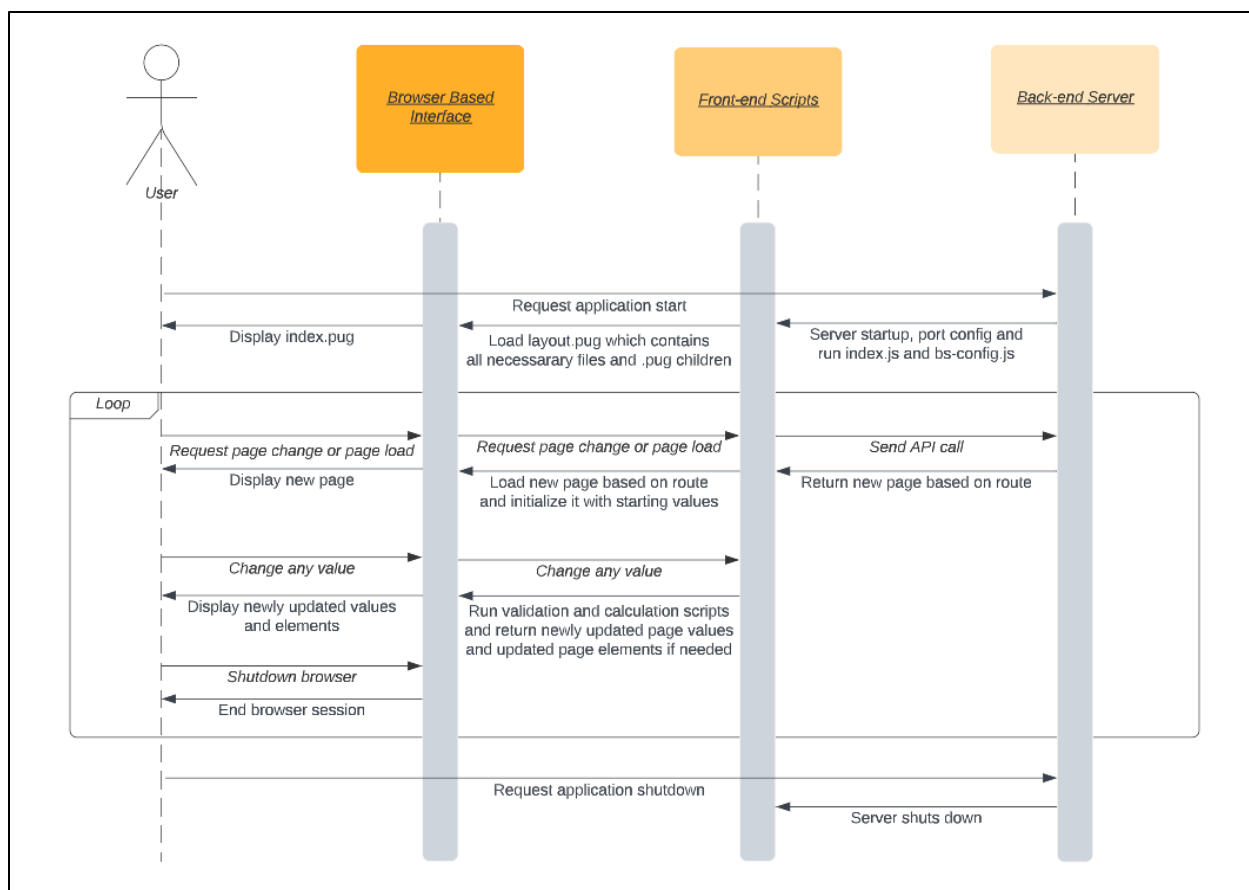
Omogućava lako odabiranje *HTML* elemenata i njihovo upravljanje (dodavajući im ponašanje za određene događaje ili menjajući im *HTML* atribut).

3.2.4. CAMTASIA 9

Camtasia 9 je najnovija instalacija (verzija softvera) u softverskom paketu *Camtasia* koju razvija *TechSmith*. Namenjena je prvenstveno za kreiranje ili snimanje video tutorijala i prezentacija. Ima veoma jednostavan korisnički interfejs što ga čini lakim za upravljanje i omogućava ubacivanje različitih objekata i anotacija što je i razlog zašto ga je autor izabrao za kreiranje jednostavne mrežne animacije *blockchain-a*.

3.3. ARHITEKTURA SISTEMA

Sistem je podeljen na prednji i zadnji deo (eng. front-end and back-end). Zadnji deo čini isključivo *JavaScript*, dok su na prednjem prisutni i *HTML* i *CSS* pored *JavaScript-a* (naravno sa svim potrebnim okvirima). Većinu funkcija obavlja *front-end* jer nema nikakve potrebe za čuvanjem podataka ili složenim kalkulacijama u ovoj verziji implementacije koje bi obavljao back-end, mada to bi moglo da se promeni kada bi se ovaj sistem dalje nadogradio.



Slika 4. Dijagram sekvence sistema

Kao što se vidi na slici iznad *back-end* je isključivo zadužen za pokretanje i zaustavljanje aplikacije, kao i za rutiranje stranica, dok sve ostalo obavlja *front-end*. Korisnik komunicira sa aplikacijom (osim pokretanja i zaustavljanja) kroz web browser. Aplikacija se startuje preko komandne linije, a prekida gašenjem trenutne sesije komandne linije.

3.4. ORGANIZACIJA FAJLOVA

Fajlovi su organizovani po *Pug* preporučenoj organizaciji *Node.js* projekta. Folder *node_modules* sadrži uvezene *Node.js* module i njihove metapodatke. Folder *public* sadrži korišćene ikonice, skripte, .css fajlove i video u zasebnim podfolderima. Folder *views* sadrži .pug fajlove. Celokupno stablo fajlova (koje dobijemo iz komandne linije komandom `tree /f /a`) dato je u prilogu u bloku koda 1.

3.4.1. METADATA FAJLOVI

Ovde spadaju .json fajlovi koje većinski *node packet manager* sam ažurira prilikom instalacije nekog od modula. Služe za praćenje zavisnosti između različitih modula. *package.json* je fajl koji opisuje direktno ovaj projekat i njegov izgled je dat je u prilogu u bloku koda 2.

.package-lock.json fajl radi identičnu stvar samo za uvezene module u okviru projekta.

3.4.2. INDEX.JS

index.js podiže web server pomoću *Node.js*-a. Ovaj fajl se pokreće prilikom pokretanja aplikacije i on prikazuje sve druge stranice na osnovu zadatih putanja za rutiranje koje dobija iz *layout.pug*-a na osnovu API poziva kroz *URL*. Izgled fajla moguće je pronaći u prilogu u bloku koda 3.

3.4.3. LAYOUT.PUG

Ovo je glavni roditeljski *.pug* fajl na koji se kasnije nadovezuju ostali. Predstavlja glavnu strukturu svake stranice i sadrži navigacioni meni i *footer* između kojih se umeću druge stranice u zavisnosti od rutiranja, tj. *URL*-a na kom se nalazimo. Takođe uključuje sve potrebne *.js* i *.css* fajlove koje koriste sve stranice. U blok *layout-content* se umeću sve druge *children* stranice što je moguće videti u prilogu u bloku koda 4.

3.4.4. BS-CONFIG.JS

bs-config.js je kratak konfiguracioni fajl za *BrowserSync* modul koji olakšava testiranje web aplikacije u toku razvoja automatizujući repetativne zadatke poput stalnog osvežavanja stranice. Sadrži jednostavne ključ-vrednost parove u *JSON* sintaksi, a njihova značenja se mogu pronaći na sledećem linku <https://browsersync.io/docs/options>. Sadržaj fajla dat je u prilogu u bloku koda 5.

3.4.5. GLOBALNE FUNKCIJE

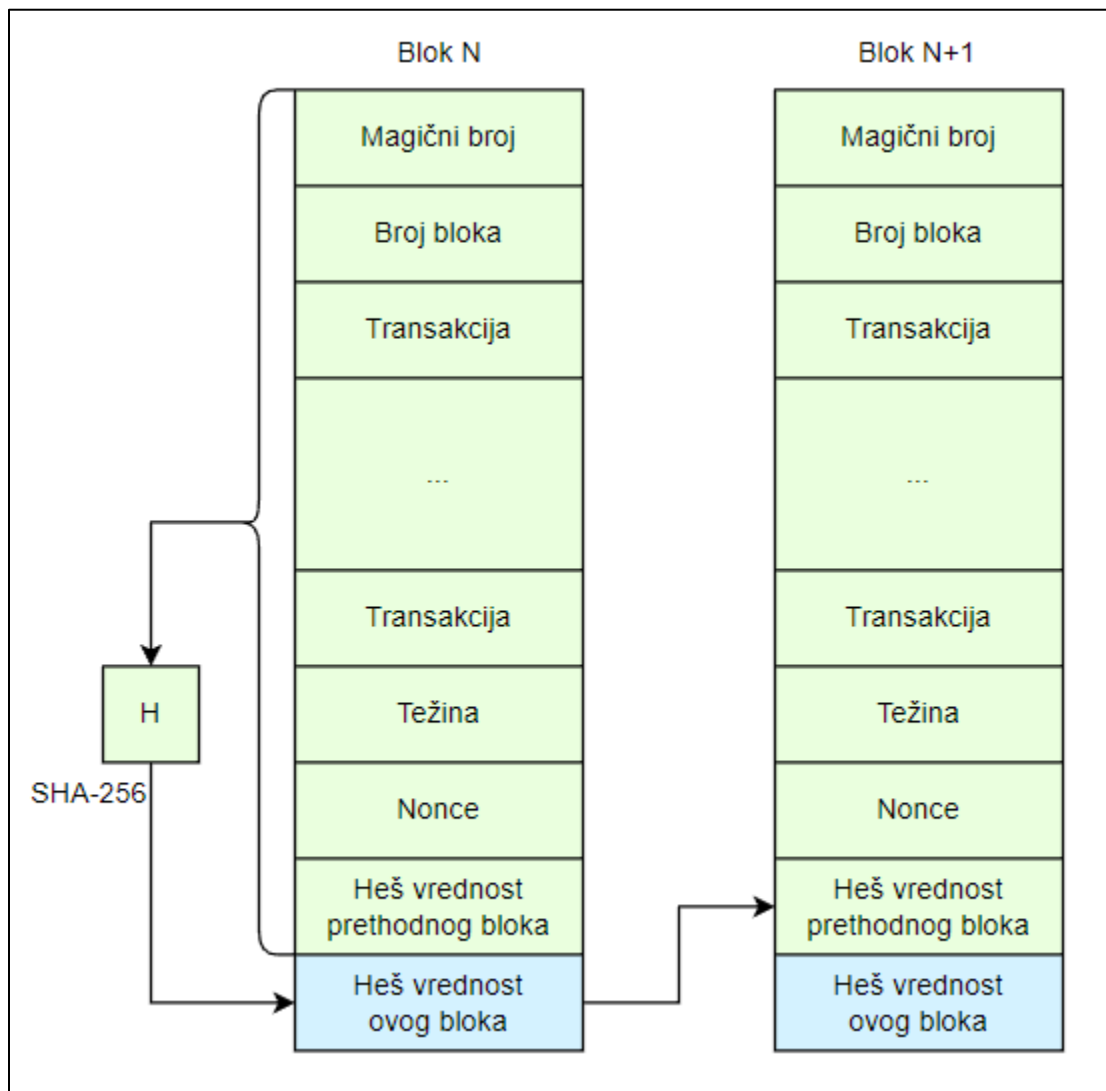
Autorove *JavaScript* funkcije se nalaze u folderu */public/javascripts*, dok se eksterno uvezene funkcije nalaze u */public/javascripts/lib*. Slična struktura važi i za *.css* fajlove. Sve funkcije i *.css* fajlovi su uključeni u *layout.pug* fajl, a kroz veliki deo njih ćemo proći u glavi 4.

4. PRIMER RADA SISTEMA

U ovoj glavi ćemo proći kroz rad čitave aplikacije od početka do kraja i kroz interaktivne primere dodatno objasniti rad *blockchain*-a, ali sada sa praktičnog aspekta.

Stranice unutar aplikacije za koje će biti priložene slike će biti stavljene u zasebna poglavlja, a za neke kraće funkcije će dodatno biti priloženi i delovi koda u prilogu pomoću kojih su prikazane funkcionalnosti implementirane.

Svaka stranica će uvoditi nove delove implementacije krajnjeg rešenja i kroz interakciju sa njima omogućiti lakše razumevanje karakteristika i svhre tih delova. Krajnje implementirani blok će izgledati ovako:



Slika 5. Struktura i povezanost susednih blokova u autorovoj implementaciji

4.1. POKRETANJE APLIKACIJE

Aplikacije se pokreće preko komandne linije iz *root* direktorijuma komandom `npm run dev`, gde `dev` u stvari predstavlja alijas za `nodemon ./index.js`, kako je definisano u fajlu `package.json`. Druga bitna komanda za pokretanje prilikom razvijanja aplikacije je `npm run ui`, gde je `ui` alijas za `browser-sync start --config bs-config.js`.

`dev` komanda pokreće lokalni server pokretanjem komandi definisanih u `index.js` fajlu, postavlja aplikaciju na njega i osluškiva port 8000, dok komanda `ui` konfiguriše aplikaciju tako da se prilikom njenog razvijanja (eng. *development*) automatski, bez potrebe za restartovanjem servera, najnovija verzija aplikacije nalazi na portu 8000 ili portu 3000 (koji je proxy za port 8000) prilikom svake promene *source* koda.

Aplikaciji se može pristupiti preko više *URL*-ova unutar web čitača:

- `http://localhost:3000/`
- `http://localhost:8000/`
- `http://<privatna IP adresa servera>:3000`

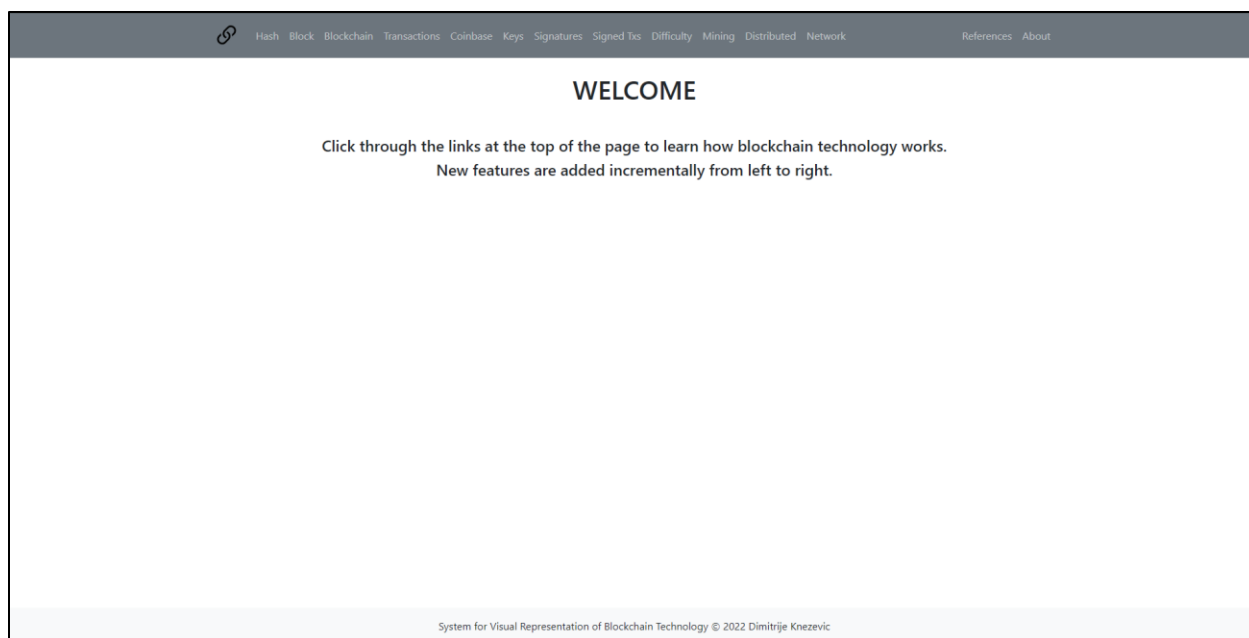
U poslednjem slučaju moguće je pristupiti aplikaciji i sa drugih uređaja unutar iste mreže kao što je prikazano na slici **Error! Reference source not found.** Zahvaljujući *Bootstrap*-u, *CSS* s tilovi su automatski rekonfigurisani za druge oblike ekrana.



Slika 6. Pristupanje aplikaciji sa mobilnog uređaja

4.2. INDEX.PUG

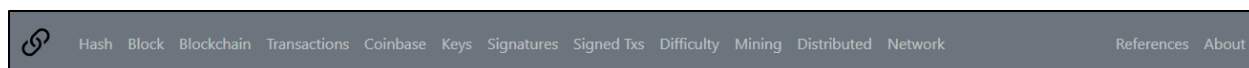
Početna (Indeks) stranica sa kratkim upustvima za korišćenje sajta data je na slici ispod.



Slika 7. Početna (Index) stranica

Njoj se ponovo može pristupiti klikom na ikonicu lanca u navigacionom meniju. Navigacioni meni pored te ikonice sadrži za svaku stranicu po jedan hiperlink.

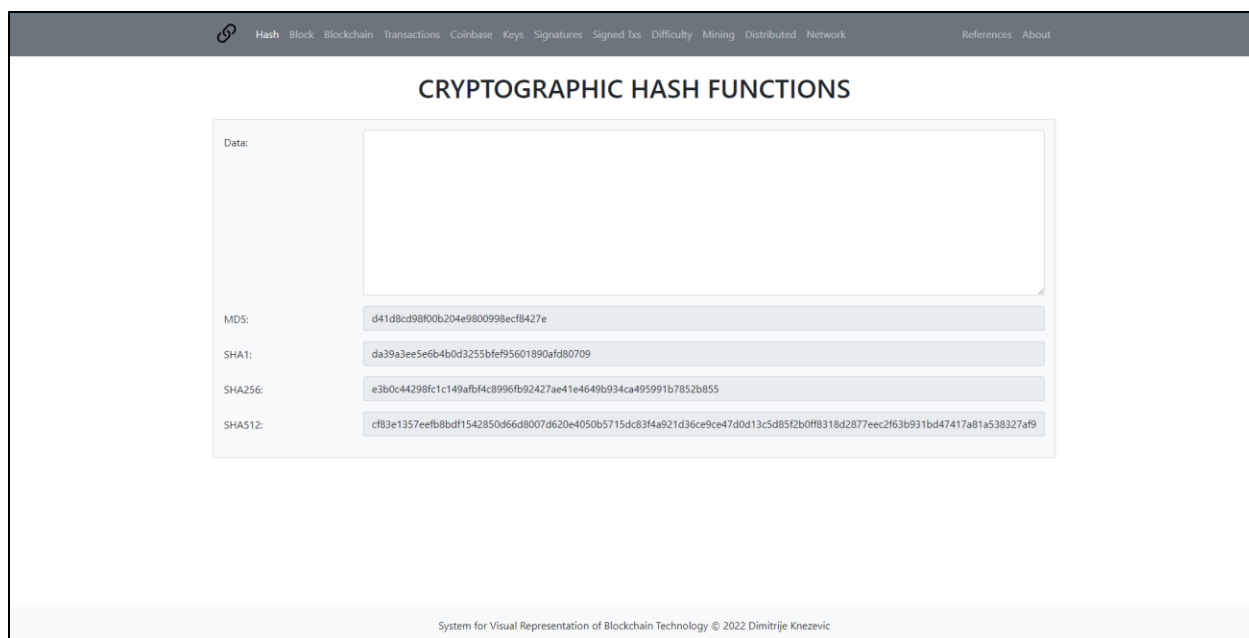
Aplikacija je zamišljena tako da se inkrementalno gradi čitav blokčejn sistem, dodavajući nove elemente na svakoj sledećoj stranici.



Slika 8. Navigacioni meni

4.3. CHF.PUG

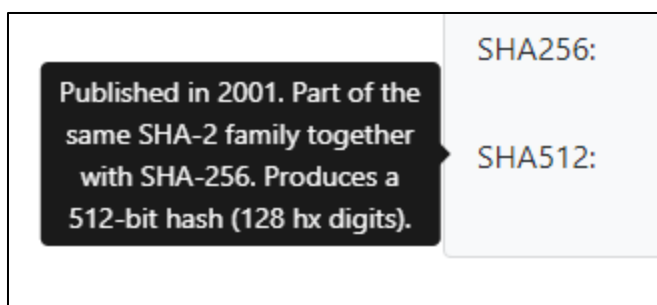
CHF je skraćenica za *Cryptographic Hash Functions*, tj. kriptografske heš funkcije. Ova stranica se sastoji od jednog *TextArea* elementa promenljive veličine koji korisnik može da koristi za proizvoljan tekstualni unos i 4 *TextBox*-a koja predstavljaju izlaz različitih *CHF* u stvarnom vremenu u zavisnosti od trenutnog korisničkog unosa.



Slika 9. Izgled *Hash (CHF)* stranice

Stranica demonstrira rad različitih kriptografskih heš funkcija, gde korisnik može da vidi da je izlaz svake od *CHF* uvek iste veličine za bilo koji unos, i njihove osobine poput: kompjuterski nemogućeg pronalaska dve iste poruke koje daju istu *hash* vrednost, lavinskog efekta – mala promena u početnoj poruci dovodi do velike promene u izlazu funkcije, nemogućnost pronalaska početne poruke znajući izlaznu vrednost.

Prelaskom miša preko *label*-a sa leve strane iskače tooltip u kom je dat kratak opis za svaku od *CHF*. Ovo važi i za labele na drugim stranicama, međutim zbog dužine rada te slike neće biti priložene.



Slika 10. Primer *tooltip*-a na *label*-e SHA512

Primer rada *hash* funkcija će u nastavku biti priložen kroz vizuelne primere:

Data:	<input type="text" value="Hello world"/>
MD5:	<input type="text" value="3e25960a79dbc69b674cd4ec67a72c62"/>
SHA1:	<input type="text" value="7b502c3a1f48c8609ae212cdfb639dee39673f5e"/>
SHA256:	<input type="text" value="64ec88ca00b268e5ba1a35678a1b5316d212f4f366b2477232534a8aeca37f3c"/>
SHA512:	<input type="text" value="b7f783baed8297f0db917462184ff4f08e69c2d5e5f79a942600f9725f58ce1f29c18139bf80b06c0fff2bdd34738452ecf40c488c22a7e3d80cdf6f9c1c0c"/>

Slika 11. Primer izlaza *hash* funkcija za ulaz *Hello world*

Data:	<input type="text" value="Helo world"/>
MD5:	<input type="text" value="8576b8cd3ade0bed4379b6f22844eef4"/>
SHA1:	<input type="text" value="d9ac96d592fdc99a1d057738f365cdb628609c13"/>
SHA256:	<input type="text" value="f610cec9b877fc644d273ed4d8b0728ee49b193e538b194ba9281e07cc1366be"/>
SHA512:	<input type="text" value="c63086f8f1a0191a48b64295e4f4c43b406398f181e5639f6f95653fa255875874f14af7fe2f0357246d7eb380550438630852446651888fcb12ef04bfa5t"/>

Slika 12. Primer izlaza *hash* funkcija za ulaz *Helo world*

Iako postoji jednoslovna razlika u ulazima funkcija na slikama 11 i 12, izlazi su potpuno različiti, te je nemoguće uopšte i pretpostaviti da je razlika u ulazima toliko mala.

Ukoliko izlaz *MD5* funkcije sa slike 12 dovedemo sada kao novi ulaz (Slika 13) takođe se dobija novi neprepoznatljiv niz heksadecimalnih znakova.

Data:	<input type="text" value="8576b8cd3ade0bed4379b6f22844eef4"/>
MD5:	<input type="text" value="10d80b0ca4c0a80c5b7e27e2e1308694"/>
SHA1:	<input type="text" value="fa6b1969c368167e1cc6e27430bc1fd5f89f299e"/>
SHA256:	<input type="text" value="f096e48909a44633a1af091394dd7dcb398b176f5fb1fa520e06606532e39723"/>
SHA512:	<input type="text" value="345cb541b9124fcc548dd91213876e7907bb8bc88e7d45e9501f39eccbeee58684670c40263f3a3e7030f2c652873faade4aa35007779c78cd8b288d6"/>

Slika 13. Primer izlaza *hash* funkcija za ulaz *8576b8cd3ade0bed4379b6f22844eef4*

Treba imati u vidu da je *MD5* potpuno kriptografski razbijena, dok je *SHA1* podložna kolizionim napadima, tako da se ove dve funkcije ne smatraju bezbednim. U nastavku aplikacije i rada se koristi samo *SHA256*.

Za generisanje izlaza ovih kriptografskih funkcija je korišćena javno dostupna *JavaScript* biblioteka *CryptoJS*. Poziv ovih funkcija je dosta jednostavan i dat je u prilogu u bloku koda 6.

U ovom slučaju nemamo parametre `chain` i `block`, tako da je ponašanje funkcije `getText(chain, block)` uvek isto – vraća vrednost korisničkog unosa iz elementa sa *ID*-jem *data*, a to je gorepomenuta *TextArea*. *JavaScript* nam omogućava da pozovemo funkcije i bez parametara iz njihovih definicija, što nama u ovom slučaju odgovara, jer se oni svakako i ne koriste.

Svaka stranica *overload*-uje funkciju `getText()` novom definicijom, što je u suštini samo nadovezivanje (konkateniranje) svih mogućih vrednosti polja za korisnički unos iz jednog bloka u jedan string.

`sha256(chain, block)` vraća *hash* vrednost bloka prosleđenog kroz parametre pomoću poziva `CryptoJS.SHA256` funkcije čija malo komplikovanija implementacija je data u prilogu u bloku koda 7.

4.4. BLOCK.PUG

Ova stranica prikazuje osnovna polja svakog bloka. *Data* polje sa prethodne stranice je sada razbijeno na tri polja i zadržana je samo *SHA256 hash* funkcija. Ta polja su predstavljena promenljivim *TextBox*-ovima za magični broj - *Magic* i *ID* (ili vremenski žig) – *Block (number)*, a *TextArea*-jom za same podatke unutar bloka, koji se takođe zovu *Data*. *Hash* vrednost stringa koji se dobije kada se sva prethodna polja konkatiraju se nalazi u nepromenljivom *TextBox*-u.

The screenshot shows a web interface titled "BLOCK". At the top, there is a navigation bar with links: Hash, Block, Blockchain, Transactions, Coinbase, Keys, Signatures, Signed txs, Difficulty, Mining, Distributed, Network, References, and About. The main content area contains a form with four input fields: "Magic" (value: 51948), "Block" (value: 14), "Data" (a large text area), and "Hash" (value: bd99996c5fa809c0de5657b46c7979add7a6ed8da5d46ba721dba8ec65e15de). At the bottom of the page, it says "System for Visual Representation of Blockchain Technology © 2022 Dimitrije Knezevic".

Slika 14. Izgled *Block* stranice

Magic i *Block* polja su podešena lambda funkcijama vezanim za događaje (eng. *events*) tako da mogu da primaju samo numeričke karaktere i da ne mogu da ostanu prazna, dok *Data* može da primi sve *ASCII* karaktere i može sadržati prazan unos. Kao i na prethodnoj stranici, *Hash* polje se računa u stvarnom vremenu pri svakoj promeni nekog od prethodnih polja bez poziva parametara. Primer načina korišćenja lambda funkcija vezanih za polja koja se veoma često koriste

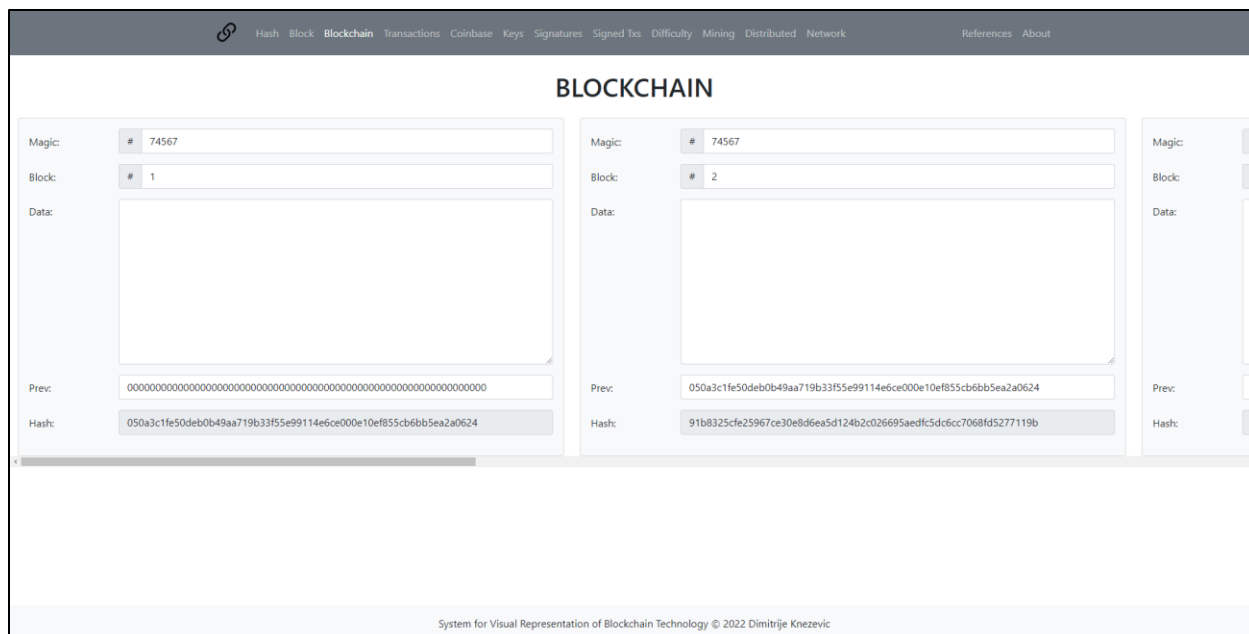
u autorovoj implementaciji moguće je pronaći u prilogu u bloku koda 8.

4.5. BLOCKCHAIN.PUG

Na blokčejn stranici uvodimo dosta novih stvari koje će se koristiti do kraja rada i aplikacije:

- Enumeraciju lanaca i blokova koji se koriste kao parametri funkcija
- Polje koje nam zapravo omogućava da ulančavamo blokove – *Prev*, skraćeno za *Previous Hash*
- *Pug* module koji u suštini predstavljaju stranice unutar stranica, time je realizovano ponavljanje blokova koji se referenciraju brojevima lanca (za sada imamo samo jedan) i bloka
- Provere validnosti blokova na osnovu različitih polja

Inicijalne vrednosti polja u blokovima su unapred izračunate i manuelno postavljene na određenu vrednost (eng. *hard coded*) prilikom inicijalizacije stranice. *Prev* vrednosti su podešene tako da su identične *Hash*-u prethodnog bloka, osim u prvom bloku (eng. *genesis block*) koji u suštini može da sadrži proizvoljnu *Prev* vrednost, jer nema prethodnika, tako da je radi jednostavnosti inicijalizovana svim nulama. Pošto postoji veći broj blokova, stavljen je *scroll bar* kojim korisnik može da se kreće kroz lanac.



Slika 15. Izgled *Blockchain* stranice

Svaki blok je generisan na osnovu *Pug* fajla *includes/simpleblock.pug* čiji je sadržaj naveden u prilogu u bloku koda 9.

Kao što vidimo, svi *ID*-jevi elemenata su parametrizovani (npr. `id='chain'+simpleblock.chain+'block'+simpleblock.block+'hash'`) tako da jednostavno možemo da referenciramo svaki od njih u fajlu koji sadrži više ovakvih blokova na identičan način kao što je dato u prilogu u bloku koda 10.

Kao što je već spomenuto, a i može se videti u gornjem bloku kodu, *Prev* polja na slici **Error! Reference source not found.** su izračunata unapred zbog bržeg učitavanja stranice i boljeg korisničkog iskustva. Sličan niz blokova će se koristiti na većini preostalih stranica.

Sada ćemo da prođemo kroz mehanizme provere validnosti blokova unutar ovog jednostavnog lanca:

1. Magični broj – Svi blokovi moraju da sadrže identičnu vrednost u *Magic* polju. Vrednost magičnog broja se proverava sa vrednošću broja iza i tako sve do *genesis* bloka. Ovo znači da svi blokovi moraju da sadrže magični broj koji se nalazi u *genesis* bloku, inače se smatraju nevalidnim. Implementaciju videti u prilogu u bloku koda 11.

Slika 16. Primer nevalidnog magičnog broja

2. Broj bloka – Svi blokovi osim prvog moraju da sadrže za jedan veću vrednost u *Block* polju od svog prethodnika. Ovo je pojednostavljena varijanta vremenskog žiga, gde blok koji je kasnije dodat ne može da se nađe pre ranije dodatog bloka u lancu. Implementaciju videti u prilogu u bloku koda 12.

Slika 17. Primer nevalidnog broja bloka

3. Prethodni *Hash* – Svi blokovi osim prvog moraju da imaju vrednost *Prev* polja jednaku vrednosti *Hash* polja svog prethodnika. Ukoliko se te vrednosti ne poklapaju, a heš trenutnog bloka je validan onda je došlo do promene podataka u prethodnom bloku, tako da je on označen kao nevalidan. Implementaciju videti u prilogu u bloku koda 13.

Block #	Magic	Block	Data	Prev	Hash
3	74567	3	New data	91b8325cfe25967ce30e8d6ea5d124b2c026695aedfc5dc6cc7068fd5277119b	a628c0b4c4f943db36abe56108494746b999fc30bf4ac0d46e181b52b23a9103
4	74567	4		3eae7011c3e4525f8d5f221f8c60e3facec81040ca774abd3b648ff835cf3a3	4be027c3c4df3d226a35d337ba400dc7097c085632ade089bc56ff9c1e47eb

Slika 18. Primer nevalidne prethodne heš vrednosti

4. *Hash* – Svi blokovi moraju da imaju *hash* vrednost koja se dobija heširanjem svih drugih svojih konkateniranih polja (u ovom slučaju *Magic+Block+Data+Prev*) jednaku vrednosti u svom *Hash* polju. Radi jednostavnosti i brzine korišćenja aplikacije, autor je stavio da se pri promeni neke od ulaznih vrednosti unutar bloka, nova vrednost *Hash* polja za sada automatski izračunava, tako da je ovaj uslov validnosti trenutno uvek ispunjen.

Ove provere važe i za sve sledeće lance sa kojima ćemo se susresti, te tada nećemo prolaziti kroz njih, već samo kroz novouvedene validacije.

Na ovoj stranici je takođe uvedena vizuelna reprezentacija nevalidnih podataka i blokova unutar lanca što je vidljivo na slikama iznad. Polje koje nije validno je označeno crvenom bojom, i svi blokovi do kraja lanca, uključujući i taj blok su takođe obojeni crvenom bojom. Ovo bi u stvarnom sistemu značilo da tačno znamo koji blok nije validan i zbog čega, tj. koji podaci su menjani unutar kog bloka.

Ova funkcionalnost je realizovana dodavanjem i oduzimanjem klasa određenim elementima na osnovu rezultata funkcija koje proveravaju validnosti polja. Implementacija je data u prilogu u bloku koda 14. Slične funkcije su korišćene i na ostalim stranicama, te kroz njih nećemo prolaziti.

Pravilo **!important** iz bloka koda u 15 prilogu se koristi da bi ovi stilovi nadjačali *Bootstrap* stilove koji se ne poklapaju sa njima. **bg-light** koga ovaj stil nadjačava je *Bootstrap*-ova klasa iz već viđenog bloka koda 14.

4.6. TRANSACTIONS.PUG

Ova stranica uvodi koncept transakcija koje zamenjuju do sada uopšteno i apstrahovano polje *Data* u prethodnim primerima blokova. Transakcije su u autorovoj implementaciji sistema vezane za imaginarsku kriptovalutu i za sada će sadržati numeričku svotu kriptovalute, pošiljaoca i primaoca (u *plain text*-u).

Slika 19. Izgled Transactions stranice

Svaki blok je generisan na osnovu *Pug* fajla *includes/transactionblock.pug* koji umesto *Data* dela iz *includes/simpleblock.pug* sadrži kod iz bloka koda 16 u prilogu.

Transakcije unutar bloka se prosleđuju iz roditeljske stranice detetu u vidu niza prilikom kreiranja bloka na način koji je prikazan u bloku koda 17 u prilogu.

Uvećan primer izgleda dela bloka posvećenog transakcijama, gde se jasno može videti sadržaj polja generisanih u kodu iznad dat je na slici:

Tx:	£	12.50	From:	Andrei	To:	Nichole
	£	3.44	From:	Yamato	To:	George
	£	9.03	From:	Yamato	To:	Mark
	£	25.00	From:	Luke	To:	Andrei
	£	25.00	From:	Luke	To:	Yamato
	£	18.67	From:	Andrei	To:	George

Slika 20. Primer transakcija u *plain text*-u

value polje koje predstavlja količinu kriptovalute je numeričnog tipa sa istim proveravama koje imaju i druga numerička polja (ne može biti prazno i ne može sadržati nenumeričke karaktere). **from** i **to** polja koja predstavljaju pošiljaoca i primaoca za sada nemaju dodatne provere.

4.7. COINBASE.PUG

Coinbase stranica u sistem uvodi mehanizam za praćenje kreiranja i kretanja novca preko polja *Coinbase* i *References*. Sada je omogućeno da kriptovalutu šalju samo korisnici koji već iz prethodnih transakcija imaju dovoljnu količinu novca na njihovom računu. Radi efikasnosti algoritma provere, prilikom svake transakcije, pošiljalac mora sav preostali novac da pošalje sam sebi, time eliminišući potrebu da se svaki put ide nazad do početka lanca radi provere stanja na računu. Veoma sličan mehanizam koristi i *Bitcoin*.

The screenshot displays the 'COINBASE AND TX TRACKING' interface. It features three panels, each representing a transaction. The top panel shows a transaction with Magic # 74567, Block # 1, and a Coinbase value of £ 20.00 sent to Mark. The middle panel shows a transaction with Magic # 74567, Block # 2, and a Coinbase value of £ 20.00 sent to Sophia, with references to previous transactions (4.55 to Mark, 15.45 to Mark). The bottom panel shows a transaction with Magic # 74567, Block # 3, and a Coinbase value of £ 20.00 sent to Sophia, with references to previous transactions (2.50 to Mark, 17.50 to Mark). Each panel includes a 'Prev' field with a long hash and a 'Hash' field with a shorter hash. The interface is titled 'COINBASE AND TX TRACKING' and includes a navigation bar at the top with links like Hash, Block, Blockchain, Transactions, Coinbase, Keys, Signatures, Signed txs, Difficulty, Mining, Distributed, Network, References, and About.

Slika 21. Izgled *Coinbase* stranice

Coinbase polje postoji u svakom bloku i sastoji se od količine kriptovalute koja se dodeljuje kao nagrada za kreiranje, tj. rudarenje bloka **coinvalue** (rudarenje ćemo videti u kasnijim poglavljima kako je implementirano) i imena nagrađenog korisnika **cointo**. Posle svakog petog bloka, nagradna svota kriptovalute se prepolovi što je vidljivo u prilogu u bloku koda 18. Za ovo polje važe iste provere unosa kao i za druga slična polja koja smo već videli.

Magic:	#	74567
Block:	#	5
Coinbase:	£	25.00 To: George
Tx:	References	3
	£	1.00 From: George To: Mark
	£	19.00 From: George To: George
	References	4
	£	4.00 From: Andrei To: Mark
	£	0.09 From: Andrei To: Andrei
Prev:	0cc4d3d39da93950f20e55502d2e9e973d957d8d6fb66c1f9a52b320fe892023	
Hash:	e35cc591bb586c401c94868934d7916557b2446f938d10b444f514e8bd4f23b8	

Slika 22. Primer nevalidnog Coinbase polja

References polje je vezano za svaku transakciju. Transakcije više ne postoje u *genesis* bloku, jer u trenutku kreiranja bloka niko nije posedovao ni jedan novčić kriptovalute čiji blokčejn posmatramo. *References* sadrži niz celobrojnih vrednosti odvojenih jednim blanko znakom koje predstavljaju ID prethodnih blokova koji se koriste kao dokaz o postojanju određene količine kriptovalute na računu korisnika koji on pokušava da iskoristi u transakcijama ispod. Provere sada postaju malo složenije, a većina stvarnih sistema ima dodatnu strukturu podataka pored blokčejna koja sadrži ove informacije radi ubrzanja rada algoritma, međutim to u ovom kratkom primeru nije potrebno. Implementacija ove provere je data u prilogu u bloku koda 19.

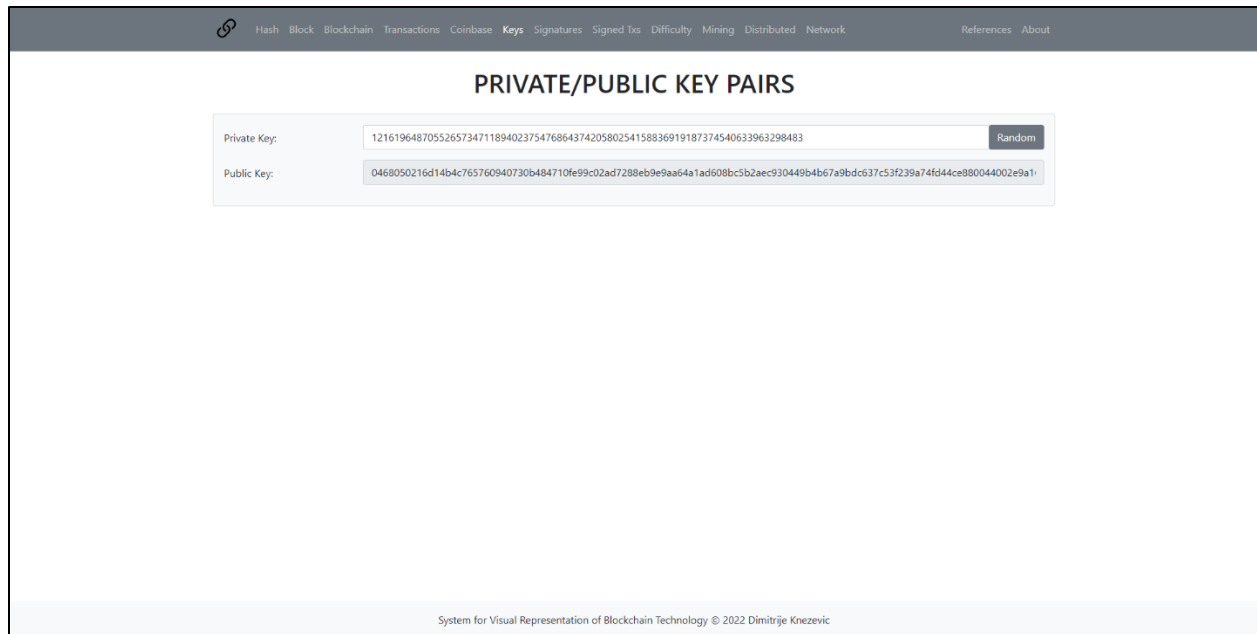
Polje je validno ukoliko se zbir dve transakcije ispod (od kojih je jedna povraćaj novčića samome sebi) gde je osoba pošiljalac slaže sa zbirom transakcija iz blokova sa *ID*-jem iz *References* niza gde je ta ista osoba bila primalac. U svim drugim slučajevima polje je nevalidno, kao i u slučajevima gde je referenca već iskorišćena u nekom od prethodnih blokova. Takođe nije validno ni referencirati trenutni ili budući blok, referencirati neki prethodni blok više od jedanput, ili ne referencirati ni jedan blok.

Magic:	#	74567
Block:	#	4
Coinbase:	£	20.00 To: Mark
Tx:	References	2 3
	£	4.90 From: Nichole To: Sophia
	£	5.15 From: Nichole To: Nichole
	References	3
	£	4.09 From: Sophia To: Andrei
	£	13.41 From: Sophia To: Sophia
Prev:	3b565fdb49a34b274389fe62a2cb4231493b1b4e3c81f6b2542cb1318e4640	
Hash:	0cc4d3d39da93950f20e55502d2e9e973d957d8d6fb66c1f9a52b320fe892023	

Slika 23. Primer nevalidne transakcione reference

4.8. KEYS.PUG

Stranica *Keys* uvodi veoma bitan pojam za kriptografiju, te stoga i blokčejn – privatne i javne ključeve.



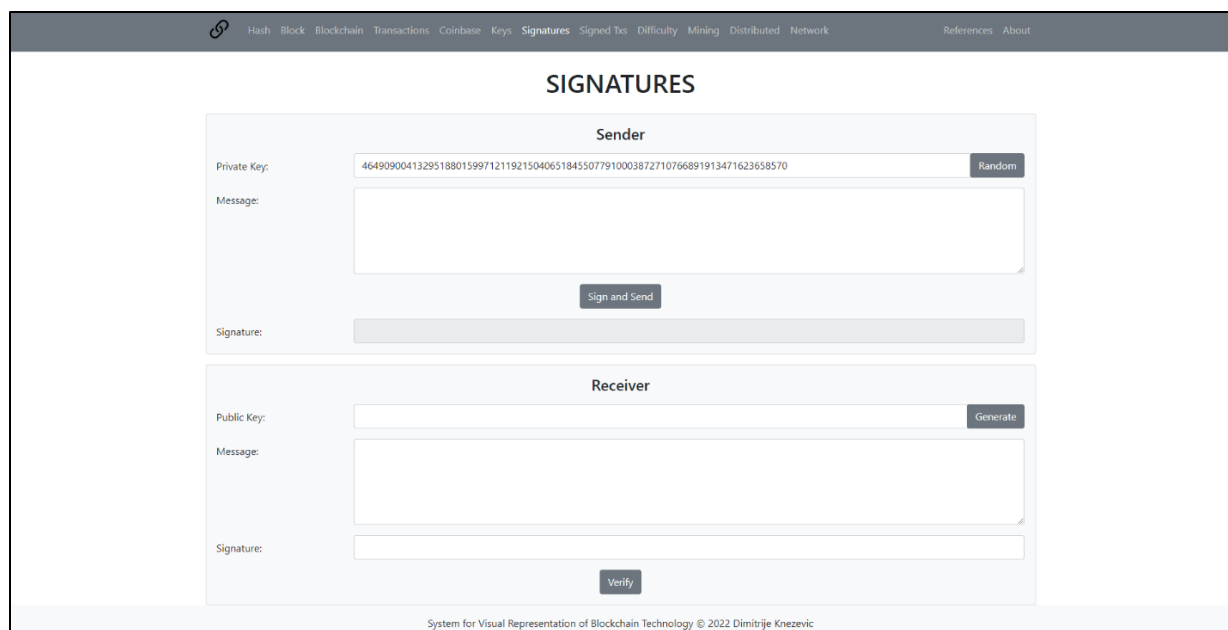
Slika 24. Izgled stranice *Keys*

Interfejs stranice je veoma jednostavan, sastoji se od dva polja od kojih jedno predstavlja proizvoljan celobrojni unos privatnog ključa, a drugo generisani heksadecimalni javni ključ na osnovu privatnog. Takođe postoji i dugme *Random* koje nam daje nasumičan celobrojni privatni ključ. Za implementaciju je korišćena javnodostupna *Javascript* biblioteka *elliptic.min.js* koja implementira datu funkcionalnost korišćenjem već pomenutog *ECDSA* algoritma nad eliptičnom krivom *secp256k1* ($y^2 = x^3 + 7$) koju takođe koristi i *Bitcoin*. Kod kojim se ovo postiže je dat u prilogu u bloku koda 20.

4.9. SIGNATURES.PUG

Na stranici *Signatures* uveden je koncept digitalnog potpisa koji u pozadini koristi privatne i javne ključeve.

U bloku *Sender* pošiljalac koristi svoj proizvoljan privatni ključ da potpiše i pošalje proizvoljnu poruku.



Slika 25. Izgled stranice *Signatures*

Sender

Private Key:
555985810992664534640781848179484572869207205549495801753706197774409695041704444
Random

Message:
Sender's customized message goes here.

Sign and Send

Signature:
3046022100c80f95df98c54ba1bc41d58fa9a4affb761f350f262f8eb405f536318873a121022100ff4c62238f697b74a4e4c4806906f06042a84f1e9391e

Receiver

Public Key:
046e231601b5f9bf5bfa0964bdd90822eb9a5061843ba1e6e7b679c348c7cea131a5d898656386c0757d2f3846ff4a0efa3439a7816a1e
Generate

Message:
Sender's customized message goes here.

Signature:
3046022100c80f95df98c54ba1bc41d58fa9a4affb761f350f262f8eb405f536318873a121022100ff4c62238f697b74a4e4c4806906f06042a84f1e9391e

Verify

Slika 26. Primer neizmenjene poruke sa validnim javnim ključem

Klikom na dugme *Sign and Send* se generiše potpis koji se zajedno sa porukom prekopira u *Receiver* blok, i generiše se javni ključ koji se koristi u *Receiver* bloku za validaciju poruke.

Primalac (*Receiver*) klikom na dugme *Verify* proverava validnost poruke i potpisa. Ukoliko

je sve regularno blok će se obojiti zelenom bojom, a u suprotnom crvenom.

Sender

Private Key:

555985810992664534640781848179484572869207205549495801753706197774409695041704444

Random

Message:

Sender's customized message goes here.

Sign and Send

Signature:

3046022100c80f95df98c54ba1bc41d58fa9a4affb761f350f262f8eb405f536318873a121022100ff4c62238f697b74a4e4c4806906f06042a84f1e9391e

Receiver

Public Key:

046e231601b5f9bf5bfa0964bdd90822eb9a5061843ba1e6e7b679c348c7cea131a5d898656386c0757d2f3846ff4a0efa3439a7816a1e

Generate

Message:

Sender's customized message goes here. You can trust me.

Signature:

3046022100c80f95df98c54ba1bc41d58fa9a4affb761f350f262f8eb405f536318873a121022100ff4c62238f697b74a4e4c4806906f06042a84f1e9391e

Verify

Slika 27. Primer izmenjene poruke sa validnim javnim ključem poruke

Sender

Private Key:

555985810992664534640781848179484572869207205549495801753706197774409695041704444

Random

Message:

Sender's customized message goes here.

Sign and Send

Signature:

3046022100c80f95df98c54ba1bc41d58fa9a4affb761f350f262f8eb405f536318873a121022100ff4c62238f697b74a4e4c4806906f06042a84f1e9391e

Receiver

Public Key:

046d231601b5f9bf5bfa0964bdd90822eb9a5061843ba1e6e7b679c348c7cea131a5d898656386c0757d2f3846ff4a0efa3439a7816a1e

Generate

Message:

Sender's customized message goes here.

Signature:

3046022100c80f95df98c54ba1bc41d58fa9a4affb761f350f262f8eb405f536318873a121022100ff4c62238f697b74a4e4c4806906f06042a84f1e9391e

Verify

Slika 28. Primer neizmenjene poruke sa nevalidnim javnim ključem

Ako je izmenjena ili poruka ili privatni ključ (a stoga i javni) kojim je potpisana poruka (ili oba) potpis neće biti validan. Ovime se garantuje da primalac može da veruje pošiljaocu poruke. Da bi se garantovalo i da pošiljaoc može da bude siguran da jedino primalac može da pročitati poruku, potrebno je na početku dodatno enkriptovati poruku primaočevim javnim ključem, ali to za potrebe našeg blokčejna nije ni potrebno, jer želimo da svi mogu da pročitaju i provere validnost naše poruke, tj. transakcije. Implementacija potpisivanja i verifikacije poruka je data u bloku koda 21 u prilogu.

4.10. SIGNED_TXS_AND_IDS.PUG

Sledeća stranica uvodi dva nova pojma zbog kojih *blockchain* može delovati komplikovano, ali oni su međutim od presudnog značaja za potvrdu transakcija i sada ćemo razotkriti šta ona zapravo predstavljaju.

The screenshot displays a web application titled "SIGNED TXS AND TX ID'S". It features three panels, each showing a transaction with the following fields:

- Magic:** # 74567
- Block:** # 3 (left), # 4 (middle), # (right)
- Coinbase:** £ 20.00 (left), £ 20.00 (middle), £ (right)
- To:** 04e493dbf1c10d80f3581e4904930b14 (left), 0479be667e9dcbbac55a06295ce870b (middle), (right)
- Tx:**
 - References:** 2 (left), 2 3 (middle), 3 (right)
 - Details:**

ID	£	From	To	Signature
1	2.50	04c604	04f930f	304602
2	17.50	04c604	04c604	304402
3	3.00	0479be	04f930f	304402
4	12.45	0479be	0479be	304502
- Prev:** e212904d6f652c48c8b9f6cf1549c87de156a129069fced896600b0d0e8365b6 (left), 690f19e1bd54828d3c05e8ce9d2bd8ffaec809c0c1ae4b1071069816274c12c5 (middle), 3f (right)
- Hash:** 690f19e1bd54828d3c05e8ce9d2bd8ffaec809c0c1ae4b1071069816274c12c5 (left), 30df378ec9f38659db333a3bc9cc1a49b452fb9c70fc9c40df5ebd7abc3b8 (middle), fa (right)

At the bottom, it states: "System for Visual Representation of Blockchain Technology © 2022 Dimitrije Knezevic".

Slika 29. Izgled stranice *Signed txs and tx ID's*

Prvi novi pojam je zamena ljudski čitljivih imena sa jedinstvenim identifikatorima koji predstavljaju javne ključeve svakog od korisnika (polja *From* i *To*). Javni ključevi su dobijeni na osnovu korisničkih privatnih ključeva koje samo njihovi vlasnici znaju i svaka transakcija je potpisana od strane pošiljaoca.

Drugi novouvedeni pojam je novo polje koje predstavlja identifikator transakcije za svakog korisnika, a ne za svaki blok (iako se na slici iznad ne vidi razlika). Njegova svrha je onemogućavanje napada na sistem dupliranjem transakcije tako što taj identifikator takođe ulazi u potpis na kraju transakcije i time nije moguće jednostavno kopiranje neke od prethodnih transakcija. Ovaj identifikator takođe može biti implementiran kao vremenski žig, ali je zbog

jednostavnosti ovde iskorišćen brojač koji se inkrementira.

The screenshot shows two transaction forms side-by-side. The left form is for a valid transaction (Block #5), and the right form is for an invalid one (Block #6). The right form's 'References' section shows 4 references instead of 5, and the 'From' field for the first reference is highlighted in red, indicating an invalid reference.

Slika 30. Primer nevalidnog potpisa transakcije

Zamislamo da je napadač, na primer, u bloku broj 6 promenio prvu transakciju tako da stoji da je poslato 15.50£ umesto 14.50£. On čak i da se vrati unazad kroz čitav *blockchain* i da izmeni prethodne transakcije kako bi čitav niz referenci unazad bio dobar, tj. matematički bio validan, i dalje nije u stanju da izmeni potpis bez poznavanja privatnog ključa pošiljaoca u ovoj transakciji, i privatnih ključeva svih prethodnih pošiljaoca u transakcijama u kojima je sadašnji pošiljalac bio primalac. Ovaj poslednji korak znači da ni sam pošiljalac ne može da izmeni već poslatu transakciju, iako već poseduje svoj privatni ključ.

The screenshot shows two transaction forms side-by-side. The left form is for a valid transaction (Block #4), and the right form is for an invalid one (Block #5). The right form's 'References' section shows 3 references instead of 4, and the 'ID' field for the first reference is highlighted in red, indicating an invalid ID.

Slika 31. Primer nevalidnog ID-ja transakcije

Kao što je već napomenuto, ID transakcije omogućava ovom lancu da se odbrani od kopiranja prethodnih transakcija, a pogotovo u istom bloku koji nije podložan proveravama validnosti referenci. Iako je potpis validan, znamo da sledeći ID mora biti 3 (ili u slučaju da je u pitanju vremenski žig, mora biti vreme veće nego u prethodnoj transakciji) te stoga ova transakcija nije validna. Funkcije koje proveravaju validnost potpisanih transakcija su date u prilogu u bloku koda 22.

Težina rudarenja se odnosi samo na sisteme koji funkcionišu na osnovu *Proof of Work* konsenzus algoritma. Autor je izabrao da predstavi ovaj algoritam zato što je najpoznatiji, pogotovo široj javnosti (skoro svi su čuli za rudarenje *Bitcoin*-a).

Slika 32. Izgled stranice *Mining Difficulty*

[illegible]

Funkcija preslikavanja *Difficulty*-ja u maksimalne heš vrednosti je: inicijalni broj nula u binarnom obliku maksimalne heš vrednosti mora biti jednak broju težine. U stvarnim sistemima se koriste komplikovanije funkcije zbog lakšeg i granularnijeg manipulisanja maksimalnom

dozvoljenom heš vrednosti. Funkcije koje su korišćene za implementaciju ovog dela Sistema su date u bloku koda 23 u prilogu.

4.12. MINING.PUG

Mining stranica uvodi poslednja bitno polja za realizaciju *PoW blockchain-a* – *Nonce* i *Difficulty*. *Nonce* polje predstavlja celobrojnu vrednost koja kad se konkatenira sa svim ostalim poljima u bloku daje validnu heš vrednost, tj. heš vrednost manju od najveće dozvoljene vrednosti na osnovu polja izračunatog na osnovu polja *Difficulty* (takode celobrojna vrednost u autorovoj implementaciji, mada u praksi ne mora da bude) na način prikazan u prethodnom poglavlju.

The screenshot shows a web application titled "BLOCK MINING" with a navigation bar at the top containing links: Hash, Block, Blockchain, Transactions, Coinbase, Keys, Signatures, Signed Tx, Difficulty, Mining, Distributed, Network, References, and About. The main content area displays three mining blocks, each with a green background and a "Mine" button at the bottom. The first block has the following values: Magic: 74567, Block: 1, Coinbase: £ 20.00, Tx: 0479be667ef9dcbbac55a06295ce870l, Diff: 8, Nonce: 54, Prev: 00, Hash: 0025b37e4f1a041a95d9892428979db83bcf2f287dba0fd922db2098c4ee815. The second block has: Magic: 74567, Block: 2, Coinbase: £ 20.00, Tx: 04c6047f9441ed7d6d3045406e95c07l, Diff: 8, Nonce: 18, Prev: 0025b37e4f1a041a95d9892428979db83bcf2f287dba0fd922db2098c4ee815, Hash: 00dfa3c9ee655975a5e35dfd4e396eb6326e2fcbce0308cfbac4821a9e3871ec. The third block is partially visible on the right side of the screen.

Slika 34. Izgled *Mining* stranice

Takođe je dodato dugme *Mine* na dnu svakog bloka koje predstavlja proces samog rudarenja tako što se menjanjem polja *Nonce* traži prva validna heš vrednost za postavljenu težinu. Težine susednih blokova mogu da se razlikuju, i u stvarnim sistemima se uglavnom postavljaju tako da je prosečno vreme rudarenja novog bloka uvek isto, tj. ukoliko se ukupna procesorska snaga rudara poveća, onda se povećava i *Difficulty* i obrnuto.

U autorovoj implementaciji nije moguće postaviti težinu *Difficulty*-ja na više od 16, jer je za potrebe demonstracije rada *blockchain-a* to sasvim dovoljno, u suprotnom u zavisnosti od procesorske snage mašine na kojoj se izvršava kalkulacija (izračunavanje se vrši u korisničkom *browser-u*, a ne na serveru) korisnik može čekati predugo na rezultat. Funkcije za obradu i validaciju korisničkog unosa su date u bloku koda 24, a funkcije za rudarenje u bloku koda 25 u

prilogu.

Sada napadač čak i da uspe da izmeni neko od polja (a videli smo da je to skoro pa nemoguće u prethodnim poglavljima), mora takođe da uloži mnogo dodatnog vremena i zaista ogromnu količinu procesorske snage da bi pronašao validan heš i *Nonce* nakon što je izmenio podatke, i to mora da uradi za svaki sledeći blok u lancu – što je zaista nemoguće uraditi, pogotovo ne sa malim brojem mašina na raspolaganju.

Magic:	#	74567								
Block:	#	6								
Coinbase:	£	10.00			To:	04c6047f9441ed7d6d3045406e95c07c				
Tx:	References	4 5								
	ID:	5	£	14.50	From:	0479be	To:	04c604	Signature:	304402
	ID:	6	£	10.50	From:	0479be	To:	0479be	Signature:	304602
Diff:	<div>8</div>									
Nonce:	<div>52</div>									
Prev:	00cf78200fcb3aa21cb2a8b966c0535c66bd967163de1b20b897e5be276d7ce7									
Hash:	<div>c723ef3659061279a5a2549e3facb472cc3abc2af1e0ea8189c24fe4ef9e7227</div>									
<div>Mine</div>										

Slika 35. Primer nevalidne težine i *Nonce*-a

Zamislamo da je napadač promenio *Coinbase* poslednjeg bloka (koji je zbog gorenavedenog razloga najlakše napasti) tako da je on sada primalac. On je sada u trci sa vremenom da opet izrudari blok (koji upravo nije uspeo da izrudari – inače bi bio nagrađen *Coinbase*-om) pre nego što bilo ko u mreži izrudari naredni blok, tj. on sam se takmiči protiv cele mreže. Matematički šanse da uspe da to uradi su skoro nikakve, a i da nekako to uspe, postoji drugi problem – a to je što svi drugi čvorovi u mreži već dobili validan blok (više o tome u narednom poglavlju).

Magic:	#	74567								
Block:	#	6								
Coinbase:	£	10.00			To:	04c6047f9441ed7d6d3045406e95c07c				
Tx:	References	4 5								
	ID:	5	£	14.50	From:	0479be	To:	04c604	Signature:	304402
	ID:	6	£	10.50	From:	0479be	To:	0479be	Signature:	304602
Diff:	8									
Nonce:	48									
Prev:	00cf78200fcb3aa21cb2a8b966c0535c66bd967163de1b20b897e5be276d7ce7									
Hash:	0082d69a13ace0b6c303afe725fb5dde3f42374d273a896b41a19e53612935ed									

Slika 36. Primer validne težine i *Nonce*-a, nakon ponovnog rudarenja prethodno nevalidnog bloka

4.13. DISTRIBUTED.PUG

Stranica *Distributed* prikazuje ponašanje *blockchain*-a kao strukture podataka u mreži, tj. kada je distribuiran između više čvorova. Na stranici su tri identična *blockchain*-a sa prethodne *Mining* stranice od kojih svaki pripada jedinstvenom čvoru.

Promene u jednom *blockchain*-u se ne propagiraju na druge čvorove, a za važeću verziju *blockchain*-a se uzima onaj koji je validan i koji se trenutno nalazi na većini čvorova koji sadrže validne lance. Ukoliko svaki čvor ima jedinstvenu validnu verziju onda se ni jedna od tih verzija ne smatra važećom.

U stvarnom sistemu sa velikim brojem korisnika, čvorovi sa istom verzijom *blockchain*-a bi verovali jedni drugima, jer se im se heševi u poslednjem bloku poklapaju. U autorovoj implementaciji zeleni čvorovi veruju drugim zelenim čvorovima, a crvenim čvorovima ne veruje niko.

Pogledajmo ovo ponašanje sistema na primeru. U inicijalnom slučaju (prilikom učitavanja stranice) sva tri čvora su saglasna oko tačne verzije trenutnog lanca, tj. svi čvorovi su zeleni (iako se na slici vidi samo čvor A).

Slika 37. Ponašanje distribuiranog *blockchain* sistema prilikom napada 1

Recimo da napadač ponovo pokušava da izmeni *coinbase* primaoca u poslednjem bloku.

Slika 38. Ponašanje distribuiranog *blockchain* sistema prilikom napada 2

Čak i da je uspešan na čvoru A (lanac na čvoru A je tehnički ispravan), što bi značilo da je izrudario blok koji je izmenio pre nego što je čitav ostatak mreže izrudario sledeći validan blok,

lanac na čvoru A ostatak mreže ne bi smatrao validnim, jer čvorovi B i C imaju drugačiju heš vrednost poslednjeg bloka.

Peer A

Magic: # 74567

Block: # 5

Coinbase: £ 20.00 To: 04e493dbf1c10d80f3581e4904930b14

Tx: References 3

ID	Amount	From	To	Signature
1	£ 1.00	04e493	0479be	304502
2	£ 19.00	04e493	04e493	304402

References 4

ID	Amount	From	To	Signature
1	£ 4.00	042f8b	0479be	304402
2	£ 0.09	042f8b	042f8b	304502

Diff: 8

Nonce: 783

Prev: 002ce423edc990135ead116394fb2dd64aa2ef41d7d98fd2bf60113fb2c0db4f

Hash: 00c78200fcb3aa21cb2a8b966c0535c66bd967163de1b20b897e5be276d7ce7

Mine

Peer B

Magic: # 74567

Block: # 6

Coinbase: £ 10.00 To: 04c6047f9441ed7d6d3045406e95c07f

Tx: References 4 5

ID	Amount	From	To	Signature
5	£ 14.50	0479be	04c604	304402
6	£ 10.50	0479be	0479be	304602

Diff: 8

Nonce: 48

Prev: 00c78200fcb3aa21cb2a8b966c0535c66bd967163de1b20b897e5be276d7ce7

Hash: 0082d69a13ace0b6c303afe725fb5dde3f42374d273a896b41a19e53612935ed

Mine

Slika 39. Ponašanje distribuiranog *blockchain* sistema prilikom napada 3

Međutim ako napadač stigne da napadne i čvor B, onda u tom trenutku ni jedna verzija *blockchain*-a ne predstavlja većinu u sistemu tako da ne postoji validna verzija.

Peer B

Magic: # 74567

Block: # 5

Coinbase: £ 20.00 To: 04e493dbf1c10d80f3581e4904930b14

Tx: References 3

ID	Amount	From	To	Signature
1	£ 1.00	04e493	0479be	304502
2	£ 19.00	04e493	04e493	304402

References 4

ID	Amount	From	To	Signature
1	£ 4.00	042f8b	0479be	304402
2	£ 0.09	042f8b	042f8b	304502

Diff: 8

Nonce: 783

Prev: 002ce423edc990135ead116394fb2dd64aa2ef41d7d98fd2bf60113fb2c0db4f

Hash: 00c78200fcb3aa21cb2a8b966c0535c66bd967163de1b20b897e5be276d7ce7

Mine

Peer C

Magic: # 74567

Block: # 6

Coinbase: £ 10.00 To: 04c6047f9441ed7d6d3045406e95c07f

Tx: References 4 5

ID	Amount	From	To	Signature
5	£ 14.50	0479be	04c604	304402
6	£ 10.50	0479be	0479be	304602

Diff: 8

Nonce: 52

Prev: 00c78200fcb3aa21cb2a8b966c0535c66bd967163de1b20b897e5be276d7ce7

Hash: c723ef3659061279a5a2549e3facb472cc3abc2af1e0ea8189c24fe4ef9e7227

Mine

Slika 40. Ponašanje distribuiranog *blockchain* sistema prilikom napada 4

Nakon što je napadač uspešno izmenio i čvor B (što traje dosta kratko jer je već na čvoru

A izračunao potreban *Nonce*), sada, iako originalno validan, čvoru C neće verovati ni čvor B ni čvor A.

The screenshot displays a blockchain simulation interface with two main panels, Peer B (top) and Peer C (bottom), both with a green background. Each panel contains fields for Magic, Block, Coinbase, Tx (with References, ID, Amount, From, To, and Signature), Diff, Nonce, Prev, and Hash. A 'Mine' button is located at the bottom of each panel.

Peer B Data:

- Magic: # 74567
- Block: # 5
- Coinbase: £ 20.00 To: 04e493dbf1c10d80f3581e4904930b14
- Tx:
 - References: 3
 - ID: 1 £ 1.00 From: 04e493 To: 0479be Signature: 304502
 - ID: 2 £ 19.00 From: 04e493 To: 04e493 Signature: 304402
 - References: 4
 - ID: 1 £ 4.00 From: 042f8b To: 0479be Signature: 304402
 - ID: 2 £ 0.09 From: 042f8b To: 042f8b Signature: 304502
- Diff: 8
- Nonce: 783
- Prev: 002ce423edc990135ead116394fb2dd64aa2ef41d7d98fd2bf60113fb2c0db4f
- Hash: 00cf78200fcb3aa21cb2a8b966c0535c66bd967163de1b20b897e5be276d7ce7

Peer C Data:

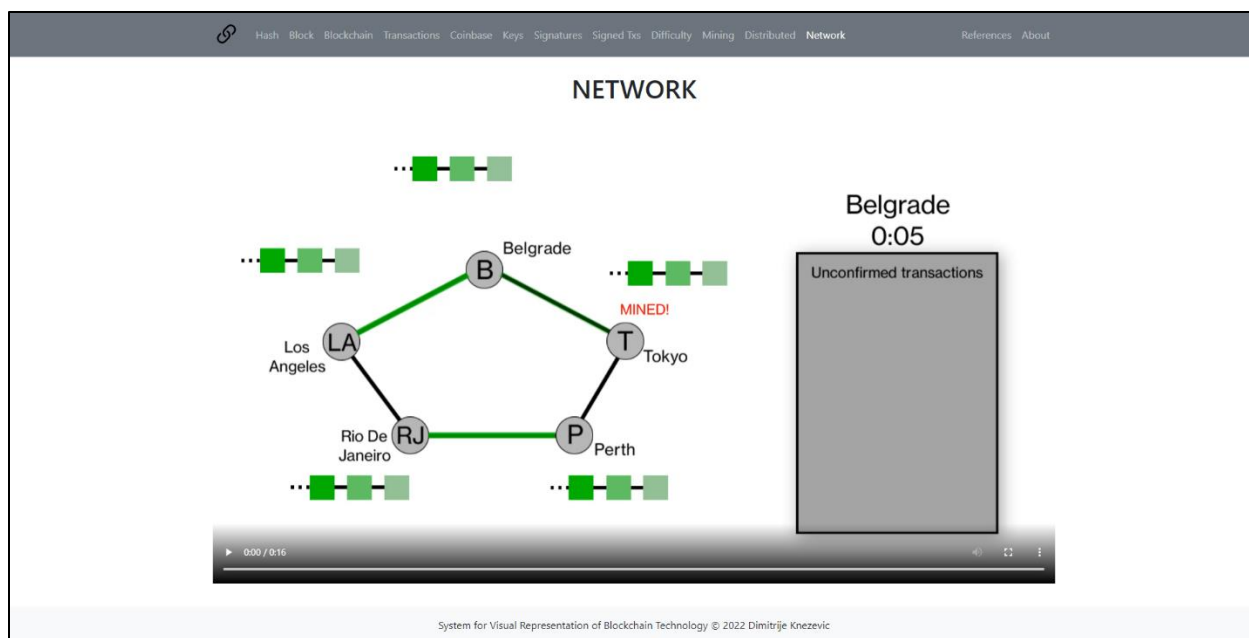
- Magic: # 74567
- Block: # 6
- Coinbase: £ 10.00 To: 04c6047f9441ed7d6d3045406e95c07
- Tx:
 - References: 4 5
 - ID: 5 £ 14.50 From: 0479be To: 04c604 Signature: 304402
 - ID: 6 £ 10.50 From: 0479be To: 0479be Signature: 304602
- Diff: 8
- Nonce: 48
- Prev: 00cf78200fcb3aa21cb2a8b966c0535c66bd967163de1b20b897e5be276d7ce7
- Hash: 0082d69a13ace0b6c303afe725fb5dde3f42374d273a896b41a19e53612935ed

Slika 41. Ponašanje distribuiranog blockchain sistema prilikom napada 5

Implementacije funkcionalnosti na ovoj stranici su date u prilogu u bloku koda 26.

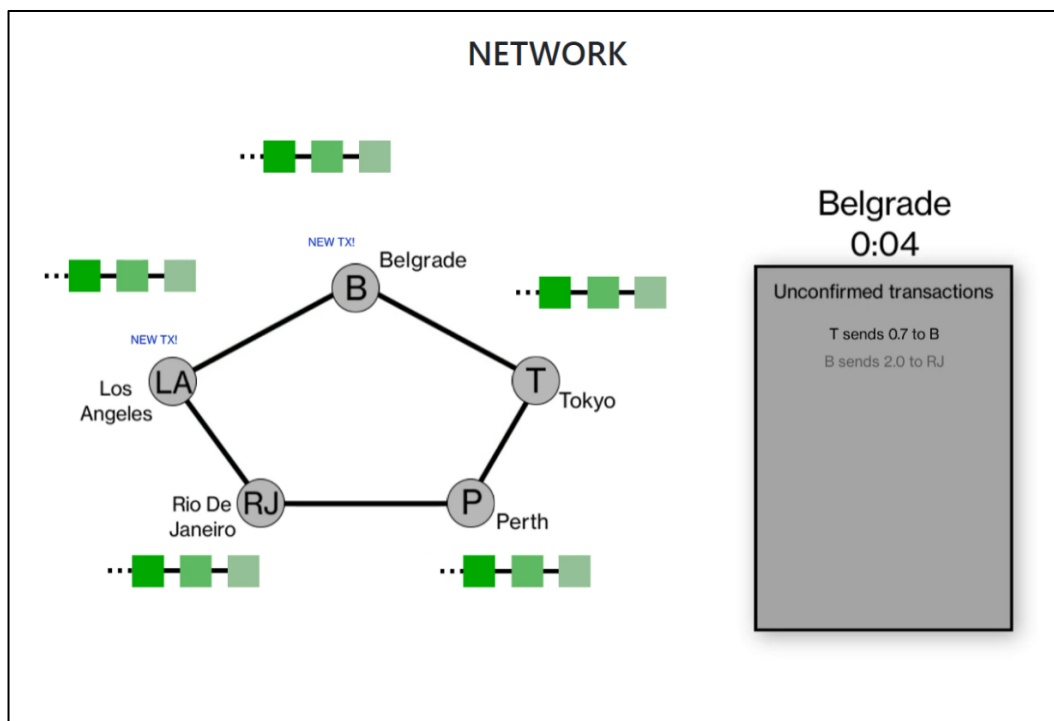
4.14. NETWORK.PUG

Poslednja stranica koja se bavi objašnjavanjem rada blokčejna jeste *Network*. Sastoji se od video zapisa koji je autor kreirao u programu *Camtasia* 9. Video zapis predstavlja pojednostavljenu animaciju kretanja paketa kroz mrežu i kreiranje novih blokova i transakcija na svakom od umreženih čvorova u trajanju od petnaestak sekundi.



Slika 42. Izgled *Network* stranice

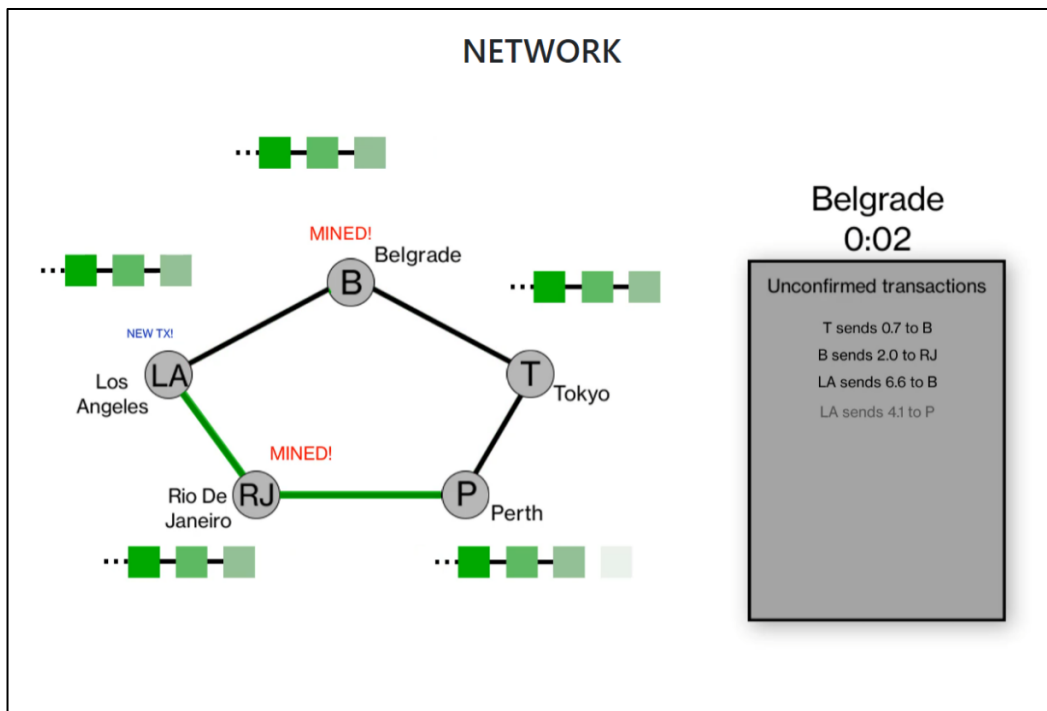
Cilj animacije je primetiti kako problem sinhronizovanja i validacije blokova zaista postaje inženjerski i algoritamski dosta zahtevniji kada su u pitanju stvarne globalne mreže koje kriptovalute koriste, tj. mreže dovoljno velike da “spora” brzina prenosa informacije kroz njiju značajno utiče na rad samog algoritma.



Slika 43. Animacija mreže 1

Na animaciji se vidi kako novi blokovi ne stižu svi u isto vreme do čvorova i to može izazvati sukob između dva ili više čvorova koji tvrde da su u isto vreme izrudarili najnoviji blok što dovodi do *fork*-ovanja lanca.

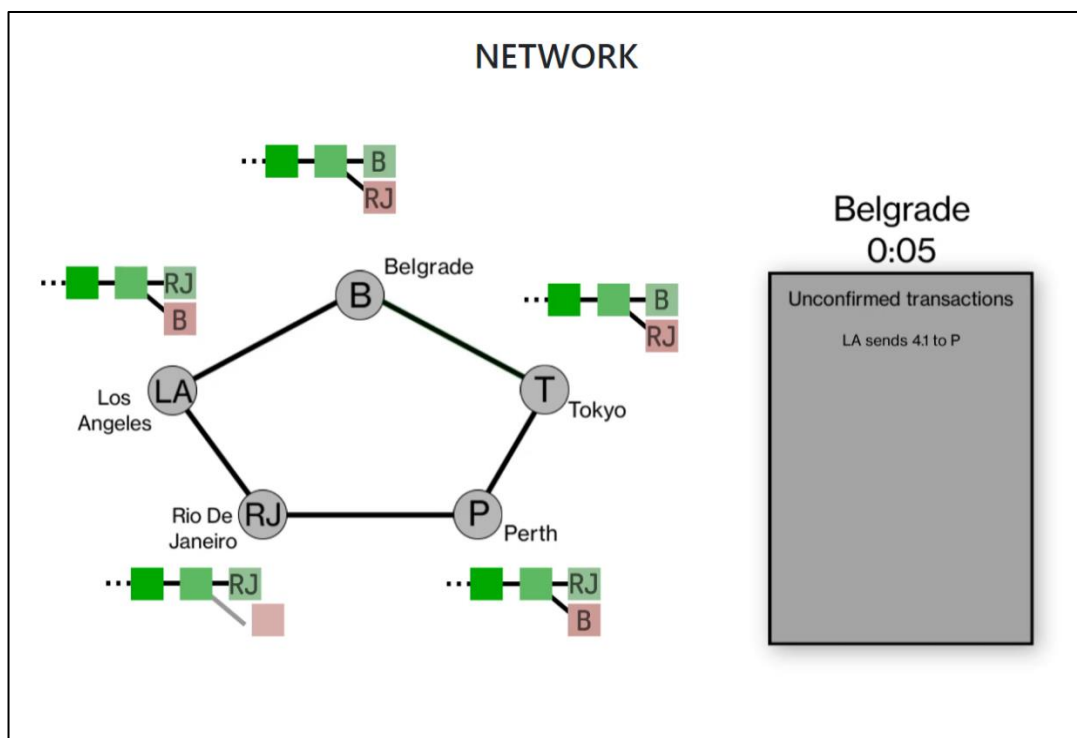
Takođe možda ne stignu sve transakcije na vreme (ili jednostavno ne budu izabrane, jer se u glavnom u blokčejn sistemima biraju na neki način transakcije koje će rudar pokušati da izrudari, tj. ubaci u najnoviji blok) do svih blokova, tako da onda problem implementacije postaje kompleksniji, jer ne raspolažu svi čvorovi istim informacijama u svakom trenutku.



Slika 44. Animacija mreže 2

Zbog toga je generalno praksa da se ne veruje u potpunosti najnovijim blokovima, zbog mrežnih problema sa konzistencijom i zbog toga što je njih najlakše napasti, već da se sačeka da se doda još nekoliko blokova u lanac pre nego što se koriste informacije iz datog bloka. Animacija prikazuje verodostojnost bloka nijansama zelene boje (što je blok svetliji, to je manje verodostojan).

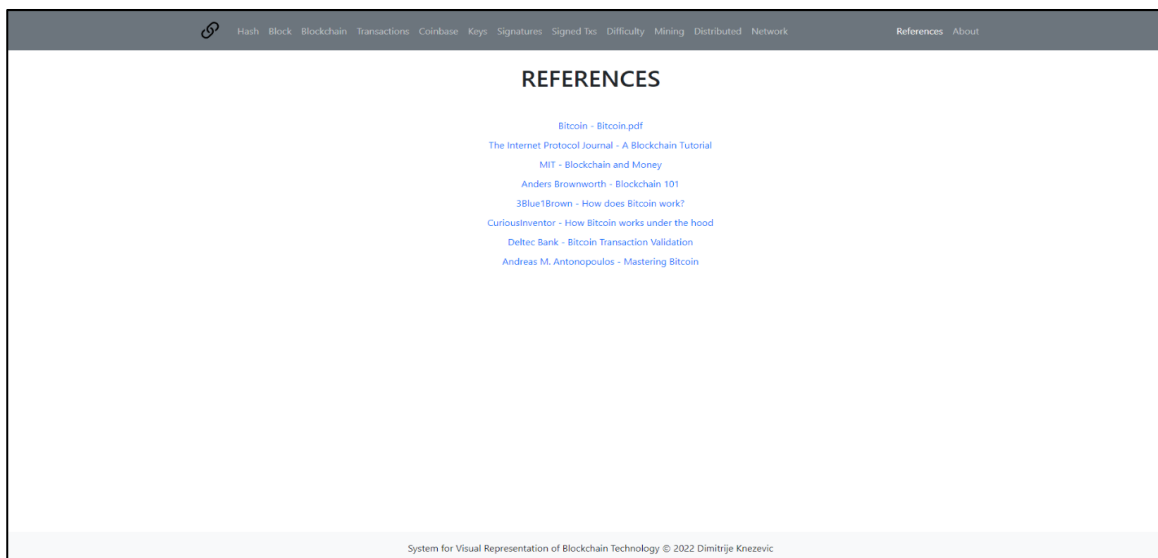
Prikaz *fork*-a u lancu:



Slika 45. Animacija mreže 3

4.15. REFERENCES.PUG

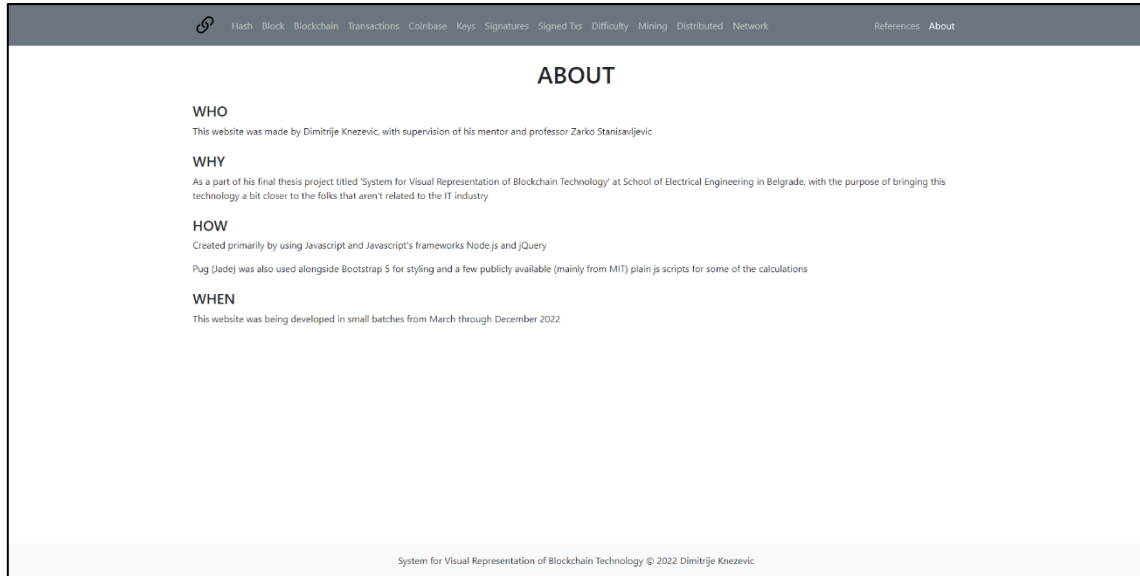
References stranica sadrži listu referenci korišćenih prilikom izrade aplikacije u vidu hiperlinkova.



Slika 46. Izgled *References* stranice

4.16. ABOUT.PUG

Poslednja stranica *About* sadrži kratke informacije o nastanku same web aplikacije – ko je kreator, kako, kad i zašto je kreirana.



Slika 47. Izgled *About* stranice

5. ZAKLJUČAK

U ovoj glavi ćemo se osvrnuti na to šta je urađeno u ovom radu i implementaciji aplikacije, i koji su dalji koraci za njeno usavršavanje.

Dat je teorijski uvod u *blockchain*, koji pokriva različite aspekte: *blockchain* kao struktura podataka i kao mrežni protokol, njegova istorija, njegove mane i mogući napadi na sistem koji je implementiran pomoću ove tehnologije. Zatim je data teorijska osnova korišćenih tehnologija za implementiranje web aplikacije kao i zašto su baš te tehnologije izabrane, i na kraju najduže poglavlje je posvećeno prolasku kroz implementiranu web aplikaciju za vizuelizaciju *blockchain*-a u kojoj su takođe dati implementacioni detalji za većinu funkcionalnosti.

Kao što je i prikazano, autor nije implementirao zasebnu strukturu podataka koja će eksplicitno predstavljati *blockchain*, već se odlučio za njegovu apstrakciju kroz vizuelni korisnički interfejs, jer je smatrao da će se ovakvom inkrementalnom implementacijom ova tehnologija najviše približiti publici koja nije bliže upoznata sa detaljima njenog rada. Rešenje je baš zbog toga namerno podeljeno u celine koje su labavo spregnute (eng. *loosely coupled*), da bi bila olakšana postepena implementacija pojedinačnih polja na svakoj sledećoj stranici (primer: transakcije počinju od toga što su samo jedno polje nazvano *Data*) i zbog toga što ta polja ne interaguju međusobno osim što daju zajedničku *hash* vrednost, dok su istovremeno moguća pojedinačna poređenja sa istim poljima iz prethodnih blokova u okviru lanca. Autor smatra da je ovakvom implementacijom prikazao veliki broj bezbednosnih mehanizama koje *blockchain* koristi, čineći ga zaista nepromenljivom strukturom podataka – a to je njegova glavna karakteristika.

Mane trenutnog sistema jesu da, zbog kodne implementacije koja se prvenstveno fokusirala na korisnički doživljaj sa veoma jasnim i unapred isplaniranim obimom funkcionalnosti, dalja (neplanirana) unapređenja sistema postaju sve teža i komplikovanija zbog nepoštovanja objektno orijentisanih principa programiranja.

Moguća poboljšanja sistema jesu prvenstveno u rešavanju gorepomenutih mana, tj. autor predlaže da, ako se u budućnosti planira dodavanje većeg broja drugih funkcionalnosti, prvo krene sa izmenom postojeće arhitekture, i jasno podeli *back-end* OOP implementacija *blockchain* strukture i njena validacija od *front-end* vizuelizacije. Moguće funkcionalnosti koje nisu pokrivene ovim radom, a predstavljaju sledeće korake u njegovom proširenju, jesu interaktivno korisničko dodavanje transakcija u blokove kao i novih blokova u lanac, a kasnije i interaktivne mrežne simulacije funkcionisanja sistema sa promenljivim parametrima (broj čvorova, broj transakcija u sekundi, prosečno vreme za rudarenje bloka, itd.) i/ili simulacija napada pokrivenih u teorijskom delu rada.

Krajnja ocena autora je da je *blockchain* nedovoljno ili pogrešno shvaćena tehnologija u široj javnosti, da joj se pridaje veći nego potreban značaj zbog nedostatka znanja o radu pozadinskih mehanizama i tehnologije koje sve zajedno čine *blockchain*, ali isto tako i da nisu dovoljno svesni stvarnih prednosti koje on pruža, novih mogućnosti koje otvara i svih njegovih mogućih primena (ne samo kod kriptovaluta već i šire). Bitno je razumeti da je *blockchain* dosta širok pojam, i da se različite *blockchain* implementacije kreiraju svakodnevno i postaju sve bolje i bolje, neretko i rešavajući prethodno nerešive probleme. Autor se nada da sistem za vizuelnu reprezentaciju *blockchain* tehnologije, čija je implementacija bio cilj ovog rada, može poslužiti kao prvi korak ka demistifikaciji ove relativno nove, ali komplikovane tehnologije.

6. LITERATURA

1. W. Stallings, "A Blockchain Tutorial", The Internet Protocol Journal, Dostupno na: <http://ipj.dreamhosters.com/wp-content/uploads/2017/12/ipj20-3.pdf> (Poslednji put pristupljeno decembra 2022.)
2. S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", Bitcoin.org, Dostupno na: <https://bitcoin.org/bitcoin.pdf> (Poslednji put pristupljeno decembra 2022.)
3. A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone, "Hash Functions" str. 33, Handbook of Applied Cryptography
4. E. Paul, "What is a Digital Signature – How it works, Benefits, Objectives, Concept", EMP Trust HR, Dostupno na <https://www.emptrust.com/blog/benefits-of-using-digital-signatures/> (Poslednji put pristupljeno marta 2023.)
5. Član 6 Pravilnika o uslovima koje mora da ispunjava kvalifikovano sredstvo za kreiranje elektronskog potpisa, odnosno pečata, i uslovima koje mora da ispunjava imenovano telo ("Sl. glasnik RS", br. 34/2018, 3/2020, 87/2020 i 64/2021), Dostupno na: <https://www.paragraf.rs/propisi/pravilnik-uslovi-kvalifikovano-sredstvo-za-kreiranje-elektronskog-potpisa.html> (Poslednji put pristupljeno aprila 2023.)
6. S. Driscoll, "How Bitcoin Works Under the Hood", ImponderableThings, Dostupno na: <http://www.imponderablethings.com/2013/07/how-bitcoin-works-under-hood.html> (Poslednji put pristupljeno marta 2023.)
7. "Elliptic Curve Digital Signature Algorithm", Wikipedia, Dostupno na: https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm (Poslednji put pristupljeno marta 2023.)
8. "How Many Bitcoins Are There?", Buy Bitcoin Worldwide, Dostupno na: <https://buybitcoinworldwide.com/how-many-bitcoins-are-there/> (Poslednji put pristupljeno januara 2023.)
9. "Here is Why Bitcoin Transactions Take So Long", OriginStamp, Dostupno na: <https://originstamp.com/blog/here-is-why-bitcoin-transactions-take-so-long> (Poslednji put pristupljeno marta 2023.)
10. "Blockchain forks", GeeksForGeeks, Dostupno na: <https://www.geeksforgeeks.org/blockchain-forks/> (poslednji put pristupljeno januara 2023.)
11. "Controlled supply", Bitcoin Wiki, Dostupno na: https://en.bitcoin.it/wiki/Controlled_supply (Poslednji put pristupljeno aprila 2023.)

12. "Consensus Algorithms in Blockchain", GeeksForGeeks, Dostupno na: <https://www.geeksforgeeks.org/consensus-algorithms-in-blockchain/> (Poslednji put pristupljeno januara 2023.)
13. A. T. Sherman, F. Javani, H. Zhang, E. Golaszewski, "On the Origins and Variations of Blockchain Technologies", IEEE, Dostupno na: <https://ieeexplore.ieee.org/document/8674176> (Poslednji put pristupljeno decembra 2022.)
14. D. Oberhaus, "The World's Oldest Blockchain Has Been Hiding in the New York Times Since 1995", Vice, Dostupno na: <https://www.vice.com/en/article/j5nzx4/what-was-the-first-blockchain> (Poslednji put pristupljeno decembra 2022.)
15. "History of Blockchain", The Institute of Chartered Accountants in England and Wales, Dostupno na: <https://www.icaew.com/technical/technology/blockchain-and-cryptoassets/blockchain-articles/what-is-blockchain/history> (Poslednji put pristupljeno decembra 2022.)
16. G. Gensler, "Blockchain and Money", Massachusetts Institute of Technology, Dostupno na: <https://ocw.mit.edu/courses/15-s12-blockchain-and-money-fall-2018/pages/lecture-slides/> (Poslednji put pristupljeno marta 2023.)
17. "Blockchain 1.0 vs. 2.0 vs. 3.0 - What's the Difference?", OriginStamp, Dostupno na: <https://originstamp.com/blog/blockchain-1-vs-2-vs-3-whats-the-difference/> (Poslednji put pristupljeno marta 2023.)
18. S. Outten, "Bitcoin Transaction Validation, What Exactly Goes on Under the Hood?", Deltec Bank, Dostupno na: <https://www.deltecbank.com/2021/10/05/bitcoin-transaction-validation-what-exactly-goes-on-under-the-hood> (Poslednji put pristupljeno marta 2023.)
19. A. M. Antonopoulos, "Mastering Bitcoin", O'Reilly, Dostupno na: <https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch01.html> (Poslednji put pristupljeno marta 2023.)
20. Repozitorijum "Bitcoin", Github, Dostupno na: <https://github.com/bitcoin/bitcoin/tree/9546a977d354b2ec6cd8455538e68fe4ba343a44> (Poslednji put pristupljeno marta 2023.)
21. "Genesis Block", Wikipedia, Dostupno na: https://en.bitcoin.it/wiki/Genesis_block (Poslednji put pristupljeno marta 2023.)
22. M. von Haller Grønbæk, "Blockchain 2.0, smart contracts and challenges", Bird & Bird Copenhagen, Dostupno na: https://www.twobirds.com/-/media/pdfs/in-focus/fintech/blockchain2_0_martinvonhallergroenbaek_08_06_16.pdf (Poslednji put pristupljeno marta 2023.)
23. A. Bhati, "Attacks on Blockchain", WeSecureApp, Dostupno na: <https://wesecureapp.com/>

[blog/attacks-on-blockchain/](#) (Poslednji put pristupljeno marta 2023.)

24. “10 Blockchain and New Age Security Attacks You Should Know”, Aruba, Dostupno na: <https://blogs.arubanetworks.com/solutions/10-blockchain-and-new-age-security-attacks-you-should-know/> (Poslednji put pristupljeno marta 2023.)
25. “Ethereum Whitepaper”, Ethereum, Dostupno na: <https://ethereum.org/en/whitepaper/> (Poslednji put pristupljeno aprila 2023.)

7. TABELA SLIKA

Slika 1. Jedan od mogućih načina korišćenja <i>CHF</i> -a za povezivanje blokova unutar <i>blockchain</i> -a	7
Slika 2. Primer načina funkcionisanja Merkleovog stabla.....	8
Slika 3. Dva primera izgleda eliptičke krive.....	11
Slika 4. Dijagram sekvence sistema.....	22
Slika 5. Struktura i povezanost susednih blokova u autorovoj implementaciji	24
Slika 6. Pristupanje aplikaciji sa mobilnog uređaja	25
Slika 7. Početna (Index) stranica	26
Slika 8. Navigacioni meni.....	26
Slika 9. Izgled <i>Hash (CHF)</i> stranice.....	27
Slika 10. Primer <i>tooltip</i> -a na <i>label</i> -e SHA512	27
Slika 11. Primer izlaza <i>hash</i> funkcija za ulaz <i>Hello world</i>	28
Slika 12. Primer izlaza <i>hash</i> funkcija za ulaz <i>Helo world</i>	28
Slika 13. Primer izlaza <i>hash</i> funkcija za ulaz <i>8576b8cd3ade0bed4379b6f22844eef4</i>	28
Slika 14. Izgled <i>Block</i> stranice.....	29
Slika 15. Izgled <i>Blockchain</i> stranice.....	30
Slika 16. Primer nevalidnog magičnog broja.....	31
Slika 17. Primer nevalidnog broja bloka.....	31
Slika 18. Primer nevalidne prethodne heš vrednosti.....	32
Slika 19. Izgled <i>Transactions</i> stranice	33
Slika 20. Primer transakcija u <i>plain text</i> -u.....	33
Slika 21. Izgled <i>Coinbase</i> stranice.....	34
Slika 22. Primer nevalidnog <i>Coinbase</i> polja.....	35
Slika 23. Primer nevalidne transakcione reference.....	35
Slika 24. Izgled stranice <i>Keys</i>	36
Slika 25. Izgled stranice <i>Signatures</i>	37
Slika 26. Primer neizmenjene poruke sa validnim javnim ključem.....	37
Slika 27. Primer izmenjene poruke sa validnim javnim ključem poruke	38
Slika 28. Primer neizmenjene poruke sa nevalidnim javnim ključem.....	38
Slika 29. Izgled stranice <i>Signed txs and tx ID's</i>	39
Slika 30. Primer nevalidnog potpisa transakcije.....	40
Slika 31. Primer nevalidnog <i>ID</i> -ja transakcije.....	40
Slika 32. Izgled stranice <i>Mining Difficulty</i>	41

Slika 33. Primer maksimalne moguće heš vrednosti za težinu 12	41
Slika 34. Izgled <i>Mining</i> stranice	42
Slika 35. Primer nevalidne težine i <i>Nonce</i> -a	43
Slika 36. Primer validne težine i <i>Nonce</i> -a, nakon ponovnog rudarenja prethodno nevalidnog bloka	44
Slika 37. Ponašanje distribuiranog <i>blockchain</i> sistema prilikom napada 1	45
Slika 38. Ponašanje distribuiranog <i>blockchain</i> sistema prilikom napada 2	45
Slika 39. Ponašanje distribuiranog <i>blockchain</i> sistema prilikom napada 3	46
Slika 40. Ponašanje distribuiranog <i>blockchain</i> sistema prilikom napada 4	46
Slika 41. Ponašanje distribuiranog <i>blockchain</i> sistema prilikom napada 5	47
Slika 42. Izgled <i>Network</i> stranice	48
Slika 43. Animacija mreže 1	48
Slika 44. Animacija mreže 2	49
Slika 45. Animacija mreže 3	50
Slika 46. Izgled <i>References</i> stranice	50
Slika 47. Izgled <i>About</i> stranice	51
Slika 48. New York Times koristi Merkleovo stablo heširanih digitalnih dokumenata kao dokaz njihove autentičnosti	66
Slika 49. Sadržaj genesis block-a Bitcoin-a sa obeleženom skrivenom porukom.....	68

8. TABELA BLOKOVA KODA

Blok koda 1. Organizaciona struktura fajlova	70
Blok koda 2. Izgled fajla <i>package.json</i>	71
Blok koda 3. Sadržaj fajla <i>index.js</i>	72
Blok koda 4. Sadržaj fajla <i>layout.pug</i>	73
Blok koda 5. Sadržaj <i>bs-config.js</i> fajla	74
Blok koda 6. Primer poziva i korišćenja <i>sha256</i> funkcije	74
Blok koda 7. Implementacija <i>CryptoJS.SHA256</i> funkcije.....	78
Blok koda 8. Primer korišćenja <i>lambda</i> funkcija vezanih za određene akcije.....	78
Blok koda 9. Sadržaj <i>simpleblock.pug</i> fajla.....	79
Blok koda 10. Korišćenje <i>pug</i> fajla unutar drugog <i>pug</i> fajla.....	80
Blok koda 11. Funkcija koja proverava validnost magičnog broja	80
Blok koda 12. Funkcija koja proverava validnost broja bloka	80
Blok koda 13. Funkcija koja proverava validnost heševa uzastopnih blokova	81
Blok koda 14. Funkcija koja ažurira izgled i heš vrednosti blokova u lancu	82
Blok koda 15. CSS stilovi koji se primenjuju na nevalidne elemente u lancu.....	82
Blok koda 16. Uvedena izmena u <i>transactionblock.pug</i> u odnosu na <i>simpleblock.pug</i>	83
Blok koda 17. Primer prenosa transakcija između roditeljske stranice i stranice deteta	83
Blok koda 18. Funkcija koja proverava validnost <i>Coinbase</i> polja	83
Blok koda 19. Funkcije koja proveravaju validnost jednostavnih transakcija i <i>References</i> polja	85
Blok koda 20. Kod za generisanje javnog ključa na osnovu privatnog pozivom funkcija iz <i>elliptic.min.js</i>	86
Blok koda 21. Funkcije koje implementiraju najbitnije funkcionalnosti na <i>Signatures</i> stranici ..	87
Blok koda 22. Funkcije koje proveravaju validnost potpisa i ID-ja transakcije	90
Blok koda 23. Funkcije pomoću kojih su realizovana računanja maksimalnih heš vrednosti na osnovu težine, i promene labela.....	91
Blok koda 24. Funkcije odgovorne za obradu korisničkog unosa za polja <i>Difficulty</i> i <i>Nonce</i>	91
Blok koda 25. Funkcije za rudarenje	92
Blok koda 26. Funkcije koje određuju validnosti <i>blockchain</i> -ova i čvorova u distribuiranom sistemu	94

9. PRILOG

9.1. OSTALI KONSENZUS ALGORITMI

9.1.1. PROOF OF STAKE

Drugi po redu najpoznatiji i najkorišćeniji konsenzus algoritam jeste dokaz ulogom (eng. *Proof of Stake – PoS*). Na primer kriptovaluta *Ethereum* je prešao sa korišćenja *PoW*-a na *PoS*, jer je smatrao da je ovo bolji način dokazivanja validnosti transakcija.

Čvorovi koji validiraju blokove se zovu validatori i oni stavljaju neku svotu svog novca kao ulog kladeći se na blokove za koje smatraju da su validni i da se mogu dodati u lanac. Nakon toga na osnovu blokova koji su zaista dodati, validatori dobijaju nagradu srazmernu njihovim ulozima.

Mreža onda dodeljuje nekom od validatora zadatak da dodaju novi blok u lanac nakon čega ukoliko se ostali validatori slože da je taj blok validan, svi koji su se kladili na taj blok dobijaju nagradu srazmernu njihovom ulogu.

Ukoliko se ostatak validatora ne slaže da je novododati blok zapravo validan (npr. može biti potrebno da se više od trećine validatora ne slaže), proces se ponavlja, a validator koji je bio prethodno izabran da doda taj blok gubi ceo ili deo svog uloga. Na ovaj način postoji finansijska inicijativa da se validatori slože i takođe postoje kazne za čvorove koji su maliciozni.

PoS je kao sistem ekonomski mnogo isplativiji od *PoW*-a, jer ne postoji takmičenje u ulozima i potrošenim resursima (hardverskim, mrežnim, električnim, itd.) među čvorovima, i potrebno je manje vremena da se transakcije potvrde. Međutim i dalje postoji 51% napad, samo što u ovom slučaju napadač mora da kontroliše više od 50% novčanog uloga u validaciji što je teže nego kod *PoW*-a da kontroliše više od 50% resursa, jer je broj validatora mnogo veći (lakše dostupno validiranje znači da veliki broj čvorova može da učestvuje u validaciji).

Mana ovog algoritma jeste da se favorizuju čvorovi sa velikim ulozima, jer je veća šansa da dobiju zadatak da validiraju blok i ukoliko to uspešno urade još više povećavaju svoj ulog u narednim rundama.

9.1.2. DELEGATED PROOF OF STAKE

Delegirani dokaz ulogom (eng. *Delegated Proof of Stake – DPoS*) je sličan običnom *PoS*-u osim što validatori glasaju za entitete za koje smatraju da treba da zapravo validiraju blokove (jačina glasa je srazmerna ulogu), umesto da ga algoritam sam izabere. U većini sistema broj validatora koji mogu da budu izabrani je između 21 i 101.

Izabrani validatori se zovu svedoci (eng. *witnesses*). Kada svedok uspešno validira blok, on i njegovi glasači dobijaju novčanu nagradu. Svedoci imaju ograničeno vreme da validiraju novi blok i ukoliko ne uspeju blok se smatra promašenim, transakcije ostaju i dalje nevalidirane, a nagrada se dodaje na sledeću nagradu koju dobija sledeći svedok koji uspešno validira taj blok.

Ukoliko svedok pokuša da uradi nešto maliciozno, on gubi svoj ulog isto kao kod *PoS*-a, tako da čvorovi nemaju nikakvu inicijativu da napadaju sistem. Određivanje da li je blok koji je svedok validirao mogu da rade svi čvorovi u mreži, mada za ovo ne postoje nagrade, već je inicijativa održavanje tačnosti i stabilnosti mreže.

U mreži postoje i posebni čvorovi zvani delegati koji nadgledaju celu mrežu i predlažu promene, npr. broj svedoka, veličine blokova, nagradne svote, itd.. Svi čvorovi mogu da učestvuju u biranju delegata i u glasanju za predložene promene.

Prednosti u odnosu na *PoS* algoritam jeste što je mreža više demokratizovana i finansijski inkluzivnija, jer je generalno potreban mali ulog. Manji ulog znači da više čvorova učestvuje u validaciji transakcija i napad na mrežu je teži.

9.1.3. PROOF OF BURN

Dokaz spaljivanjem (eng. *Proof of Burn – PoB*) je sličan *PoS*-u međutim umesto da validatori stavljaju novac kao ulog, koji kasnije može da se povuče ukoliko čvor ne želi više da bude validator, validatori spaljuju novac tako što ga šalju na nedostupnu adresu sa koje ne mogu nikako da ga vrate.

Mreža isto kao u *PoS*-u određuje koji validator će dobiti pravo da validira sledeći blok, a veću prednost imaju čvorovi koji su više novca spalili. Ukoliko validator uspešno validira blok dobija nagradu, a ukoliko ne uspe dobija zabranu od dalje validacije. Ovime se validatori dugoročno obavezuju da će ispravno validirati blokove, jer im je u suprotnom finansijski neisplativo.

Problem *PoB*-a je što se kao i kod *PoW*-a nepotrebno troše resursi (novac u ovom slučaju), i isto kao kod *PoS*-a čvorovi sa većim novčanim sredstvima su u prednosti da više zarađuju.

Primer kriptovalute koja koristi *PoB* je *BitShares*.

9.1.4. PROOF OF CAPACITY

Dokaz kapacitetom (eng. *Proof of Capacity – PoC*) je jedan od najnovijih algoritama koji je sličan *PoW*-u osim što rudari ne ulažu u procesorsku snagu već kapacitet hard diskova, što ga čini dosta ekonomski isplativijim i boljim ekološkim rešenjem, jer troši mnogo manje električne energije.

Rudari pre procesa samog rudarenja moraju da na svojim hard diskovima sačuvaju velike

fajlove pre-izračunatih heš funkcija koje se inače veoma teško računaju. Čuvanje ovog fajla se naziva *plotting* i može da potraje i nedeljama. U ove heš funkcije ulaze i podaci o rudaru tako da su unikatne za svakog rudara. Svaki *nonce* se sastoji od nekog broja *scoop*-ova (npr. 4096 kod *Burstcoin*-a), gde *scoop* predstavlja dve uzastopne heš vrednosti.

Rudar zatim izrudari (na sličan način kao kod *PoW*-a, samo što je problem dosta lakši) broj *scoop*-a koji ubacuje u svaki *nonce* (npr. rudarenjem dobije broj 12, i onda redom posećuje svaki 12-ti *scoop* svakog *nonce*-a koji ima sačuvan na hard disku) ne bi li dobio što manju *deadline* vrednost. Očigledno što više *nonce*-ova ima sačuvanih, to je veća šansa da dobije manji *deadline*. *Deadline* vrednost predstavlja broj sekundi od generisanja prethodnog bloka koji ostali rudari imaju da naprave novi validan blok. Ukoliko ne uspeju, rudar koji je izračunao taj *deadline* dobija nagradu i on je izabran da validira sledeći blok.

Iako je brži i pristupačniji od *PoW*-a, mogu se javiti slični problemi gde rudari masovno kupuju velike količine hard diskova ne bi li imali bolje šanse da dobiju *coinbase* nagradu. Takođe *malware* se dosta teže detektuje nego kod *PoW*-a, a i sam algoritam nije puno rasprostranjen među *blockchain* sistemima.

9.1.5. PROOF OF ELAPSED TIME

Dokaz proteklom vremenom (eng. *Proof of Elapsed Time – PoET*) je najviše fer algoritam od svih koje smo do sada pogledali, tj. svi čvorovi imaju podjednake šanse da budu izabrani kao validatori sledećeg bloka, međutim algoritam se u glavnom koristi na privatnim mrežama ili mrežama gde čvorovi nisu anonimni.

Algoritam je osmislio *Intel* 2016. godine. Funkcioniše tako što je svakom čvoru dodeljeno određeno nasumično vreme koje procesori moraju da budu u *sleep* režimu. Procesor koji se prvi probudi, tj. kome je bilo dodeljeno najmanje vreme dobija pravo kreiranja novog bloka i nagradu.

9.2. NAPADI NA BLOCKCHAIN

U teoriji *blockchain* je veoma sigurna struktura podataka, ali pošto se radi o distribuiranim decentralizovanim sistemima, postoji dosta napada na mrežnu infrastrukturu na kojoj je uspostavljena *blockchain* mreža i same čvorove koji su korisnici *blockchain* sistema. Takođe je od presudne važnosti koristiti veoma siguran kod (koji se ne može nikako zloupotребiti) prilikom implementacije bilo kod dela sistema, slično kao i kod bankarskog softvera, jer se radi o velikim količinama novca. U ovom poglavlju ćemo proći kroz četiri tipa napada.

U većini slučajeva cilj napada je krađa tuđih novčića, ometanje rada cele mreže ili takozvani *double-spending*, tj. mogućnost ponovnog korišćenja već iskorišćenih novčića, koje je

omogućeno zbog menjanja stanja unutar lanca i/ili mreže.

9.2.1. PEER-TO-PEER MREŽNI NAPADI

9.2.1.1. ECLIPSE ATTACK

Eclipse attack (napad pomračenjem) je slikovit opis za ovaj napad koji pokušava da odseče čvor metu od ostatka mreže. Svaki čvor u mreži ima određen broj suseda sa kojim je povezan i od kojih dobija informacije o ostatku mreže, jer naravno nije moguće da bude povezan sa apsolutno svim čvorovima koji čine mrežu. Ukoliko napadač uspe da uspostavi svoje maliciozne čvorove tako da čvor meta bude povezan samo sa njima, on će uspešno odseći dati čvor od ostatka mreže i “zamračiti” pravi lanac svojom nevalidnom verzijom lanca tako što će mu slati izmišljene informacije[23].

9.2.1.2. SYBIL ATTACK

Sybil attack je sličan *Eclipse*-u, ali sada je žrtva napada čitava mreža, a ne samo jedan čvor. Napadač pokušava da preplavi mrežu svojim čvorovima pod različitim pseudonimima koji ostatku mreže izgledaju kao međusobno nezavisni čvorovi (te stoga i ime *Sybil* prema knjizi u kojoj pacijent sa tim imenom ima višestruki poremećaj ličnosti). Cilj ovog napada je najčešće da se uništi reputacija i verodostojnost *blockchain* sistema tako što će se njegovi čvorovi ponašati tako da izazovu što više štete unutar mreže tako da validni čvorovi više ne znaju kome mogu da veruju[23].

9.2.2. NAPADI NA KONSENZUS MEHANIZAM I PROCES RUDARENJA

9.2.2.1. SELFISH-MINING ATTACK

Selfish-mining attack (napad sebičnim rudarenjem) kao preduslov uzima to da *blockchain* mreža smatra najduži lanac tačnom verzijom *ledger*-a što većina sistema i čini. Sebični rudar ovo može da iskoristi tako što tajno gradi svoju verziju lanca na vrhu postojeće verzije i onda ako dođe do prednosti od dva ili više bloka u odnosu na ostatak mreže on može da objavi svoj privatani *fork* koji će biti prihvaćen od ostatka mreže, jer predstavlja najdužu verziju lanca[23].

Ukoliko u trenutku pre objavljivanja svog dužeg tajnog lanca napadač oglasi transakcije u javnoj mreži on može u novoobjavljenoj verziji da poništi te iste transakcije i time uspešno izvrši *double-spending* u tom kratkom vremenskom periodu.

9.2.2.2. RUDARSKI MALWARE

Napadači mogu da zaraze tuđe uređaje malverom koji koristi kompjuterske resurse na uređaju u cilju rudarenja kriptovalute za napadače, a na novčanu štetu mete. Ovakvi malveri se

nazivaju kripto-majneri i ovo predstavlja najčešći napad, iako se ne napada sama infrastruktura, tj. ne narušava se *blockchain* sistem ni na koji način.

9.2.2.3. 51% ATTACK

51% napad spada među najpoznatije napade, a preduslov za njega jeste da jedan entitet (bilo pojedinac ili grupa) u svom vlasništvu ima više od 50% ukupnih mrežnih resursa za rudarenje blokova (npr. procesorske snage u *PoW* mrežama). Ovo je naravno veoma teško izvodljivo u većim mrežama, ali takođe i veoma čest problem kod malih mreža.

Kada entitet osigura većinu resursa za rudarenje on onda teoretski može da kontroliše blokčejn birajući koje transakcije će biti odabrane ili čak može ukloniti i neke već ranije procesuirane transakcije. Neke od kriptovaluta koje su pretrpele ovaj napad su *Ethereum Classic*, *Bitcoin Satoshi's Vision* (što je rezultovalo padom od 5% u vrednosti novčića) i *Bitcoin Gold Blockchain* (gde je ukradeno \$18M)[23].

Ranije u radu su pomenuti *pool*-ovi za rudarenje i njihovo učešće u prevenciji ovog problema je veoma značajno, jer se neretko može desiti da neki *pool* ima više od 50% resursa, ali u većini slučajeva se dobrovoljno ograničavaju tako da ostanu ispod te granice.

9.2.2.4. TIMEJACK ATTACK

Timejack attack (napad krađom vremena) je moguć kod blokčejnova koji funkcionišu pomoću sinhronizovane satnice (npr. *Bitcoin*), tj. gde se koriste vremenski žigovi kao *ID*-jevi blokova. U takvim mrežama je moguće ili promenom sistemskog vremena čvora ili ukoliko se čvorovi međusobno sinhronizuju prvo *Eclipse* napadom promeniti lokalno vreme nekog čvora koje taj čvor koristi prilikom provere vremenskih žigova pristižućih blokova. Kada se to dogodi čvor će odbacivati validne blokove koji dolaze od nemalicioznog ostatka mreže nakon čega je moguće izvršiti *double-spending* nad njim ili jednostavno smanjiti verodostojnost mreže pravljenjem masivne desinhronizovanosti među čvorovima[23].

9.2.2.5. FINNEY ATTACK

Finney napad je veoma sličan sebičnom rudarenju. Napadač u tajnosti izrudari blok sa jednom od svojih transakcija. Zatim pošalje čvoru trgovcu (npr. neka prodavnica koja prima takozvane munjevite transakcije – odmah prihvata transakciju pre nego što bude dodata u lanac) novu nepotvrđenu transakciju. Ukoliko čvor trgovac prihvati tu transakciju, onda napadač u veoma kratkom vremenskom periodu može da objavi izrudareni blok koji je čuvao u tajnosti i tako *double-spend*-uje upravo poslatu transakciju[23].

9.2.2.6. RACE ATTACK

Race attack (napad utrivanjem) je varijacija Finney napada gde napadač ne mora unapred

da izrudari blok, već može ukoliko je konektovan direktno sa žrtvom (čvorem trgovcem) da njoj pošalje jednu transakciju a ostatku mreže prosledi drugu. U tom slučaju žrtvi će izgledati kao da je prva transakcija bila upućena njoj i smatraće je validnom, iako to nije slučaj[23].

9.2.3. NAPADI KORIŠĆENJEM PAMETNIH UGOVORA

9.2.3.1. THE DAO ATTACK

Pametni ugovori su potpuno automatizovani ugovori i jednom kad otpočnu ne mogu biti zaustavljeni. *DAO (Decentralized Autonomous Organization)* je bila jedna od najambicioznijih stavki koje je *Ethereum* pokušao da implementira. Privatna kompanija je napravila *crowdfunding* za projekat u koji su korisnici *Ethereum*-a finansirali.

Međutim zbog postojanja *bug*-a u kodu, jedan korisnik je uspeo da rekurzivno poziva funkciju za povrat doniranog novca bez toga da ugovor proverava trenutno stanje donacione transakcije. Tako je inicijalno donirajući malu sumu, i zatim neprestanim pozivanjem funkcije za povrat doniranog novca uspeo da izvuče \$70M iz *crowdfunding*-a što je bilo skoro pola od ukupno prikupljenih \$150M (14% ukupnog *Ethereum*-a tada)[24].

Ethereum je zapretio napadaču da će mu zamrznuti nalog i novčanik međutim napadač se pozvao na to da radi sve što je dozvoljeno pametnim ugovorom i da će tužiti *Ethereum* ukoliko to urade. Kopija njegovog pisma se može pronaći na linku <https://pastebin.com/CcGUBgDG>.

Srećom po *Ethereum*, napadač je ipak prestao sa napadom ali ne pre nego što je ozbiljno zaljuljao pouzdanje javnosti u *Ethereum*, *blockchain* i pametne ugovore. *Ethereum* se odlučio da napravi *hard fork* tadašnjeg lanca kako bi povratili izgubljeni novac time kreirajući *Ethereum* i *Ethereum Classic*.

9.2.4. NAPADI NA NOVČANIKE

Naravno osim toga da napadač sazna kredencijale korisnika, moguće je i napasti infrastrukturu koja stoji iza novčanika ukoliko kod nije dovoljno siguran.

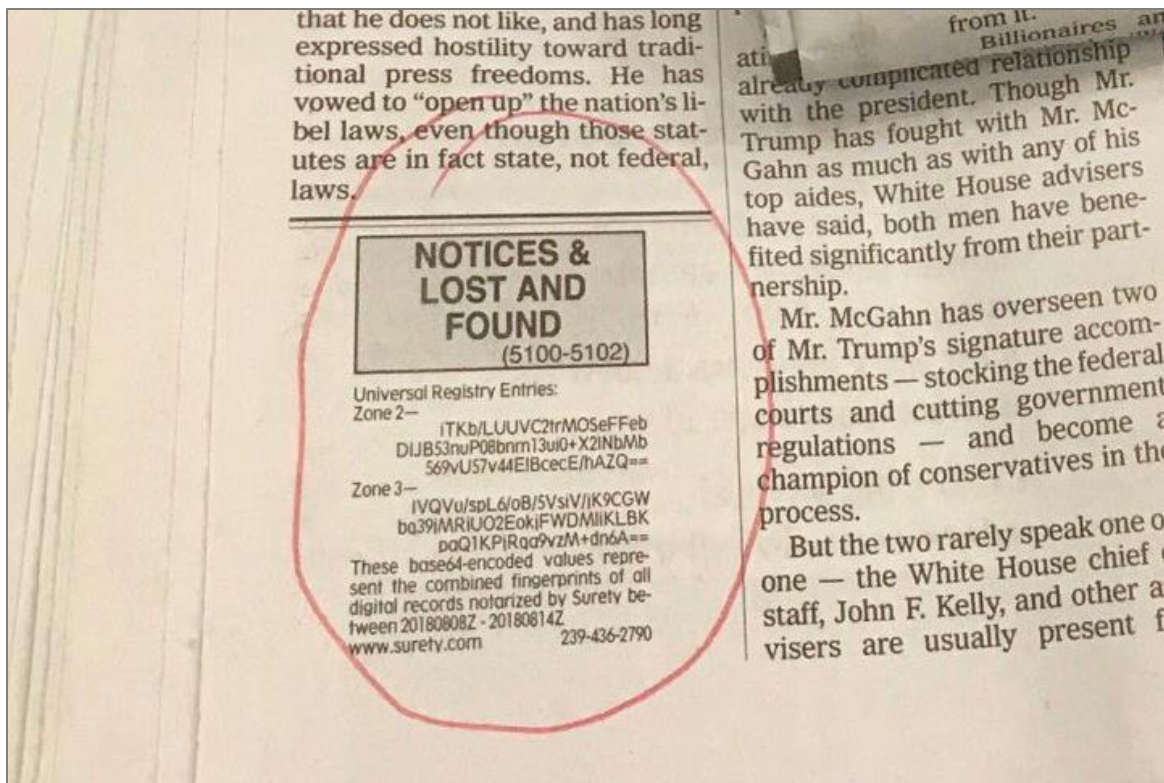
Jedan takav primer je zamrzavanje \$77M (\$155M tada) *Ethereum*-a zauvek u kriptografski nepristupačnim novčanicima. Klijent (softver) za pristup *Ethereum* novčanicima zvan *Parity* je koristio centralizovane bibliotetske ugovore radi smanjivanja transakcijskih troškova kod *Multisig* novčanika. *Multisig* novčanici su identični zajedničkim bankovnim računima, gde više korisnika deli isti račun.

Međutim kod u ovim bibliotekama uopšte nije bio ni približno dovoljno siguran, tako da je napadač lako uspeo da doda svoj nalog kao vlasnika bibliotetskog ugovora, tako da je postao zajednički vlasnik svih novčanika koji su izdati posle određenog datuma. Zatim je obrisao sve svoje naloge i time je 500,000 *Ether*-a ostalo zauvek zamrznuto[24].

9.3. ISTORIJA

Iako se *blockchain* kao termin koristi tek od skoro, njegova istorija počinje pre 40 godina. Trezorni sistem (eng. *vault system*) je tehnologija nalik *blockchain*-u koja se prvi put pominje još 1982. godine u doktorskoj disertaciji pod nazivom “Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups” od strane autora Dejvida Čauma (eng. *David Chaum*)[13]. Dejvid se smatra pioninom kriptografije i tehnologija koje čuvaju privatnost i izumiteljem digitalnog novca. U svojoj disertaciji dao je kod pomoću kog bi se implementirao protokol kao i skoro sve elemente *blockchain*-a koje danas koristi *Bitcoin*, čak 27 godina pre nego što je on uopšte i izumljen.

Sledeći veliki korak u razvoju *blockchain*-a daju 1991. god. kriptografi Stjuart Hejber (eng. *Stuart Haber*) i Skot Storneta (eng. *Scott Stornetta*). Prvobitna upotreba ove verzije *blockchain*-a nije imala veze sa novcem, već su njegovi tvorci zamislili da se on koristi za dokazivanje autentičnosti digitalnih dokumenata korišćenjem vremenskih oznaka (eng. *timestamp*) i kriptografije. Oni su prvi implementirali Merkleova stabla u ove svrhe, a *New York Times* još od 1995. god. koristi njihovu ideju za dokazivanje autentičnosti digitalnih dokumenata heširanjem[14].



Slika 48. New York Times koristi Merkleovo stablo heširanih digitalnih dokumenata kao dokaz njihove autentičnosti

Slika preuzeta sa Twitter-a od korisnika @SovereignCarel: <https://twitter.com/SovereignCarel/status/1226436702752866304>

Dalji doprinos razvoju *blockchain*-a dali su u svojim radovima iz polja kriptografije 1996. Ros Anderson (eng. *Ross Anderson*), a potom 1998. Brus Šnajer (eng. *Bruce Schneier*) i Džon Kelsi (eng. *John Kelsey*)[13].

1998. god. Nik Šabo (eng. *Nick Szabo*) daje teorijsku osnovu za jednu od prvih kriptovaluta ikada zvanu *bit gold*[15]. Iako nikad nije zapravo bila implementirana, u teoriji ona koristi dosta mehanizama koje i današnje stvarne kriptovalute koriste. Nik je takođe dao teorijsku osnovu za pametne ugovore (eng. *smart contracts*) koje danas koriste novije verzije kriptovaluta.

2000. god. Stefan Konst (eng. *Stefan Konst*) objavljuje u naučnom radu način rada kriptografski ulančanih lanaca podataka i daje svoje ideje za implementaciju istog[15].

Konačno 2008. god. objavljen je rad koji čini prekretnicu u ovom polju od strane anonimnog lica ili grupe ljudi pod pseudonimom Satoši Nakamoto (eng. *Satoshi Nakamoto*). Naziv rada je “Bitcoin: A peer-to-peer electronic cash system” i u njemu je opisan *blockchain* kao protokol i decentralizovana distribuirana baza podataka koja omogućava korišćenje danas najpoznatije kriptovalute *Bitcoin*[2]. Rad se može pronaći ovde: <https://bitcoin.org/bitcoin.pdf>.

2009. god. Nakamoto implementira prvi moderan *blockchain* koji je opisao u svom radu prethodne godine[15]. Satoši je u svom radu zasebno koristio reči *block* i *chain*, a termin *blockchain* se pojavio tek oko 2014. godine.

Danas se Satošijeva verzija *blockchain*-a naziva *Blockchain 1.0*[17]. On je unapredio prvobitnu Dejvid Čaumovu zamisao koristeći mehanizam koji se zove dokaz radom (eng. *Proof of Work*) i dodao parametar težine (eng. *difficulty*) kako bi ograničio količinu kriptovalute koja ulazi u cirkulaciju.

Na slici 49 prikazan je izgled sadržaja prvog bloka *Bitcoin*-ovog *blockchain*-a. Prvi blok u lancu se zove blok postanka (eng. *genesis block*) i on u većini *blockchain* implementacija ima unapred zadate vrednosti promenljivih u kodu (eng. *hardcoded*). Zanimljiv detalj je da je Nakamoto ubacio poruku u dve promenljive (coinbase i potpis) koja glasi “The Times 03/Jan/2009 Chancellor on brink of second bailout for banks” referišući jedan od naslova na naslovnoj stranici čuvenih britanskih novina *The Times*[21]. Postoje razne spekulacije oko značenja ove poruke, ali je sigurno da je ta poruka služila, između ostalog, kao dokaz da blok nije nastao pre 3. januara 2009. godine.

neprimetno uključivanje korporativnih aplikacija i baza podataka u decentralizovane sisteme radi sigurnosti i transparentnosti.

9.4. BLOKOVI KODA

```
+---node_modules
|   |   .bin
|   |   ...uvezeni Node.js moduli (ima ih oko 200)
|   |   .package-lock.json
+---public
|   +---icons
|   |   |   chain.svg
|   +---javascripts
|   |   |   blockchain.js
|   |   |   chf.js
|   |   \---lib
|   |           BigInteger.min.js
|   |           bootstrap.bundle.min.js
|   |           buffer.js
|   |           crypto-js.js
|   |           elliptic.min.js
|   |           jquery-3.6.0.min.js
|   +---stylesheets
|   |   |   blockchain.css
|   |   \---lib
|   |           bootstrap-horizon.css
|   |           bootstrap.min.css
|   \---videos
|           Network.mp4
+---views
|   |   about.pug
```

```

| | block.pug
| | blockchain.pug
| | chf.pug
| | coinbase.pug
| | distributed.pug
| | index.pug
| | keys.pug
| | layout.pug
| | mining.pug
| | miningdiff.pug
| | network.pug
| | references.pug
| | signatures.pug
| | signed_txs_and_ids.pug
| | transactions.pug
| \---includes
| |         coinbaseblock.pug
| |         miningblock.pug
| |         signedandidblock.pug
| |         simpleblock.pug
| |         transactionblock.pug
| .gitignore
| .bs-config.js
| .index.js
| .package-lock.json
\ .package.json

```

Blok koda 1. Organizaciona struktura fajlova

```

{
  "name": "blockchain-virtualization",

```

```

"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
  "dev": "nodemon ./index.js",
  "test": "echo \\\"Error: no test specified\\\" && exit 1",
  "ui": "browser-sync start --config bs-config.js"
},
"keywords": [],
"author": "Dimitrije Knezevic",
"license": "UNLICENSED",
"private": "true",
"devDependencies": {
  "browser-sync": "^2.27.9",
  "nodemon": "^2.0.15"
},
"dependencies": {
  "bootstrap-horizon": "^1.0.1",
  "buffer": "^6.0.3",
  "crypto-js": "^4.1.1",
  "express": "^4.17.3",
  "jquery": "^3.6.0",
  "pug": "^3.0.2"
}
}

```

Blok koda 2. Izgled fajla package.json

```

/**
 * Required External Modules
 */
const express = require("express");
const path = require("path");
/**
 * App Variables
 */
const app = express();
const port = process.env.PORT || "8000";
/**
 * App Configuration
 */
app.set("views", path.join(__dirname, "views"));

```

```

app.set("view engine", "pug");
app.use(express.static(path.join(__dirname, 'public')));
/**
 * Routes Definitions
 */
app.get("/", (req, res) => {
    res.render("index", { title: "Home" });
});
app.get('/:page', function(req, res) {
    res.render(req.params.page, {page: req.params.page, title: req.params.page});
});
/**
 * Server Activation
 */
app.listen(port, () => {
    console.log(`Listening to requests on http://localhost:${port}`);
});

```

Blok koda 3. Sadržaj fajla *index.js*

```

doctype html
html
  head
    meta(charset="utf-8")
    link(rel="shortcut icon", href="/icons/chain.svg")
    meta(name="viewport", content="width=device-width, initial-scale=1, shrink-to-fit=no")
    title Blockchain Virtualization - #{title}
    link(rel='stylesheet', href='stylesheets/blockchain.css')
    link(rel='stylesheet', href='stylesheets/lib/bootstrap-horizon.css')
    link(rel='stylesheet', href='stylesheets/lib/bootstrap.min.css')
    script(src='javascripts/lib/jquery-3.6.0.min.js')
    script(src='javascripts/lib/bootstrap.bundle.min.js')
    script(src='javascripts/lib/crypto-js.js')
    script(src='javascripts/lib/bigInteger.min.js')
    script(src='javascripts/lib/elliptic.min.js')
    script(src='javascripts/lib/buffer.js')
    script(src='javascripts/chf.js')
    script(src='javascripts/blockchain.js')

  body.d-flex.flex-column.min-vh-100
    nav.navbar.navbar-expand-lg.navbar-dark.bg-secondary

```



```


container
  a.navbar-brand(href='/')
    img(src="/icons/chain.svg", width="42", height="42")
    button.navbar-toggler(type='button', data-bs-toggle="collapse", data-bs-
target="#navbar", aria-controls="navbar", aria-expanded="false", aria-label="Toggle
navigation")

    span.navbar-toggler-icon
  div#navbar.collapse.navbar-collapse
    div.navbar-nav.me-auto
      a.nav-link(href='/chf', class={active: page === 'chf'}) Hash
      a.nav-link(href='/block', class={active: page === 'block'}) Block
      a.nav-link(href='/blockchain', class={active: page ===
'blockchain'}) Blockchain
      a.nav-link(href='/transactions', class={active: page ===
'transactions'}) Transactions
      a.nav-link(href='/coinbase', class={active: page === 'coinbase'})
Coinbase
      a.nav-link(href='/keys', class={active: page === 'keys'}) Keys
      a.nav-link(href='/signatures', class={active: page ===
'signatures'}) Signatures
      a.nav-link(href='/signed_txs_and_ids', class={active: page ===
'signed_txs_and_ids'}) Signed TxS
      a.nav-link(href='/miningdiff', class={active: page ===
'miningdiff'}) Difficulty
      a.nav-link(href='/mining', class={active: page === 'mining'})
Mining
      a.nav-link(href='/distributed', class={active: page ===
'distributed'}) Distributed
      a.nav-link(href='/network', class={active: page === 'network'})
Network
    div.navbar-nav
      a.nav-link(href='/references', class={active: page ===
'references'}) References
      a.nav-link(href='/about', class={active: page === 'about'}) About
    block layout-content
  footer.bg-light.text-center.mt-auto
    div.text-center.p-3.bg-light System for Visual Representation of Blockchain
Technology © 2022 Dimitrije Knezevic


```

Blok koda 4. Sadržaj fajla layout.pug

```

module.exports = {

```

```

proxy: "localhost:8000",
files: ["**/*.css", "**/*.pug", "**/*.js"],
ignore: ["node_modules"],
reloadDelay: 10,
ui: false,
notify: false,
port: 3000,
};

```

Blok koda 5. Sadržaj *bs-config.js* fajla

```

$('#sha256').val(sha256());
function sha256(chain, block) { return CryptoJS.SHA256(getText(chain, block)); }
function getText(chain, block){ return $('#data').val(); }

```

Blok koda 6. Primer poziva i korišćenja sha256 funkcije

```

(function (Math) {
    // Shortcuts
    var C = CryptoJS;
    var C_lib = C.lib;
    var WordArray = C_lib.WordArray;
    var Hasher = C_lib.Hasher;
    var C_algo = C.algo;

    // Initialization and round constants tables
    var H = [];
    var K = [];

    // Compute constants
    (function () {
        function isPrime(n) {
            var sqrtN = Math.sqrt(n);
            for (var factor = 2; factor <= sqrtN; factor++) {
                if (!(n % factor)) { return false; }
            }
            return true;
        }
    })
}

```

```

function getFractionalBits(n) {
    return ((n - (n | 0)) * 0x100000000) | 0;
}

var n = 2;
var nPrime = 0;
while (nPrime < 64) {
    if (isPrime(n)) {
        if (nPrime < 8) {
            H[nPrime] = getFractionalBits(Math.pow(n, 1 / 2));
        }
        K[nPrime] = getFractionalBits(Math.pow(n, 1 / 3));
        nPrime++;
    }
    n++;
}
})();

// Reusable object
var W = [];

/**
 * SHA-256 hash algorithm.
 */
var SHA256 = C_algo.SHA256 = Hasher.extend({
    _doReset: function () {
        this._hash = new WordArray.init(H.slice(0));
    },

    _doProcessBlock: function (M, offset) {
        // Shortcut
        var H = this._hash.words;

        // Working variables
        var a = H[0];
        var b = H[1];
        var c = H[2];
        var d = H[3];
        var e = H[4];
        var f = H[5];
        var g = H[6];
        var h = H[7];

        // Computation

```

```

for (var i = 0; i < 64; i++) {
    if (i < 16) {
        W[i] = M[offset + i] | 0;
    } else {
        var gamma0x = W[i - 15];
        var gamma0 = ((gamma0x << 25) | (gamma0x >>> 7)) ^
            ((gamma0x << 14) | (gamma0x >>> 18)) ^
            (gamma0x >>> 3);

        var gamma1x = W[i - 2];
        var gamma1 = ((gamma1x << 15) | (gamma1x >>> 17)) ^
            ((gamma1x << 13) | (gamma1x >>> 19)) ^
            (gamma1x >>> 10);

        W[i] = gamma0 + W[i - 7] + gamma1 + W[i - 16];
    }

    var ch = (e & f) ^ (~e & g);
    var maj = (a & b) ^ (a & c) ^ (b & c);

    var sigma0 = ((a << 30) | (a >>> 2)) ^ ((a << 19) | (a >>> 13)) ^
        ((a << 10) | (a >>> 22));
    var sigma1 = ((e << 26) | (e >>> 6)) ^ ((e << 21) | (e >>> 11)) ^
        ((e << 7) | (e >>> 25));

    var t1 = h + sigma1 + ch + K[i] + W[i];
    var t2 = sigma0 + maj;

    h = g;
    g = f;
    f = e;
    e = (d + t1) | 0;
    d = c;
    c = b;
    b = a;
    a = (t1 + t2) | 0;
}

// Intermediate hash value
H[0] = (H[0] + a) | 0;
H[1] = (H[1] + b) | 0;
H[2] = (H[2] + c) | 0;
H[3] = (H[3] + d) | 0;
H[4] = (H[4] + e) | 0;
H[5] = (H[5] + f) | 0;

```

```

        H[6] = (H[6] + g) | 0;
        H[7] = (H[7] + h) | 0;
    },

    _doFinalize: function () {
        // Shortcuts
        var data = this._data;
        var dataWords = data.words;

        var nBitsTotal = this._nDataBytes * 8;
        var nBitsLeft = data.sigBytes * 8;

        // Add padding
        dataWords[nBitsLeft >>> 5] |= 0x80 << (24 - nBitsLeft % 32);
        dataWords[(((nBitsLeft + 64) >>> 9) << 4) + 14] =
Math.floor(nBitsTotal / 0x100000000);
        dataWords[(((nBitsLeft + 64) >>> 9) << 4) + 15] = nBitsTotal;
        data.sigBytes = dataWords.length * 4;

        // Hash final blocks
        this._process();

        // Return final computed hash
        return this._hash;
    },

    clone: function () {
        var clone = Hasher.clone.call(this);
        clone._hash = this._hash.clone();

        return clone;
    }
});

/**
 * Shortcut function to the hasher's object interface.
 * @param {WordArray|string} message The message to hash.
 * @return {WordArray} The hash.
 * @static
 * @example
 * var hash = CryptoJS.SHA256('message');
 * var hash = CryptoJS.SHA256(wordArray);
 */
C.SHA256 = Hasher._createHelper(SHA256);

```

```

/**
 * Shortcut function to the HMAC's object interface.
 * @param {WordArray|string} message The message to hash.
 * @param {WordArray|string} key The secret key.
 * @return {WordArray} The HMAC.
 * @static
 * @example
 * var hmac = CryptoJS.HmacSHA256(message, key);
 */
C.HmacSHA256 = Hasher._createHmacHelper(SHA256);
})(Math));

```

Blok koda 7. Implementacija *CryptoJS.SHA256* funkcije

```

$('#magic').bind('input propertychange', function() {
    this.value = this.value.replace(/^[^0-9]/g, '');
    $('#hash').val(sha256());
});
$('#magic').bind('focusout', function() {
    if(this.value===''){
        this.value='0';
        $('#hash').val(sha256());
    }
});
$('#data').bind('input propertychange', function() {
    $('#hash').val(sha256());
});

```

Blok koda 8. Primer korišćenja lambda funkcija vezanih za određene akcije

```

div.card.bg-light(id='chain'+simpleblock.chain+'block'+simpleblock.block+'card')
    div.card-body
        form
            div.mb-3.row
                label.col-sm-2.control-label.pt-
2(for='chain'+simpleblock.chain+'block'+simpleblock.block+'magic') Magic:
                div.col-sm-10
                    div.input-group
                        span.input-group-text #

```

```

        input.form-
control(id='chain'+simpleblock.chain+'block'+simpleblock.block+'magic', type='text',
value=simpleblock.magic)

    div.mb-3.row
        label.col-sm-2.control-label.pt-
2(for='chain'+simpleblock.chain+'block'+simpleblock.block+'blockid') Block:
        div.col-sm-10
            div.input-group
                span.input-group-text #
                input.form-
control(id='chain'+simpleblock.chain+'block'+simpleblock.block+'blockid',
type='text', value=simpleblock.blockid)

    div.mb-3.row
        label.col-sm-2.col-form-
label(for='chain'+simpleblock.chain+'block'+simpleblock.block+'data') Data:
        div.col-sm-10
            textarea.form-
control(id='chain'+simpleblock.chain+'block'+simpleblock.block+'data',rows='10')

    div.mb-3.row
        label.col-sm-2.control-label.pt-
2(for='chain'+simpleblock.chain+'block'+simpleblock.block+'prev') Prev:
        div.col-sm-10
            input.form-
control(id='chain'+simpleblock.chain+'block'+simpleblock.block+'prev', type='text',
value=simpleblock.prev)

    div.mb-3.row
        label.col-sm-2.control-label.pt-
2(for='chain'+simpleblock.chain+'block'+simpleblock.block+'hash') Hash:
        div.col-sm-10
            input.form-
control(id='chain'+simpleblock.chain+'block'+simpleblock.block+'hash', type='text',
disabled)

```

Blok koda 9. Sadržaj *simpleblock.pug* fajla

```

div.row.row-horizon(style='flex-wrap: nowrap')
  - var blocks = []
  - blocks.push({chain: 1, block: 1, magic: 74567, blockid: 1, prev:
'0000000000000000000000000000000000000000000000000000000000000000'})
  - blocks.push({chain: 1, block: 2, magic: 74567, blockid: 2, prev:
'050a3c1fe50deb0b49aa719b33f55e99114e6ce00e10ef855cb6bb5ea2a0624'})
  - blocks.push({chain: 1, block: 3, magic: 74567, blockid: 3, prev:
'91b8325cfe25967ce30e8d6ea5d124b2c026695aedfc5dc6cc7068fd5277119b'})
  - blocks.push({chain: 1, block: 4, magic: 74567, blockid: 4, prev:
'3eae7011c3e4525f8d5f221f8c60e3facec881040caf74abd3b648ff835cf3a3'})
  - blocks.push({chain: 1, block: 5, magic: 74567, blockid: 5, prev:
'4be027c3c4df33d226a352d337ba400dc7097c085632ade089bc56ff9c1e47eb'})
  - blocks.push({chain: 1, block: 6, magic: 74567, blockid: 6, prev:
'dfbfed7dd4c6e972ffad17325d87cc82f3a5663aea49344f8dd658b645f8251a'})
  each simpleblock in blocks
    div.col-lg-6
      include includes/simpleblock

```

Blok koda 10. Korišćenje *pug* fajla unutar drugog *pug* fajla

```

function validMagic(chain, block){
  if(block==1){ return true; }
  return $('#chain'+chain+'block'+block+'magic').val()==$('#chain'+chain+'block'+
(block-1).toString()+ 'magic').val();
}

```

Blok koda 11. Funkcija koja proverava validnost magičnog broja

```

function validBlockId(chain, block){
  if(block==1){ return true; }
  return parseInt($('#chain'+chain+'block'+block+'blockid').val())==parseInt
($('#chain'+chain+'block'+(block-1).toString()+ 'blockid').val()+1;
}

```

Blok koda 12. Funkcija koja proverava validnost broja bloka


```

function validPrev(chain, block){
    if(block==1){ return true; }
    return $('#chain'+chain+'block'+block+'prev').val()=='#chain'+chain+'block'+
(block-1).toString()+hash'.val();
}

```

Blok koda 13. Funkcija koja proverava validnost heševa uzastopnih blokova

```

function updateBlockchain(chain, block, chainSize) {
    updateHash(chain, block);
    for(var i = 1; i<=chainSize; i++){
        if(validMagic(chain, i) && validBlockId(chain, i) && validPrev(chain, i)){ //no
errors in the current block
            $('#chain'+chain+'block'+i+'magic').removeClass('input-error');
            $('#chain'+chain+'block'+i+'blockid').removeClass('input-error');
            $('#chain'+chain+'block'+i+'prev').removeClass('input-error');
            $('#chain'+chain+'block'+(i-1).toString()+hash').removeClass('input-error');

            if($('#chain'+chain+'block'+(i-1).toString()+card').hasClass('card-error')){
//previous card had an error
                $('#chain'+chain+'block'+i+'card').removeClass('bg-light').addClass('card-
error');
            }
            else{ //previous card had NO errors
                $('#chain'+chain+'block'+i+'card').removeClass('card-error').addClass('bg-
light');
            }
        }
        else{ //error(s) in the current block
            $('#chain'+chain+'block'+i+'card').removeClass('bg-light').addClass('card-
error');

            if(!validPrev(chain, i)){
                $('#chain'+chain+'block'+(i-1).toString()+card').removeClass('bg-
light').addClass('card-error');
                $('#chain'+chain+'block'+(i-1).toString()+hash').addClass('input-error');
                $('#chain'+chain+'block'+i+'prev').addClass('input-error');
            }
            else{ //previous and current hash match
                $('#chain'+chain+'block'+(i-1).toString()+hash').removeClass('input-
error');

```

```

        $('#chain'+chain+'block'+i+'prev').removeClass('input-error');
    }

    if(!validMagic(chain, i)){
        $('#chain'+chain+'block'+i+'magic').addClass('input-error');
    }
    else{ //magic number has NO error
        $('#chain'+chain+'block'+i+'magic').removeClass('input-error');
    }

    if(!validBlockId(chain, i)){
        $('#chain'+chain+'block'+i+'blockid').addClass('input-error');
    }
    else{ //block id has NO error
        $('#chain'+chain+'block'+i+'blockid').removeClass('input-error');
    }
}
}
}

```

Blok koda 14. Funkcija koja ažurira izgled i heš vrednosti blokova u lancu

```

.card-error{
    background-color: rgba(250, 200, 200)!important;
}

.input-error{
    color: rgb(230, 10, 10)!important;
    font-weight: 600!important;
    border: 2px solid rgb(230, 10, 10)!important;
}

```

Blok koda 15. CSS stilovi koji se primenjuju na nevalidne elemente u lancu

```

div.mb-3.row
    label.col-sm-2.col-form-label(for='chain'+transactionblock.chain+
'block'+transactionblock.block+'tx') Tx:
    div.col-sm-10
        each tx, i in transactionblock.txs

```

```


input-group
  span.input-group-text £
  input.form-control(id='chain'+transactionblock.chain+
'block'+transactionblock.block+'tx'+i+'value', type='text', value=tx.value)
  span.input-group-text From:
  input.form-control(id='chain'+transactionblock.chain+
'block'+transactionblock.block+'tx'+i+'from', type='text', value=tx.from)
  span.input-group-text To:
  input.form-control(id='chain'+transactionblock.chain+
'block'+transactionblock.block+'tx'+i+'to', type='text', value=tx.to)


```

Blok koda 16. Uvedena izmena u *transactionblock.pug* u odnosu na *simpleblock.pug*

```

- blocks.push({
chain: 1,
block: 3,
magic: 74567,
blockid: 3,
txs: [
  {value: '12.50', from: 'Andrei', to: 'Nichole'},
  {value: '3.44', from: 'Yamato', to: 'George'},
  {value: '9.03', from: 'Yamato', to: 'Mark'},
  {value: '25.00', from: 'Luke', to: 'Andrei'},
  {value: '25.00', from: 'Luke', to: 'Yamato'},
  {value: '18.67', from: 'Andrei', to: 'George'}
],
prev: 'ef610b751a42cb4875fd229547893360e369e8792e51002 c34d28a523ccb0851'
})

```

Blok koda 17. Primer prenosa transakcija između roditeljske stranice i stranice deteta

```

function validCoinbase(chain, block){
  return parseFloat($('#chain'+chain+'block'+block+'coinvalue').val())==
parseFloat(20.0/Math.pow(2,parseInt((block-1)/5)));
}

```

Blok koda 18. Funkcija koja proverava validnost *Coinbase* polja

```

function validRef(chain, block, tx){
  if(block==1){
    return true;
  }

  var spentSum = parseFloat($('#chain'+chain+'block'+block+'tx'+tx+'value').val()+
parseFloat($('#chain'+chain+'block'+block+'tx'+tx+'returnedValue').val());
  var allowedSum = 0;
  const sender = $('#chain'+chain+'block'+block+'tx'+tx+'from').val();
  const refArray = $('#chain'+chain+'block'+block+'tx'+tx+'ref').val().split(' ');
  refArray.sort(function(a, b){return a - b});
  //array is empty || there are duplicates || referenced current block or greater
  if(refArray.length==0 || (new Set(refArray)).size!==refArray.length ||
refArray[refArray.length-1]>=block){
    return false;
  }

  var j = 1;
  for(let i = block-1; i>=refArray[0] || j<=refArray.length; i--){
    if(i==refArray[refArray.length-j]){ //if this block is referenced
      j++;
      let senderFound = false;
      if($('#chain'+chain+'block'+i+'cointo').val()==sender){ //add coinbase
        senderFound=true;
        allowedSum+=parseFloat($('#chain'+chain+'block'+i+'coinvalue').val());
      }

      for(let k = 0; $('#chain'+chain+'block'+i+'tx'+k+'value').length>0; k++){
        //go through all tx in that block
        if($('#chain'+chain+'block'+i+'tx'+k+'to').val()==sender){
          allowedSum+=parseFloat($('#chain'+chain+'block'+i+'tx'+k+'value').val());
          senderFound = true;
        }

        if($('#chain'+chain+'block'+i+'tx'+k+'returnedTo').val()==sender){
          allowedSum+=parseFloat($('#chain'+chain+'block'+i+'tx'+k+'returnedValue').v
al());
          senderFound = true;
        }
      }

      if(!senderFound){
        return false;
      }
    }
  }
}

```

```

    }
    else{ //if this block isn't referenced
        for(let k = 0; $('#chain'+chain+'block'+i+'tx'+k+'value').length>0; k++){
            //go through all tx in that block
            if($('#chain'+chain+'block'+i+'tx'+k+'from').val()==sender){
                let kRefArray = $('#chain'+chain+'block'+i+'tx'+k+'ref').val().split(' ');
                let tempArray = refArray;
                if(tempArray.filter(x=>kRefArray.includes(x)).length!=0){
                    //if arrays overlap
                    return false;
                }
            }
        }
    }
}

//basic transaction (no id's and signatures)
function validTx(chain, block){
    if(block==1){
        return true;
    }
    for(let k = 0; $('#chain'+chain+'block'+block+'tx'+k+'value').length>0; k++){ //go
through all tx in block
        if(!validReturnedTo(chain, block, k) || !validReturnedFrom(chain, block, k) ||
!validRef(chain, block, k) || !validValue(chain, block, k)){
            return false;
        }
    }
    return true;
}
}

```

Blok koda 19. Funkcije koja proveravaju validnost jednostavnih transakcija i *References* polja

```

var EC = elliptic.ec;
var ec = new EC('secp256k1');
var keypair = ec.genKeyPair();

function random() {
    keypair = ec.genKeyPair();
    var prv = keypair.getPrivate('hex');
    $('#private').val(bigInt(prv, 16).toString());
    update();
}

```

```

}

$(function() {
  random();
  $('#randomButton').click(random);
  $('#private').bind('input propertychange', function() {
    this.value = this.value.replace(/^[0-9]/g, '');
    if(this.value==' ' || this.value==0)
      this.value='1';
    keypair = ec.keyFromPrivate(bigInt($('#private').val()).toString(16));
    update();
  });
});

function update() {
  var pub = keypair.getPublic('hex');
  $('#public').val(pub);
}

```

Blok koda 20. Kod za generisanje javnog ključa na osnovu privatnog pozivom funkcija iz *elliptic.min.js*

```

//sign and send button
$('#signButton').click(function(){
  resetReceiverColor();
  $('#receiverMsg').val($('#senderMsg').val())
  var binaryMsg = buffer.Buffer.from(CryptoJS.SHA256($('#senderMsg').val()).
toString(CryptoJS.enc.Hex));
  var hexSignature = buffer.Buffer.from(keypair.sign(binaryMsg).toDER()).
toString('hex');
  $('#senderSignature').val(hexSignature);
  $('#receiverSignature').val(hexSignature);
  update();
});

//verify button
$('#verifyButton').click(function(){
  var temp;
  try{
    var temp=ec.keyFromPublic($('#public').val(),'hex');
    var binaryMsg=buffer.Buffer.from(CryptoJS.SHA256($('#receiverMsg').val()).
toString(CryptoJS.enc.Hex));
    if(temp.verify(binaryMsg,$('#receiverSignature').val())){

```

```

        $('#receiverCard').removeClass('bg-light').removeClass('card-error').
addClass('card-success');
    }
    else{
        $('#receiverCard').removeClass('bg-light').removeClass('card-success').
addClass('card-error');
    }
}
catch(e){
    $('#receiverCard').removeClass('bg-light').removeClass('card-success').
addClass('card-error');
}
});

```

Blok koda 21. Funkcije koje implementiraju najbitnije funkcionalnosti na *Signatures* stranici

```

//go through previous/current blocks and find previous id
function validId(chain, block, tx){
    if(block==1){
        return true;
    }
    const currId=parseInt($('#chain'+chain+'block'+block+'tx'+tx+'id').val());
    const sender=$('#chain'+chain+'block'+block+'tx'+tx+'from').val();
    var senderFound=false;
    var prevId=-1;
    for(let i=block; i>=1; i--){
        if(i==block){//curr block
            for(let j=tx-1;j>=0;j--){
                if(sender==$('#chain'+chain+'block'+i+'tx'+j+'from').val()){
                    return currId==(parseInt($('#chain'+chain+'block'+i+'tx'+j+'returnedId').
val()+1));
                }
            }
        }
        else{//previous blocks
            for(let j = 0; $('#chain'+chain+'block'+i+'tx'+j+'value').length>0; j++){ //go
through all tx in that block
                if(sender==$('#chain'+chain+'block'+i+'tx'+j+'from').val()){
                    senderFound=true;
                    prevId=parseInt($('#chain'+chain+'block'+i+'tx'+j+'returnedId').val());
                }
            }
        }
    }
}

```

```

        if(senderFound==true){
            return currId==(prevId+1);
        }
    }
}
return currId==1;
}

function validReturnedId(chain, block, tx){
    if(block==1){
        return true;
    }
    return
parseInt($('#chain'+chain+'block'+block+'tx'+tx+'id').val()+1==parseInt($('#chain'+chain+'block'+block+'tx'+tx+'returnedId').val()));
}

function validSignature(chain,block,tx){
    var EC = elliptic.ec;
    var ec = new EC('secp256k1');
    var temp;
    var msg=$('#chain'+chain+'block'+block+'tx'+tx+'id').val()+
        $('#chain'+chain+'block'+block+'tx'+tx+'value').val()+
        $('#chain'+chain+'block'+block+'tx'+tx+'from').val()+
        $('#chain'+chain+'block'+block+'tx'+tx+'to').val();

    try{
        var
temp=ec.keyFromPublic($('#chain'+chain+'block'+block+'tx'+tx+'from').val(),'hex');
        var binaryMsg=buffer.Buffer.from(CryptoJS.SHA256(msg).
toString(CryptoJS.enc.Hex));
        //console.log('Converted msg to binary');
        if(temp.verify(binaryMsg,$('#chain'+chain+'block'+block+'tx'+tx+'signature')
.val())){
            return true;
        }
        else{
            return false;
        }
    }
    catch(e){
        return false;
    }
}

function validReturnedSignature(chain,block,tx){

```



```

var EC = elliptic.ec;
var ec = new EC('secp256k1');
var temp;
var msg=$('#chain'+chain+'block'+block+'tx'+tx+'returnedId').val()+
    $('#chain'+chain+'block'+block+'tx'+tx+'returnedValue').val()+
    $('#chain'+chain+'block'+block+'tx'+tx+'returnedFrom').val()+
    $('#chain'+chain+'block'+block+'tx'+tx+'returnedTo').val();

try{
    var temp=ec.keyFromPublic($('#chain'+chain+'block'+block+'tx'+tx+
'returnedFrom').val(),'hex');
    var binaryMsg=buffer.Buffer.from(CryptoJS.SHA256(msg).
toString(CryptoJS.enc.Hex));
    if(temp.verify(binaryMsg,$('#chain'+chain+'block'+block+'tx'+tx+
'returnedSignature').val())){
        return true;
    }
    else{
        return false;
    }
}
catch(e){
    return false;
}
}

//transaction with id's and signatures
function validSignedTx(chain, block){
    if(block==1){
        return true;
    }
    for(let k = 0; $('#chain'+chain+'block'+block+'tx'+k+'value').length>0; k++){ //go
through all tx in block
        if(!validReturnedTo(chain, block, k) ||
            !validReturnedFrom(chain, block, k) ||
            !validRef(chain, block, k) ||
            !validValue(chain, block, k) ||
            !validReturnedId(chain, block, k) ||
            !validId(chain,block,k) ||
            !validSignature(chain,block,k) ||
            !validReturnedSignature(chain,block,k)){
            return false;
        }
    }
}
return true;

```

```
}
```

Blok koda 22. Funkcije koje proveravaju validnost potpisa i ID-ja transakcije

```
const txtAvg=document.getElementById('textAvg')
const txtPrc=document.getElementById('textPerc')
$(function() {
  update();
  $('#difficulty').bind('input propertychange', function() {
    this.value = this.value.replace(/^[0-9]/g, '');
    if(parseInt(this.value)>256){
      this.value='256';
    }
    update();
  });

  $('#difficulty').bind('focusout', function() {
    if(this.value=='')
      this.value='0';
    update();
  });
});

function update() {
  var diff= parseInt($('#difficulty').val());
  var numOfZerosHex=diff/4;

  var hashHex = "0".repeat(numOfZerosHex);
  switch(diff%4){
    case 0:hashHex+="";break;
    case 1:hashHex+="7";break;
    case 2:hashHex+="3";break;
    case 3:hashHex+="1";break;
  }
  hashHex+="f".repeat(64-numOfZerosHex);
  $('#maxhashhex').val(hashHex);

  var hashBin="0".repeat(diff)+"1".repeat(256-diff);
  $('#maxhashbin').val(hashBin);

  txtAvg.textContent='On average ONE out of every 2' + diff + ' = '+Math.pow(2,diff)+
  ' hash strings is valid'
```

```
txtPrc.textContent='Chance to find a valid hash: '+100.0/Math.pow(2,diff) + '%'  
}
```

Blok koda 23. Funkcije pomoću kojih su realizovana računanja maksimalnih heš vrednosti na osnovu težine, i promene labela

```
//diff  
$('#chain'+chain+'block'+block+'diff').bind('input propertychange', function() {  
    this.value = this.value.replace(/^[^0-9]/g, '');  
    if(parseInt(this.value)>16){  
        this.value=16;  
    }  
    updateMiningBlockchain(chain, block, #{blocks.length});  
});  
  
$('#chain'+chain+'block'+block+'diff').bind('focusout', function() {  
    if(this.value==''){  
        this.value='0';  
        updateMiningBlockchain(chain, block, #{blocks.length});  
    }  
    if(parseInt(this.value)>16){  
        this.value=16;  
        updateMiningBlockchain(chain, block, #{blocks.length});  
    }  
});  
  
//nonce  
$('#chain'+chain+'block'+block+'nonce').bind('input', function() {  
    this.value = this.value.replace(/^[^0-9]/g, '');  
    updateMiningBlockchain(chain, block, #{blocks.length});  
});  
  
$('#chain'+chain+'block'+block+'nonce').bind('focusout', function() {  
    if(this.value==''){  
        this.value='0';  
        updateMiningBlockchain(chain, block, #{blocks.length});  
    }  
});
```

Blok koda 24. Funkcije odgovorne za obradu korisničkog unosa za polja *Difficulty* i *Nonce*

```

//mine button
$('#chain'+chain+'block'+block+'mineBtn').bind('click',function() {
  let spinner=document.getElementById('chain'+chain+'block'+block+'spinner');
  spinner.removeAttribute('hidden');
  setTimeout(()=>{
    mine(chain,block,{blocks.length});
    spinner.setAttribute('hidden','true');
  },400);
});

function mine(chain, block, chainSize){
  x=0;
  while(true){
    $('#chain'+chain+'block'+block+'nonce').val(x++);
    updateHash(chain,block);
    if(validNonce(chain,block)){
      updateMiningBlockchain(chain, block, chainSize);
      break;
    }
  }
}

```

Blok koda 25. Funkcije za rudarenje

```

function updateDistributedBlockchain(chain,block,chainSize){
  updateMiningBlockchain(chain,block,chainSize);
  valid1=true;
  valid2=true;
  valid3=true;
  if(!distributedBlockchainIsValid(1,chainSize)){
    $('#peerA').removeClass('text-success').addClass('text-danger');
    valid1=false;
  }
  if(!distributedBlockchainIsValid(2,chainSize)){
    $('#peerB').removeClass('text-success').addClass('text-danger');
    valid2=false;
  }
  if(!distributedBlockchainIsValid(3,chainSize)){
    $('#peerC').removeClass('text-success').addClass('text-danger');
    valid3=false;
  }
}

```

```

if(valid1){
    if(!valid2 && !valid3){//only peer A is valid
        $('#peerA').removeClass('text-danger').addClass('text-success');
        return;
    }

    if(chainsMatch(1,2,chainSize) || chainsMatch(1,3,chainSize)){
        $('#peerA').removeClass('text-danger').addClass('text-success');
    }
    else{
        $('#peerA').removeClass('text-success').addClass('text-danger');
    }
}

if(valid2){
    if(!valid1 && !valid3){//only peer B is valid
        $('#peerB').removeClass('text-danger').addClass('text-success');
        return;
    }

    if(chainsMatch(2,1,chainSize) || chainsMatch(2,3,chainSize)){
        $('#peerB').removeClass('text-danger').addClass('text-success');
    }
    else{
        $('#peerB').removeClass('text-success').addClass('text-danger');
    }
}

if(valid3){
    if(!valid1 && !valid2){//only peer C is valid
        $('#peerC').removeClass('text-danger').addClass('text-success');
        return;
    }

    if(chainsMatch(3,1,chainSize) || chainsMatch(3,2,chainSize)){
        $('#peerC').removeClass('text-danger').addClass('text-success');
    }
    else{
        $('#peerC').removeClass('text-success').addClass('text-danger');
    }
}
}

function distributedBlockchainIsValid(chain,chainSize){

```

```

for(var i = 1; i<=chainSize; i++){ //go through curr chain
    if(!validMagic(chain, i) ||
        !validBlockId(chain, i) ||
        !validPrev(chain, i) ||
        !validCoinbase(chain, i) ||
        !validSignedTx(chain, i) ||
        !validNonce(chain,i)){
        return false;
    }
}
return true;
}

function chainsMatch(chainA, chainB, chainSize){
    for(var i = 1; i<=chainSize; i++){
        if($('#chain'+chainA+'block'+i+'hash').val()!=$('#chain'+chainB+'block'+i+'hash')
        .val()){
            return false;
        }
    }
    return true;
}

```

Blok koda 26. Funkcije koje određuju validnosti *blockchain*-ova i čvorova u distribuiranom sistemu