Merkleovo stablo

Stipanić, Tomislav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

MERKLEOVO STABLO

Završni rad

Tomislav Stipanić

Osijek, 2019.

SADRŽAJ:

1. UVOD

Merkleovo stablo, nazvano po tvorcu (Ralph Merkle, 1979.), je pojam koji se sve više spominje u svijetu računarstva, naime riječ je o strukturi podataka koja se koristi iza mnogih kriptovaluta, među kojima je i Bitcoin. Samo stablo je konstruirano od podataka podijeljenih u pakete (listovi stabla) čije rekurzivno hashiranje po parovima daje Merkleov korijen.

U nastavku rada daljnje je analizirana te uz dijagrame opisana struktura stabla te definirana svojstva koja ga određuju poput hashiranja, provjere podataka i provjere dosljednosti. Također, u narednim poglavljima opširno su opisana svojstva Merkleovog stabla unutar blockchaina, definiran je rad blockchaina kao i karakteristike kriptovaluta koje se na njemu zasnivaju poput Bitcoina i Ethereuma.

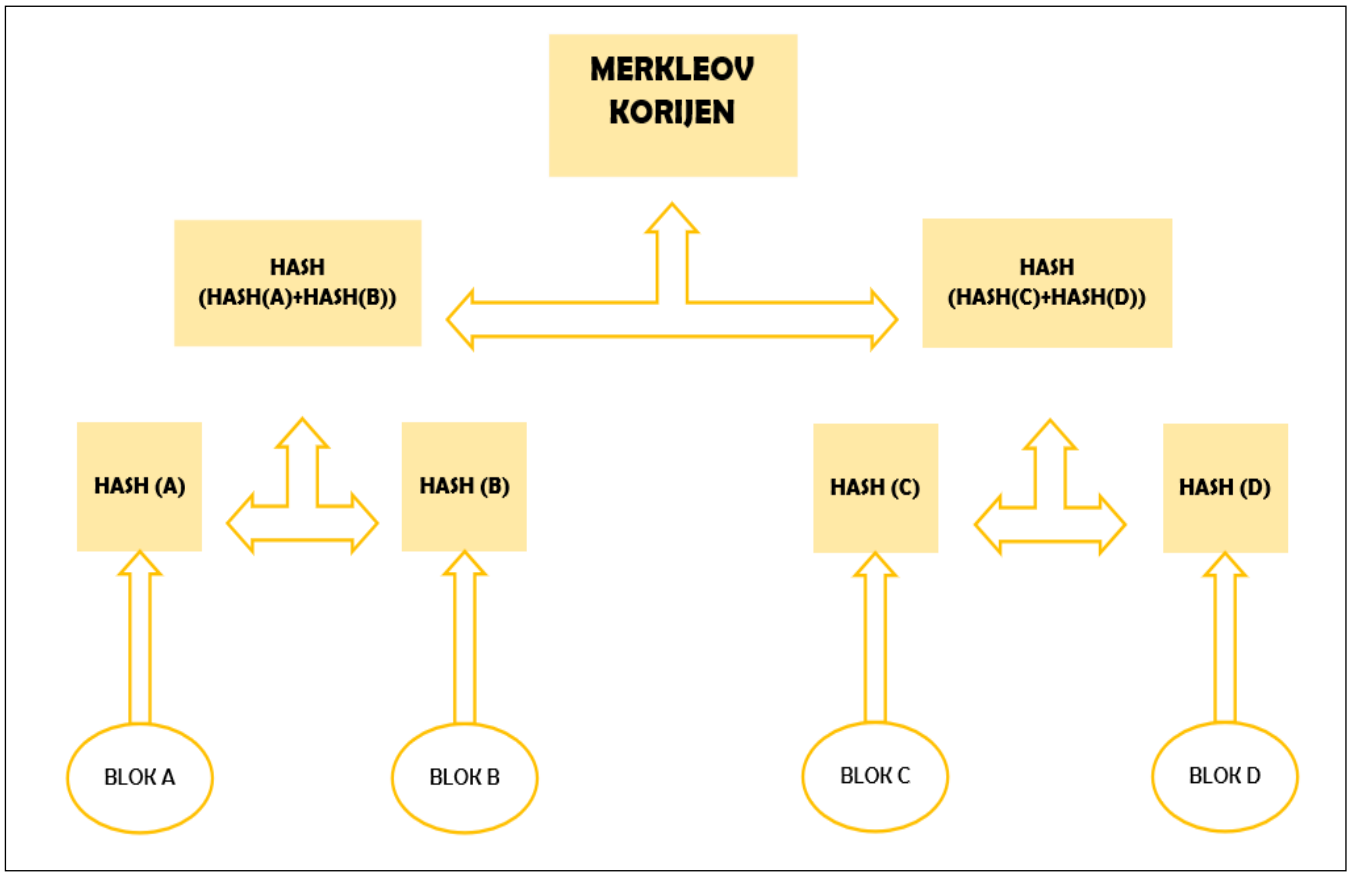
1.1. Zadatak završnog rada

U blockchain tehnologiji Merkleovo stablo ili hash stablo je struktura u kojoj je svaki čvor obilježen hashem njegove djece-čvorova i kao takav okosnica je svih kripto valuta. U radu potrebno je objasniti osnovna svojstva blockchain tehnologije i ulogu ovog stabla.

2. STRUKTURA STABLA

Termin „stablo“ se u računarstvu koristi za opis strukture podataka koja sadrži složena grananja. Stablo je fleksibilno nelinearna struktura podataka između kojih postoji hijerarhijski odnos „podređeni-nadređeni“ ili „dijete-roditelj“. [1]

Merkleovo stablo, poznato i kao *hash* stablo, je struktura podataka čiji je svaki list označen s kriptografskim hashem paketa podataka te čiji je svaki nadređeni čvor označen kriptografskim hashem oznaka svojih podređenih. [2]



Sl. 2.1. *Shema Merkleovog stabla*

2.1. *Hash* funkcija

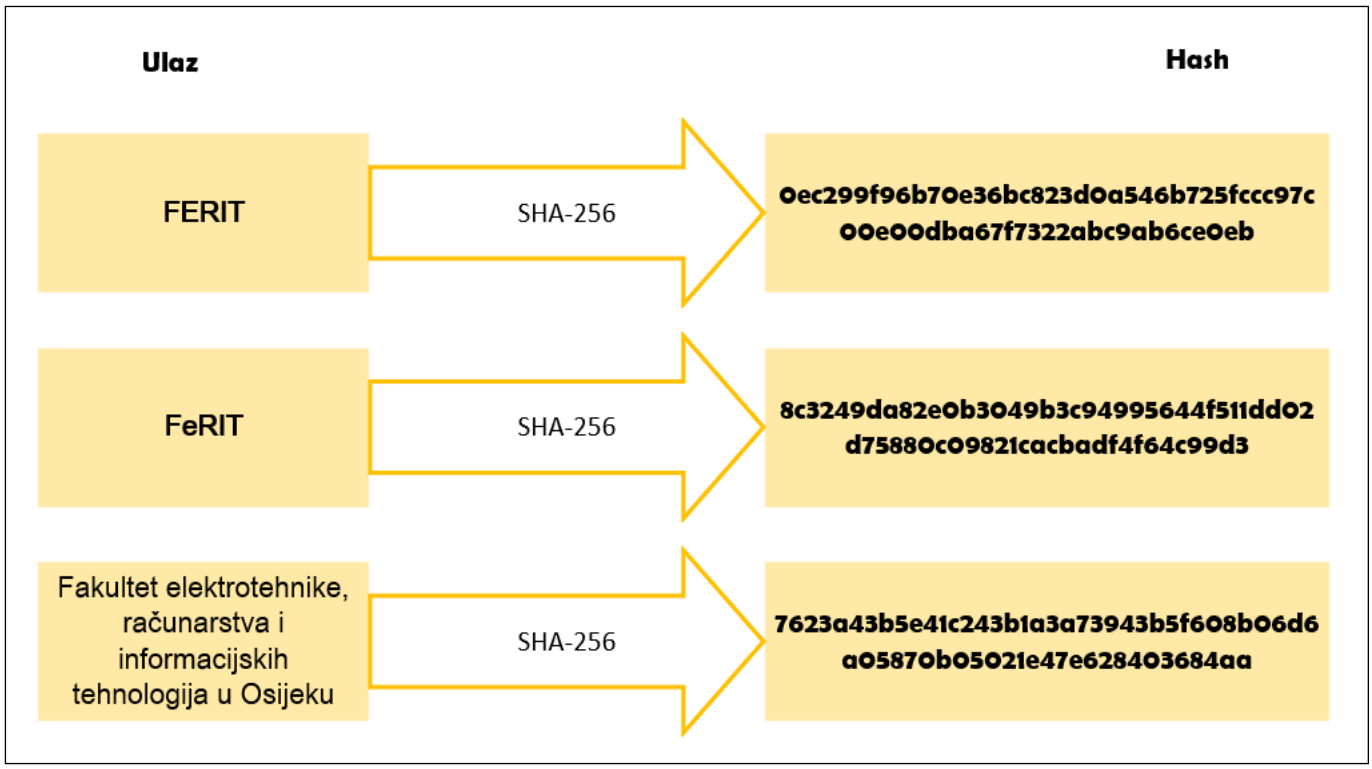
Hash i hashiranje su pojmovi koji se vrlo često pojavljuju u opisu Merkleovog stabla, stoga je potrebno upoznati se s njima na samome početku. Naime, cijeli princip Merkleovog stabla zasniva se na *hash* funkciji.

Hash funkcija je kriptografski algoritam koji za ulaz prima određene podatke, dok za izlaz izbacuje niz znakova s fiksnom dužinom koji se naziva *hash*. Hashiranje je proces generiranja vrijednosti iz niza znakova koristeći matematičke funkcije. Predstavlja način osiguravanja zaštićenog procesa prijenosa poruke koja je namijenjena određenoj osobi.

Kada korisnik šalje zaštićenu poruku potrebno je generirati i hashirati poruku te dobiveni *hash* poslati zajedno s porukom. Kada je poruka primljena na drugom kraju, primatelj hashira primljenu poruku te dobiveni hash uspoređuje s primljenim hashem. Ako se hashevi poklapaju, riječ je o izvornoj poruci pošiljatelja. Pomoću hashiranja se može utvrditi da nitko neovlašten nije mijenjao značenje poruke. [3]

Većina hashing algoritama su javni te su dostupni svakome. Svaki ulaz kroz hashing algoritam će imati jedinstveni i specifični izlaz baš za taj ulaz. Čak i kada se isti ulaz ponovi više puta, hashing funkcija će uvijek dati jednaki izlaz koji je određen ulazom. Ali ako je ulaz i najmanje promijenjen, *hash* će biti prezentiran kao potpuno drugačiji niz znakova, što znači da mala promjena ulaza neće rezultirati maloj promjeni izlaza, već potpunom promjenom. [4]

Hash funkcija je „*one way function*“ što znači da se može koristiti samo u jednom smjeru, drugim riječima, hashirana poruka ne može biti „de-hashirana“. Poruku kriptiranu pomoću hash funkcije nije moguće dekriptirati.



Sl. 2.2. *Primjer ulaza/izlaza hash funkcije SHA-256* [5]

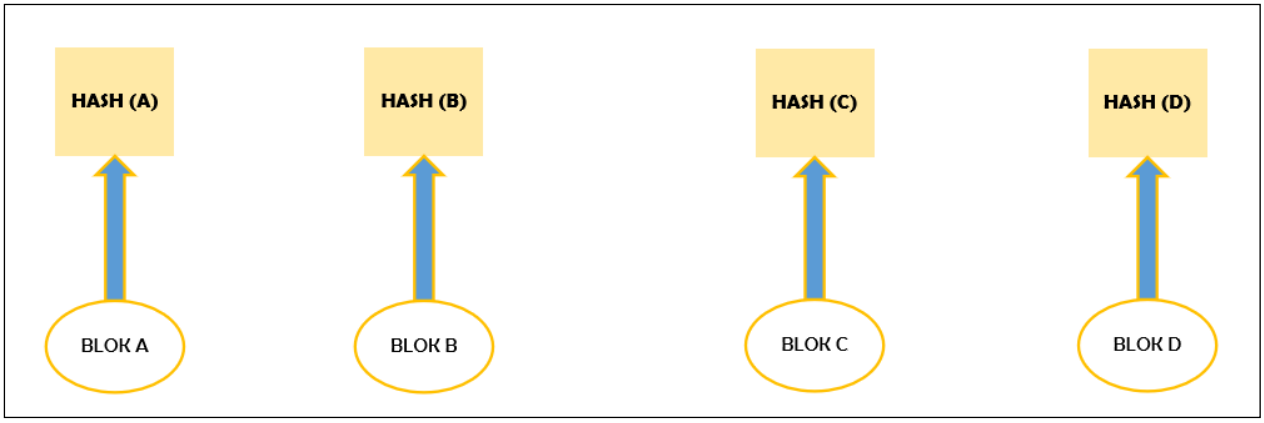
Kao što je prikazano na slici 2.2. promjena samo jednog znaka na ulazu rezultira s naizgled potpuno nasumičnim nizom znakova. Također, iz primjera valja uočiti da dužina znakovnog niza na ulazu u *hash* funkciju nije bitna. *Hash* će uvijek imati konstantan broj znakova.

Postoje razni kriptografski *hash* algoritmi s različitim svojstvima i karakteristikama. Hash algoritam koji se najviše ističe danas je SHA-256. *Secure Hash Algoritam – 256 bit* ili skraćeno SHA-256, je tip *hash* funkcije koji se koristi u blockchainu ili točnije Bitcoinu. Razvila ju je Američka sigurnosna agencija (NSA) 2002. Godine.

2.2. Struktura

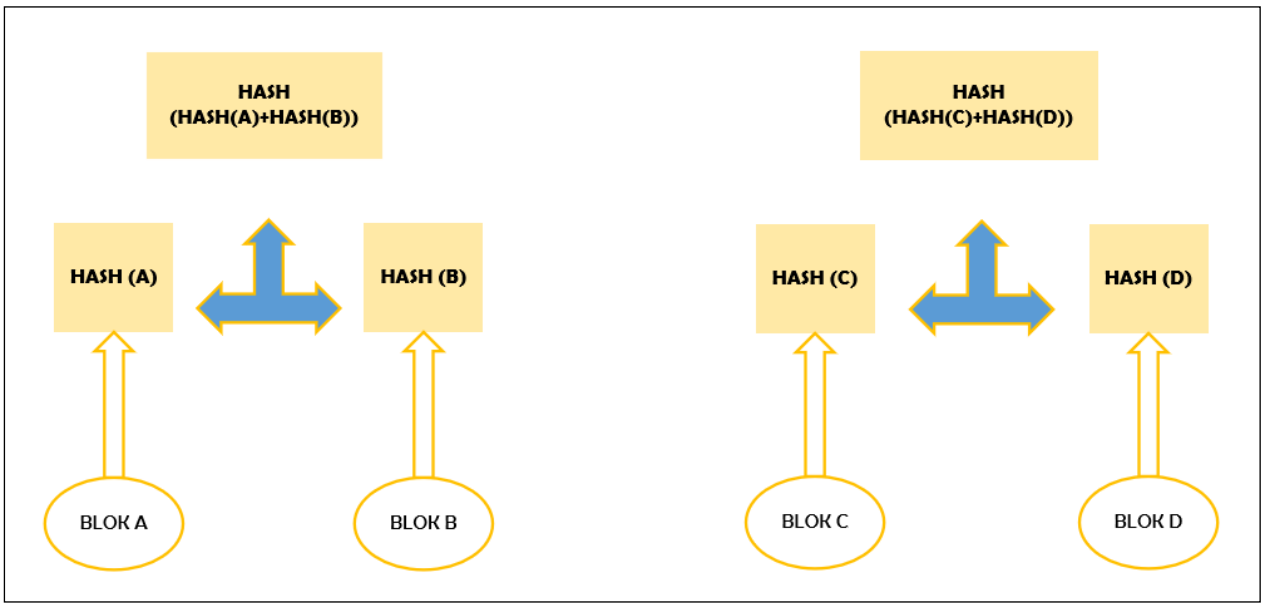
Stablo se može podijeliti na grane, čvorove i korijen. Merkleovo stablo se formira od dna prema gore, što znači da u početku imamo samo pakete podataka(transakcija) koje je potrebno kriptografski hashirati.

Znajući što je *hash* funkcija, koncept merkleovog stabla postaje jednostavan. Merkleovo stablo nije ništa drugo nego uzastopno hashiranje podataka sve dok naposljetku ne dobijemo krajnji *hash* kojeg nazivamo korijen hasha ,to jest Merkleov korijen.



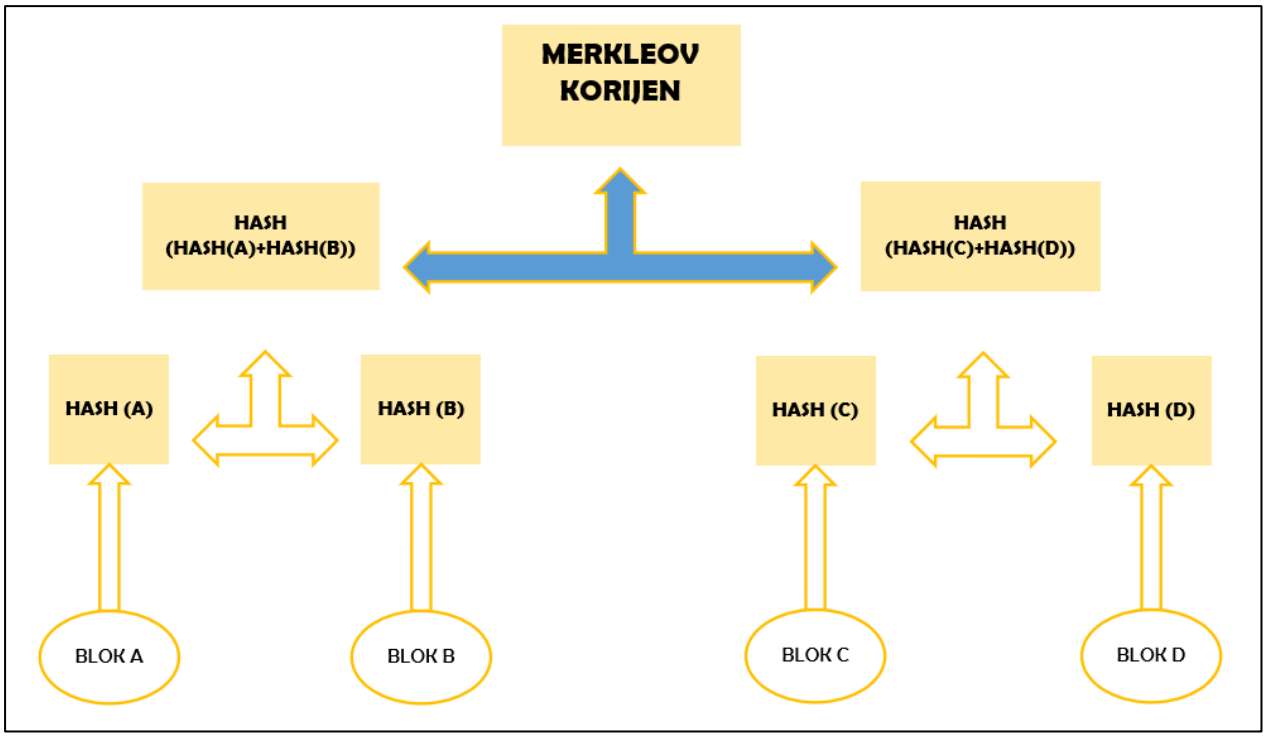
Sl. 2.3. *Hashiranje paketa podataka*

Uzmimo primjer sa slike 2.3. Postoje četiri paketa podataka (A,B,C,D) gdje je svaki paket zasebno hashiran, nakon čega će se hashevi navedenih paketa upariti stvarajući novi *hash*. Pa je tako *hash*AB nastao hashiranjem uparenog podređenog para.(Sl. 2.4.).



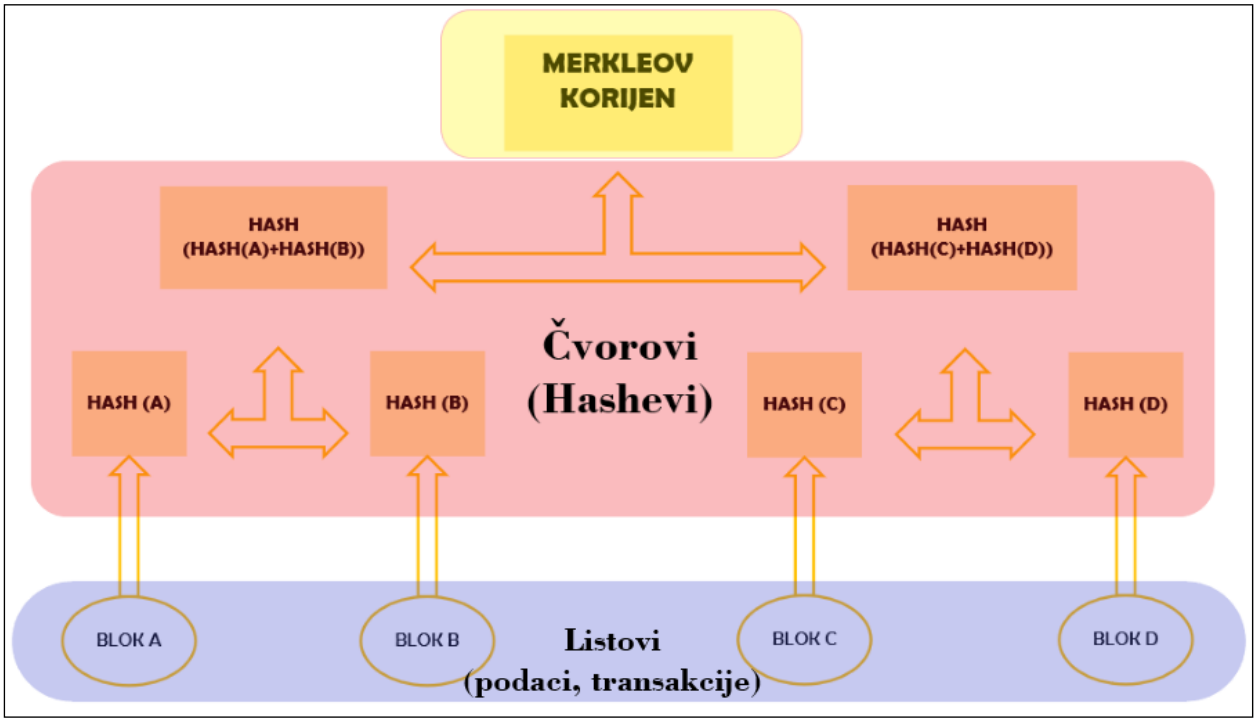
Sl. 2.4. *Hashiranje hasheva*

Svaki sljedeći čvor uključujući i Merkleov korijen nastaje istim principom, kao produkt *hash* funkcije svoje djece. (Sl 2.5.)



Sl. 2.5. *Merkleov korijen*

Kada se novi paket podataka pridružuje stablu, on nastavlja logički redoslijed stabla, te pronalazi svoju poziciju kao posljednji list stabla.



Sl. 2.6. *Kategorizacija stabla*

U transakcijskim sustavima, poput kriptovaluta, se događa da kada se navedeni paket podataka jednom pridruži određenom stablu na određenoj pozicija, ta pozicija postaje konačna te se ne može naknadno mijenjati. Promjena poprima oblik novog paketa podataka koji se, kao i svaki novi paket, dodaje na dno stabla. Suprotno tome, u sustavima poput NoSQL baze podatka, gdje je moguće naknadno mijenjati pakete podatka. Promjena će pokrenuti lančanu reakciju koje će početi promjenom jednog hasha te naposljetku cijelog stabla. Tada se Merkleovo stablo koristi kako bi se brzo i učinkovito pronašlo promijenjeni podatak te naknadno sinkroniziralo cijelo stablo. [6]

3. SVOJSTVA

Merkleovo stablo je svojim svojstvima postavilo temelje za tehnologiju blockchaina koja je u sve većem usponu. U bliskoj budućnosti možemo očekivati još mnoštvo ideja i aplikacija koje se baziraju baš na blockchainu, odnosno Merkleovom stablu. U nastavku su detaljno opisana svojstva koja doprinose poželjnim karakteristikama Merkleovog stabla:

-Uvelike smanjuje količinu podataka koje je potrebno provjeriti kako bi se dokazala dosljednost i integritet podataka

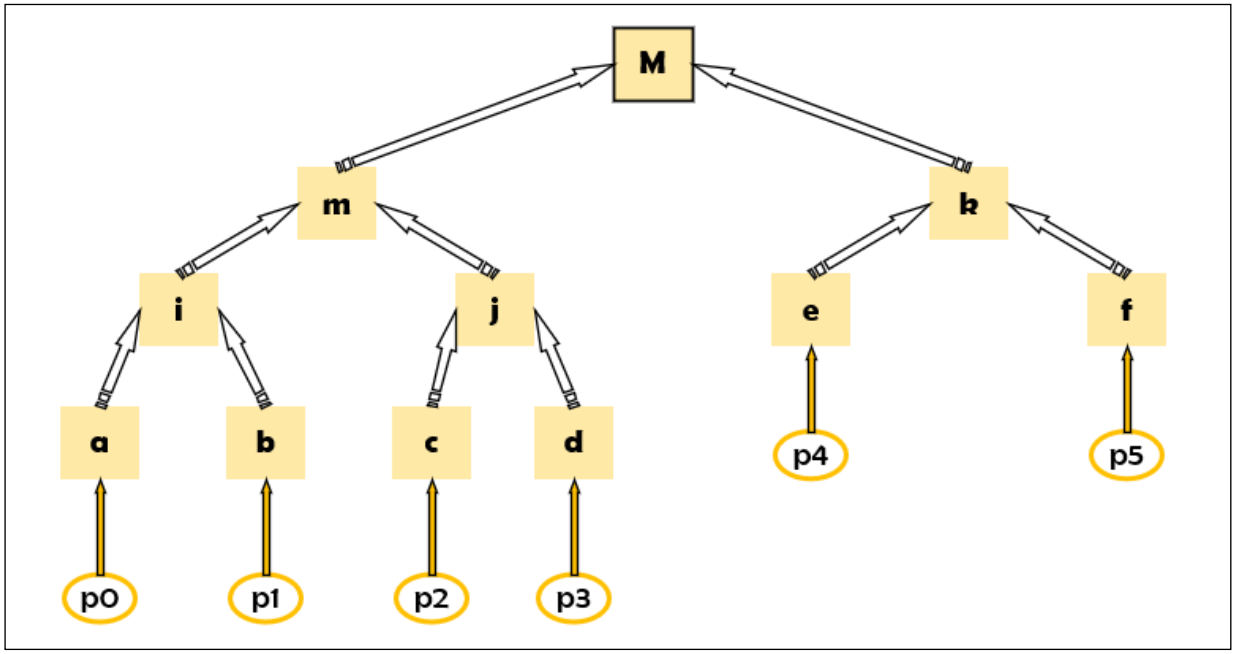
- Provjere su brze te zahtijevaju malo memorije

-Drastično smanjuje U/I veličinu paketa potrebnu za potvrdu podataka i dosljednosti [6]

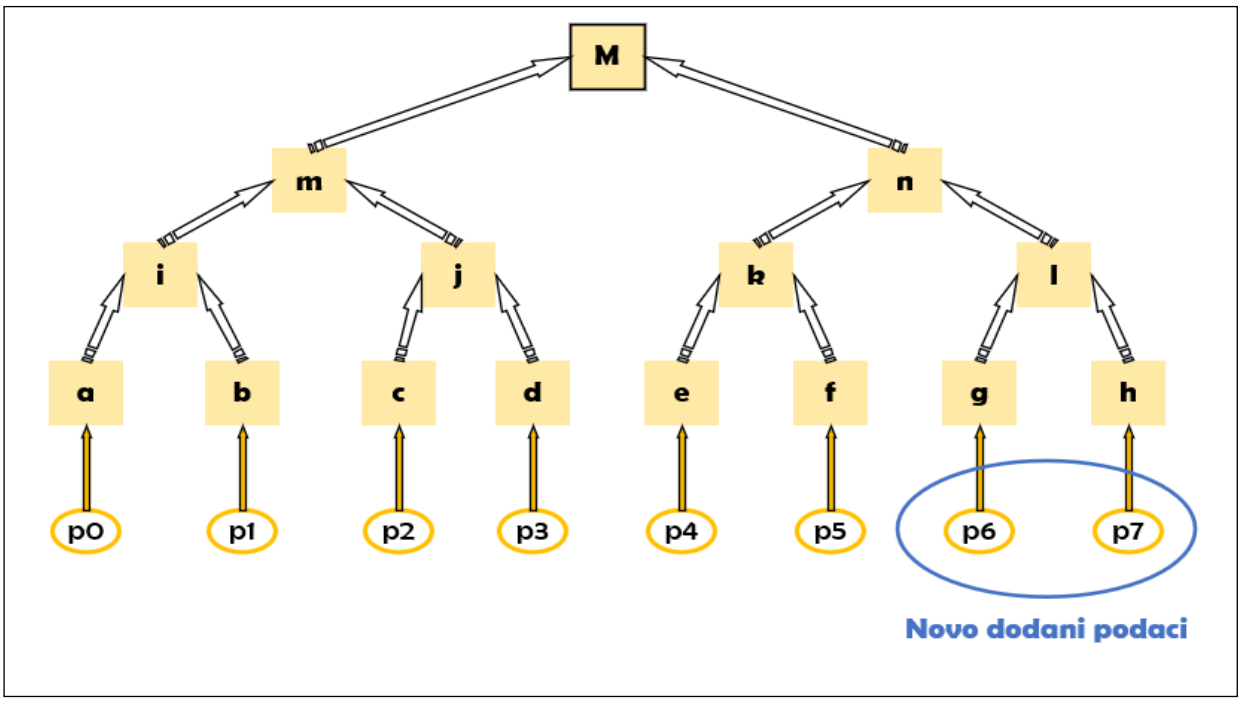
3.1. Provjera dosljednosti

Merkleova provjera dosljednosti omogućava provjeru dosljednost između dvije verzije stabala. Drugim riječima, potvrđuje da kasnija verzija stabla sadrži sve podatke kao i ranija verzija, istim redoslijedom, te da svi noviji unosi dolaze nakon unosa iz ranije verzije. Ako provjera uspije potvrditi navedene stavke to znači da se niti jedan podatak nije *back-date*o i umetnuo u stablo.[7]

Koncept provjere dosljednosti je prikazan sljedećim primjerom na slikama 3.1. i 3.2.
Slika 3.2. predstavlja proširenu verziju stabla sa slike 3.1. koja je dobivena dodavanjem novih paketa podataka (p6, p7). Cilj ove provjere je potvrditi da je prvo stablo zaista podstablo drugoga te da su svi novo dodani podaci uneseni posljednji.

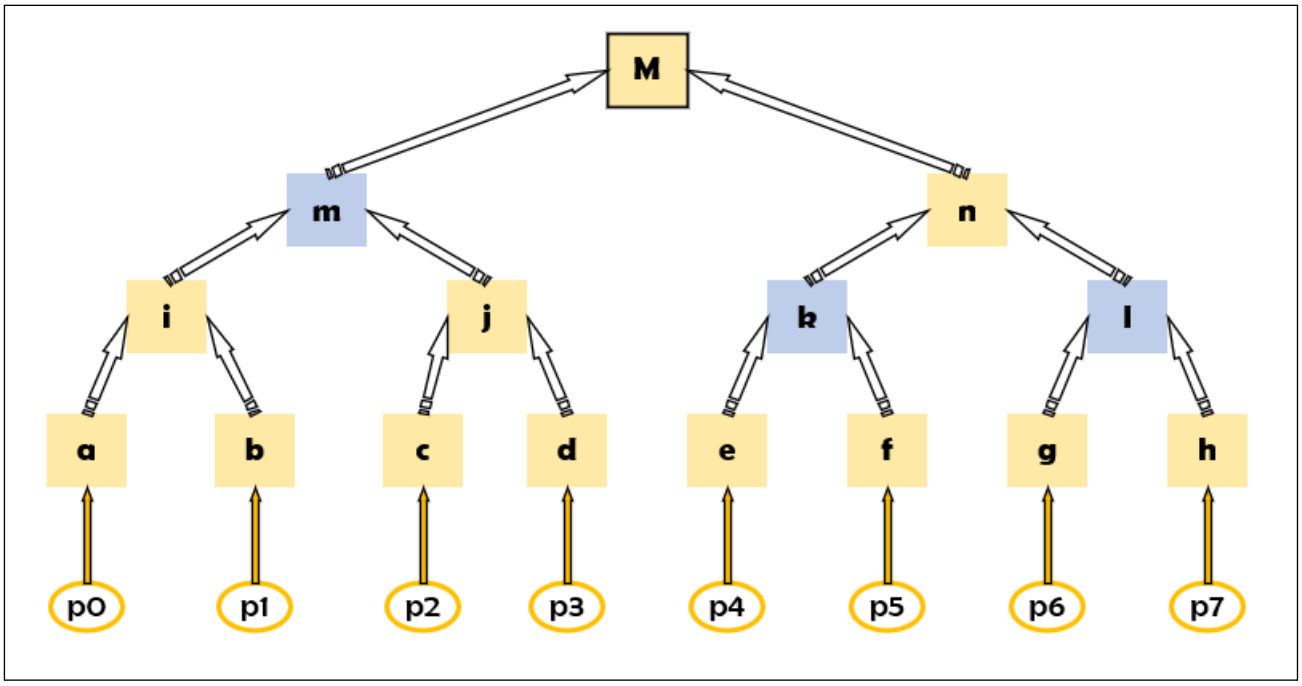


***Sl. 3.1**. Dosljednost podataka „Prvo stablo“*



***Sl. 3.2.** Dosljednost podataka „Drugo stablo“*

Prije svega je potrebno potvrditi je li prvo Merkleovo stablo podstablo drugoga. Zatim je nužno provjeriti da se hash novoga merkleovog stabla poklapa sa starim zajedno sa svim novododanim hashevima novoga stabla. Dokaz dosljednosti je minimalni skup posrednih čvorova oko kojih se moraju izračunati dvije navedene stavke.[7]



Sl. 3.3. *Minimalni skup posrednih čvorova (m,k,l)*

U ovome slučaju se provjera dosljednosti sastoji od sljedećih posrednih čvorova: **k**, **l** i **m** (Sl. 3.3.).

1. S čvorovima **m** i **k** možemo rekreirati staro stablo te time potvrditi da staro stablo postoji te da je nepromijenjeno.
2. Nakon toga koristimo čvorove **l** i **k** da bi kreirali čvor **n**.
3. Zatim koristimo čvor **n** zajedno s čvorom **m** da stvorimo novi hash stabla.

Ako se izračunati korijen hasha poklapa s onim korijenom hasha prezentiranoga nam stabla, tada je stablo dosljedno. [7]

3.2. Provjera podataka

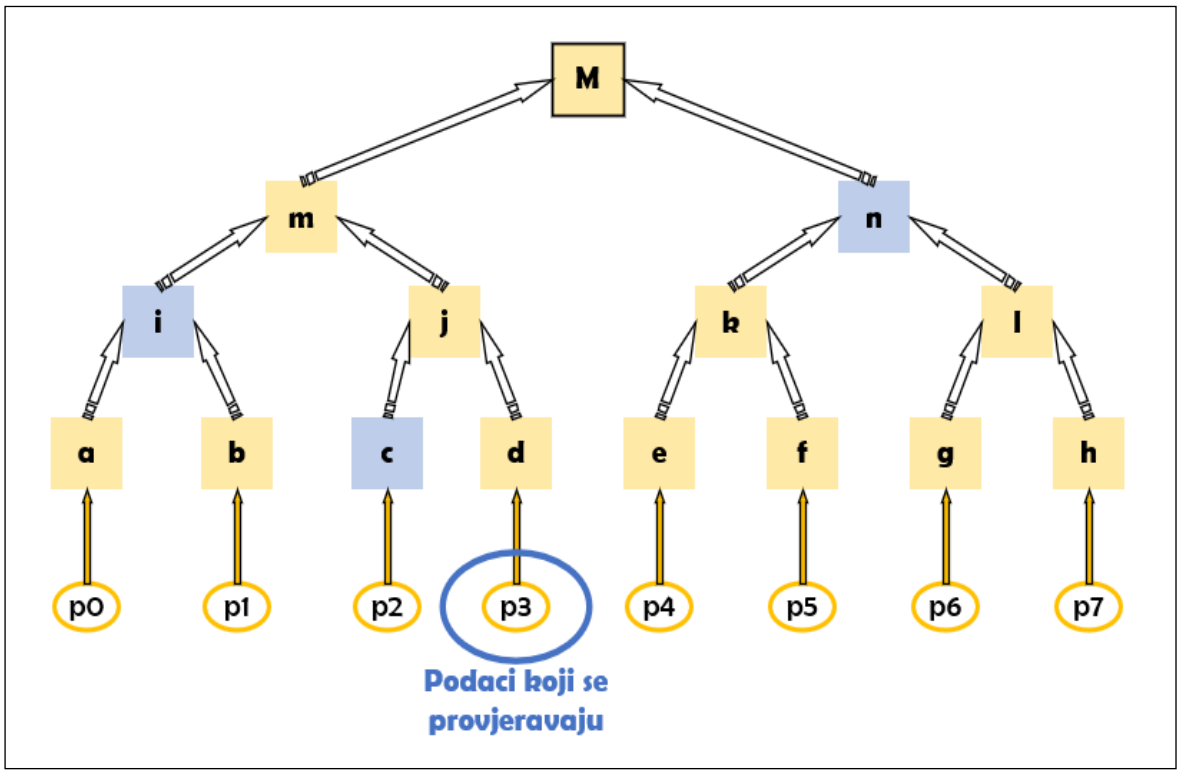
Pomoću provjere podataka je moguće dokazati je li se zaista određeni podatak priključio stablu. Provjera podataka je u mnogim distribuiranim i peer-to-peer sustavima jako važna. Razlog tome je postojanje istih podataka na više lokacija. Stoga, ako se dio podataka promjeni na jednoj lokaciji, bitno je da se taj dio podataka promjeni i na svim ostalim lokacijama. Provjera podataka se koristi da bi potvrdili kako su podatci svuda jednaki. Međutim, provjera svake datoteke u cijelosti iziskuje mnogo vremena i računalne snage.[8] Želimo što više ograničiti količinu podataka koje moramo slati mrežom. Iz tog razloga se koristi Merkleovo stablo.

Umjesto slanja cijelih datoteka preko mreže, šaljemo samo hasheve navedenih datoteka da bismo potvrdili podudaranje.

Protokol se odvija ovako: [8]

1. Računalo **A** šalje hash datoteke računalu **B**
2. Računalo **B** uspoređuje hash s korijenom Merkleovog stabla
3. U slučaju da nema razlike, provjera je završena. U suprotnom nastavljamo dalje
4. Računalo **B** šalje zahtjev za hash korijena dva podstabla navedenog hasha
5. Računalo **A** šalje potrebne hasheve računalu **B**
6. Ponavljamo korak **4.** i korak **5.** dok ne pronađemo nedosljedne pakete podataka

Da bi razumjeli kako provjera funkcionira, pretpostavimo da želimo potvrditi da je podatak **p3** priključen stablu na slici 3.4. Za provjeru određenog podatka potrebno je provjeriti hasheve svih čvorova oko njega. Ako se hash korijena iz našeg proračuna poklapa s prezentiranim stablom, dokazali smo da je navedeni podatak dio stabla.



Sl. 3.4. *Provjera podataka stabla*

U ovome slučaju, za provjeru podataka **p3** su potrebni samo hashevi čvorova **c**, **i**, **n**.

1.Nužan nam je čvor **c** jer zajedno s čvorom **d** tvori čvor **j.**

Hash(j) = Hash(c) + Hash(d)

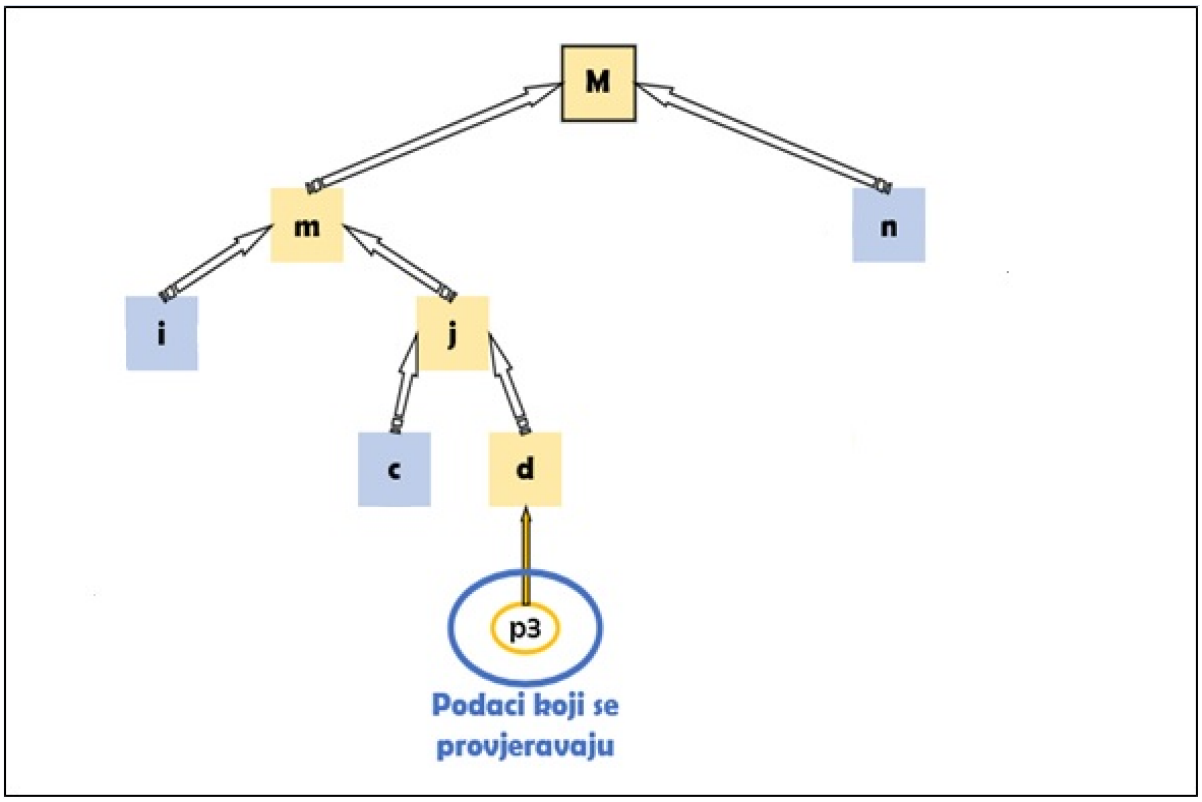
2.Nakon čega u kombinaciji s čvorom **i** proizlazi čvor **m**.

Hash(m) = Hash(i) + Hash(j)

3. Na kraju ostaje čvor **m** i čvor **n** s čijim se hashiranjem dobiva Merkleov korijen.

Hash(M) = Hash(m) + Hash(n)

Ako u procesu provjere podatka nije dobiven korijen hasha koji je jednak referenciranom stablu, provjeravani podatak nije član stabla. [7]



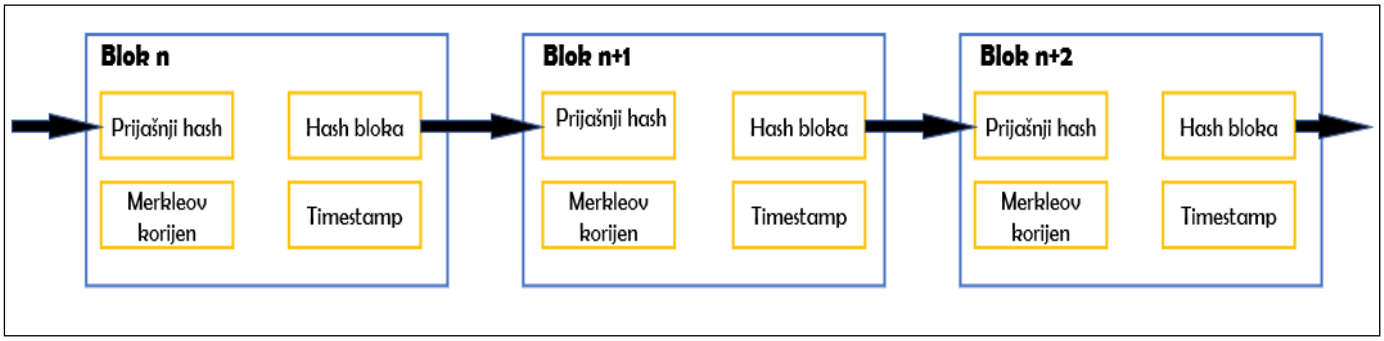
Sl. 3.5. *Članovi potrebni za rekreiranje merkleovog korijena za slučaj gdje se provjerava p3*

Da biste potvrdili provjeru, otkriva vam se vrlo malo podataka o stablu. Nadalje, paket podataka koji je potreban za ovaj dokaz je vrlo mali, što ga čini efikasnim za provjeru i sljanje preko mreže. Ovi primjeri slikovito opisuju kako krajnjem korisniku, koji želi provjeriti točnost i integritet svojih primljenih podataka, nije potrebno slati cijelo stablo za provjeru već samo ključne posredne čvorove pomoću kojih može rekreirati Merkleov korijen i samim time potvrditi integritet podataka.

4. BLOCKCHAIN

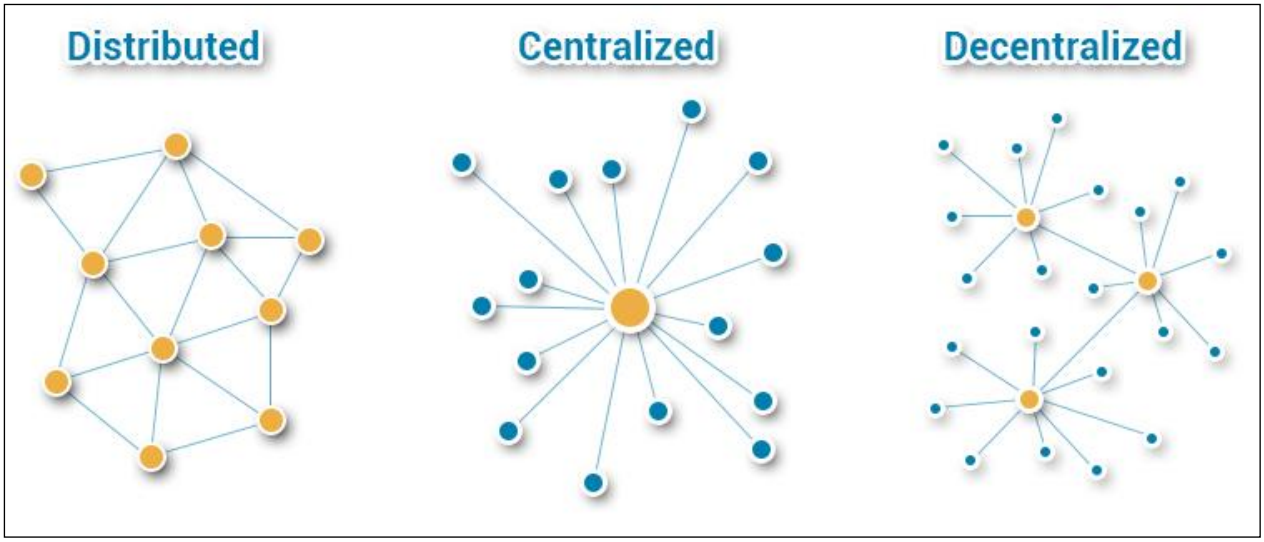
Blockchain tehnologija se predstavlja kao *distributed ledger* što u prijevodu predstavlja knjigu transakcija. Također, blockchain možemo pojmiti kao svojevrsnu bazu podataka za spremanje podataka, dokumenata i informacija.[9] Blockchain je najjednostavnije opisati kao lanac blokova koji sadrže informacije. Razvio ga je, 2009. godine, anonimna skupina/pojedinac po imenu „Satoshi Nakamoto“.

Svaki blok u nizu sastoji se od podatka, vremenske oznake, hasha bloka i hasha prethodnog bloka. Blokovi sadrže mnoštvo transakcija koje su hashirane unutar Merkleovog stabla. Svaki blok sadrži i kriptografski hash prijašnjeg bloka u nizu stvarajući vezu i tako formirajući „lanac“ s ostalim blokovima. Ovakav proces potvrđuje prethodni blok svakog pojedinog bloka sve do onog izvornog. [10]



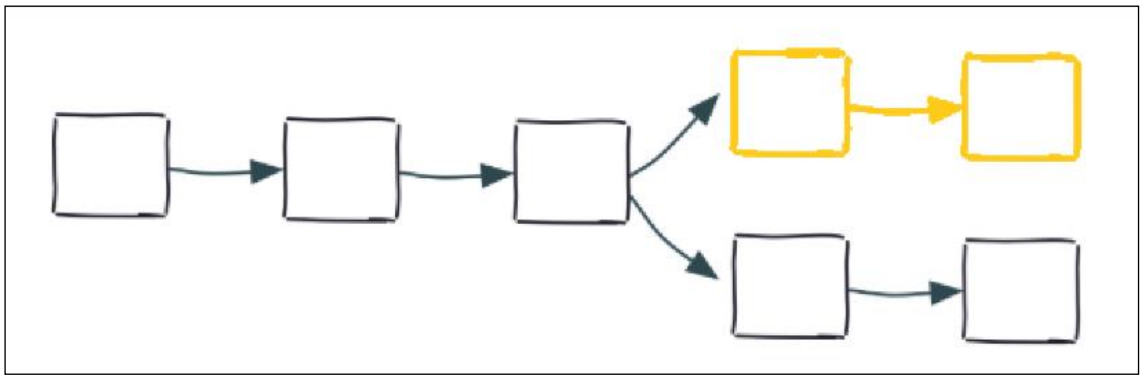
Sl. 4.1. *Struktura blockchaina*

Također, novo dodane podatke, koje se kronološki dodaje na lanac, nije moguće naknadno mijenjati. Mreža blockchaina sama vrši provjere da bi potvrdili jednakost svih kopija blockchaina. Vrlo bitna značajka blockchaina je i decentraliziranost, što znači da se podaci pohranjuju preko *peer-to-peer* mreže. Svaki čvor u decentraliziranom sustavu sadrži kopiju blockchaina. Na taj se način uklanjaju brojne opasnosti koje dolaze s centraliziranom pohranom. [11]



Sl. 4.2. *Usporedba organizacijskih struktura* [12]

Svi korisnici blockchain tehnologije rade na konstantnom proširenju mreže. Rješavajući kriptografske zagonetke ostvaruju pravo da formiraju novi blok transakcija koji će se nadodati na mrežu. Postoji mogućnost, iako vrlo mala, da 2 korisnika blockchaina riješe zagonetku u istome trenutku. Tada će se na tom mjestu na blockchainu stvoriti grananje.



Sl. 4.3. *Grananje lanca* [13]

Grananje na lancu stvara problem jer sada postoje dva lanca. U trenutku stvaranja grananja, dio korisničke populacije će smatrati da se nizanje blokova treba nastaviti u smjeru jednog lanca dok će se ostatak korisnika ipak odlučiti za drugi smjer. Upravo za ovakav slučaj postoji konsenzus zajednice koji tvrdi da se treba nadograđivati onaj lanac koji je u tom trenutku duži. Pa se tako nakon izvjesnog vremena grananje izgubi jer je vrlo mala vjerojatnost da će 2 lanca biti dugo vremena jednake duljine. Oni blokovi koji su bili dio kraćeg lanca gube legitimnost te su potpuno ignorirani.

Rudarenje (engl. Mining) je jedan od pojmova koji se puno pojavljuje u razgovoru o tehnologiji blockchaina.

Pojam rudarenja, u ovom kontekstu, predstavlja proces u koji ulažemo rad , u obliku računalne snage (električne energije), za stvaranje blokova koje će ostatak mreže blockchaina prihvatiti. Osobe koje se time bave su ekvivalent sustava za obradu kreditnih kartica. Primaju transakcije na čekanju, potvrđuju da su kriptografski točne i formiraju ih u blokove koje će se pohraniti u blockchainu. Kako mreža s vremenom sve više raste tako se i zahtjevnost rudarenja prilagođava. Zahtjevnost je konsenzus koji definira koliko rada je potrebno da bi se kreirao važeći blok.

4.1. Konsenzus

Konsenzus blockchaina je realiziran kao skup pravila po kojima svi igraju. Na primjer, jedno od pravila konsenzusa Bitcoina glasi da se vrijednost transakcija prepolavlja svakih 200,000 blokova. Prvih 200,000 blokova su se nagrađivali s 50 BTC(oznaka za kriptovalutu Bitcoin) dok se već nakon 400,000 bloka nagrada smanjila na 25 BTC. Kada bi netko tada zanemario pravilo konsenzusa i kreirao blok za 50 BTC, ostatak mreže bi taj blok proglasio nevažećim jer se uopće ne uklapa u konsenzus. Jedna od najvećih problema unutar blockchain, kojeg svaki ozbiljan konsenzus treba adresirati, je napad većine. Napad većine je izraz kojim se opisuje situacija u kojoj jedan pojedinac ili udruga ima 51% zastupljenosti u mreži. U slučaju *Proof-of-Work* konsenzusa to bi značilo da pojedinac ili skupina ima 51% sveukupne računalne snage unutar zajednice. To omogućava nesmetano generiranje blokova i bogaćenje pojedinca. Iako se takav slučaj još nije dogodio, opasnost postoji.

Trenutno najzastupljeniji konsenzus je Bitcoinov *Proof-of-Work* što ne znači da se to uskoro neće promijeniti. Svakim danom se razvijaju novi konsenzusi za nove aplikacije blockchaina, trenutno se broje u desetcima, a u nastavku su opisani oni najzastupljeniji.

4.1.1. Dokaz rada

Dokaz rada (engl. *Proof of Work*) je trenutno najkorištenije metoda konsenzusa. Rudari trebaju riješiti kriptografsku zagonetku metodom pokušaja i pogrešaka. Ne postoji nikakav drugi način, nema formule po kojoj se zagonetka rješava, jedino ju je moguće riješiti metodom pokušaja i pogrešaka Tko prvi riješi dobiva nagradu i dobiva privilegiju dodavanja bloka u lanac. Ovakva metoda iziskuje puno računalne snage i električne energije. Mnoge kriptovalute, uključujući i Bitcoin, koriste ovakav mehanizam. [14]

4.1.2. Dokaz uloga

Dokaz uloga (engl. *Proof of Stake*) je mehanizam koji ne uključuje rudarenje i trošenje ogromnih količina električne energije. Velike blockchain mreže ovakav pristup još uvijek razmatraju. Naime, sudionici u ovom konsenzusu dobivaju mogućnost da odlože i zaključaju svoje tokene na određeno vrijeme te zauzvrat dobivaju priliku, proporcionalno svome ulogu, da dodaju sljedeći blok. Problem u ovom sustavu je taj da će sudionici koji već imaju puno tokena dobiti priliku da imaju još više. Ovakav pristup je kontroverzan jer se kosi s temeljnim načelima blockchaina. Postoji strah da će se upotrebom ovakvog konsenzusa blockchain centralizirati. Da bi se takav scenarij izbjegao ljudi su počeli razvijati hibride konsenzusa PoW i PoS. [15] Jedan takav se naziva *Proof of Activity* te se koristi u sustavu kriptovalute Decred.

4.1.3. Delegirani dokaz uloga

DpoS je konsenzus unutar kojeg se odabire fiksni broj delegata. Zajednica izabire delegate i tada ti delegati imaju pravo stvarati blokove u zamjenu za kriptovalute. Svi korisnici mreže imaju pravo glasati, no svaki glas nije jednako bitan. Utjecajniji korisnici, oni s više kriptovaluta, imaju veću važnost glasa. Sve radnje delegata su javne, što znači da će svako sumnjivo ponašanje biti vidljivo.[16] Ovaj konsenzus je ekološki prihvatljiv jer nema pretjeranog korištenja električne energije, ali se javljaju drugi problemi poput olakšane okolnosti za stvaranje napada većine i scenarija kada udruženi delegati manipuliraju mrežom.

4.2. Bitcoin

Bitcoin (₿) je oblik elektroničke valute, odnosno kriptovalute. Bitcoini se dobivaju kao nagrada za određivanje procesa poznatog kao rudarenje.

Bitcoinov blockchain sadrži potpunu povijest svih transakcija Svi čvorovi unutar bitcoina su računala koja izvode *Bitcoin Core software* te čuvaju lokalnu kopiju cijelog blockchaina koja se konstantno nadograđuje dodavanjem novih transakcija. Svaka nova transakcija se hashira više puta stvarajući strukturu Merkleovog stabla. Kako bi potvrdili novi blok transakcija, čvorovi će provjeriti hash sadržan u zaglavlju bloka. Blok će biti dodan kronološki na začelje lanca ako zaglavlje sadrži referencu na prijašnji blok. [17]

Korištenje Merleovog stabla je temeljni princip rada Bitcoina, odnosno blockchaina. *Hashing* algoritmi su iznimno otporni na „hakiranje“. Sve transakcije unutar zasebnih blokova su zaštićene stablom hasheva i uz to je svaki blok vezan hashem prijašnjeg. Osoba s namjerom

mijenjana podataka bi trebala probiti zaštitu hasheva pojedinog bloka te nakon toga promijeniti podatke u svim ostalim blokovima blockchaina. S današnjom tehnologijom, takav scenarij je nemoguć. [17]

Unutar bitcoina, nabava je strogo kontrolirana algoritmom koji limitira broj bitcoina u opticaju na 21 milijuna. To ga u teoriji čini privlačnijom valutom jer ako ponuda i potražnja ostaje ista, vrijednost će se povećati. Najmanja jedinici bitcoina naziva se *Satoshi*, ona predstavlja sto milijunti dio bitcoina(0.00000001 ₿). Tako mala novčana jedinica bi mogla omogućiti dodatne mikrotransakcije koje tradicionalni novac ne može popratiti.

Sl. 4.4. *Simbol Bitcoina* [18]

4.3. Ethereum

Poput Bitcoina, Ethereum je distribuirana javna blockchain mreža. Iako postoje značajne tehničke razlike između njih, najviše se razlikuju u svrsi i sposobnosti. Bitcoin pruža jednu aplikaciju blockchain tehnologije, *peer-to-peer* sustav elektroničnog novca. Ethereum se usredotočuje na pokretanje programskog koda bilo koje decentralizirane namjene.

U sustavu Ethereuma, rudari rade kako bi zaradili Ether, tip kripto tokena koji hrani mrežu. Osim što je Ether prodajna kriptovaluta, također se koristi za plaćanje usluga na Ethereum mreži. Uz Ether, postoji i drugi tip tokena. On se naziva Gas te je potreban je za provedbu svakog „pametnog ugovora“. [19]

Pametni ugovor je izraz koji opisuje programski kod koji se može razmijeniti za novac, usluge, sadržaj, imovinu, dionice... Unutar blockchaina, pametni ugovor se ponaša kao svojevrsni računalni program koji se automatski izvršava ako su zadovoljeni određeni uvjeti.

Ethereum koristi Merkleova stabla iz istih razloga iz kojih ih koristi i Bitcoin, zbog jednostavne i iznimno brze verifikacije podataka. Koristi ih za potvrdu, provjeru i opis transakcija. Merkleovo stablo je u tome iznimno učinkovito, no kada je u pitanju uređivanje već postojećih podataka, Merkleovo stablo ozbiljno zaostaje. Iz tog razloga, Ethereum ide korak dalje, koristeći strukturu podataka zvanu „*Merkle Patricia tree*“ koju primjenjuje za održavanje podataka o stanju računa. Stanje u Ethereum-u je opisano pomoću mapiranih vrijednosti po ključu gdje su ključevi adrese a neke od vrijednosti su pohrana, kod i saldo računa. Spomenute vrijednosti se moraju konstantno ažurirati, pa tako zadužena struktura mora biti u stanju brzo računati nove korijene stabala nakon svakog dodavanja, brisanja i ažuriranja bez potrebe da iznova preračunava cijelo stablo. Za razliku od Merkleovog stabla koje je binarno, svaki čvor Merkle Patricia stabla može imati i do šesnaest podređenih čvorova. [21]

Sl. 4.5. *Simbol Ethereuma* [20]

5. PRIMJER KREIRANJA MERKLEOVOG STABLA

U ovom će poglavlju biti opisano i prikazano stvaranje Merkleovog stabla na praktičnom primjeru napisanom u programskom jeziku Kotlin.[22] Kod je pisan u IntelliJ razvojnom okruženju. [23]

5.1. Kreiranje *hash*-a

```
class Hash {
    lateinit var value: ByteArray
        private set

    companion object {
        fun newInstance(buffer: ByteArray): Hash {
            val hash = Hash()
            hash.computeHash(buffer)
            return hash
        }

        fun create(buffer: String): Hash = newInstance(buffer.toByteArray(StandardCharsets.UTF_8))

        fun create(left: Hash, right: Hash): Hash = newInstance(join(left.value, right.value))

        fun join(a: ByteArray, b: ByteArray): ByteArray {
            val c = ByteArray(a.size + b.size)
            System.arraycopy(a, 0, c, 0, a.size)
            System.arraycopy(b, 0, c, a.size, b.size)
            return c
        }
    }

    private fun computeHash(buffer: ByteArray) {
        try {
            this.value = MessageDigest.getInstance("SHA-256").digest(buffer)
        } catch (e: NoSuchAlgorithmException) {
            e.printStackTrace()
        }
    }
```

Sl. 5.1. *Klasa Hash*

U ovoj klasi nema konstruktora već se sve instanciranje vrši statično. Pri kreiraju svakog hasha, funkcija kao parametar prima ili niz znakova ili dva podređena čvora čija se združena vrijednost hashira po, već spomenutom, „SHA-256“ algoritmu. *Hash* objekt ima samo jedan atribut, a to je njegova vrijednost u obliku niza bajtova.

5.2. Kreiranje čvora

```
class Node {
    lateinit var hash: Hash
        private set
    private var leftNode: Node? = null
    private var rightNode: Node? = null
    private var parent: Node? = null

    constructor(hash: Hash) {
        this.hash = hash
    }

    constructor(left: Node, right: Node?) {
        this.leftNode = left
        this.rightNode = right
        this.leftNode!!.parent = this
        if (this.rightNode != null) this.rightNode!!.parent = this

        this.computeHash()
    }

    private fun computeHash() {
        if (this.rightNode == null) {
            this.hash = this.leftNode!!.hash
        } else {
            this.hash = Hash.newInstance(
                Hash.join(
                    this.leftNode!!.hash.value, this.rightNode!!.hash.value
                )
            )
        }
        if (this.parent != null) {
            this.parent!!.computeHash()
        }
    }
```

Sl. 5.2. *Klasa Node(Čvor)*

Klasa *Node* određena je s *hash* vrijednošću čvora i okolnim čvorovima, 2 podređena i jedan nadređeni. Pri inicijalizaciji postoje dvije opcije, kada konstruktor prima samo hash i opcija kada konstruktor prima lijevi i desni čvor. Prva opcija je za slučaj kada čvor stvaramo direktno iz bloka podataka, odnosno kada se radi o listu stabla. Druga opcija se odnosi na stvaranje čvora iz već postojećih čvorova. U ovome dijelu se javlja i dodatni scenarij, a to je kada na istoj razini stabla postoji neparan broj čvorova. Tada će posljednji čvor „pauzirati“ i pričekati svoju priliku za povezivanjem na idućoj, višoj razini.

5.3. Kreiranje stabla

```
class MerkleTree {
    lateinit var root: Node
        private set
    private val nodes: MutableList<Node>
    private val leaves: MutableList<Node>
    init {
        this.nodes = ArrayList()
        this.leaves = ArrayList()
    }

    fun appendLeaf(node: Node): Node {
        this.nodes.add(node)
        this.leaves.add(node)
        return node
    }

    fun appendLeaves(hashes: Array<Hash>): List<Node> {
        val nodes = ArrayList<Node>()
        for (hash in hashes) {
            nodes.add(this.appendLeaf(hash))
        }
        return nodes
    }
    fun buildTree(nodes: List<Node>) {
        if (nodes.isEmpty()) throw InvalidParameterException("Node list not expected to be empty!")
        if (nodes.size == 1) {
            this.root = nodes[0]
        } else {
            val parents = ArrayList<Node>()
            var i = 0
            while (i < nodes.size) {
                val right = if (i + 1 < nodes.size) nodes[i + 1] else null
                val parent = Node(nodes[i], right)
                parents.add(parent)
                i += 2
            }
            buildTree(parents)
        }
    }
    fun buildTree(): Hash? {
        if (this.leaves.isEmpty()) throw InvalidParameterException("Cannot add to a tree with no leaves!")
        this.buildTree(this.leaves)
        return this.root.hash
    }
```

Sl. 5.3. *Klasa MerkleTree(Merkleovo stablo)*

Predajom liste hasheva ili listova uz pomoć funkcija *appendLeaves* i *buildTree*, objekt klase *MerkleTree* može u potpunosti kreirati Merkleovo stablo. Počevši od prve razine, gdje su hashirani podatci u obliku listova stabla, funkcija prolazi kroz listu čvorova te ih povezuje tako da parovima stvara nadređene čvorove. Isti proces se ponavlja sve dok ne ostane samo jedan čvor kojemu ne možemo pridodati roditeljski čvor. Taj čvor postaje Merkleov korijen stabla.

5.4. Testiranje Merkleovog stabla

Na slici 5.4. je postavljen kod za kreiranje Merkleovog stabla iz pet podatkovnih blokova gdje svaki blok sadrži hashiranu vrijednost jednoga znaka. Slika 5.5. pokazuje ishod provedenoga koda, prikazane su hashirane vrijednosti početnih blokova i hashevi stvorenih čvorova zajedno s posljednjim čvorom, Merkleovim korijenom.

```
fun main() {

    val dataHashA = Hash.create("A")
    val dataHashB = Hash.create("B")
    val dataHashC = Hash.create("C")
    val dataHashD = Hash.create("D")
    val dataHashE = Hash.create("E")

    val merkleTree = MerkleTree()
    merkleTree.appendLeaves(
        arrayOf(
            dataHashA,
            dataHashB,
            dataHashC,
            dataHashD,
            dataHashE
        )
    )

    println("\n ${merkleTree.getLeaves()} \n")
    merkleTree.buildTree()
    println("\n Merkle root hash value is ${merkleTree.root}")
}
```

Sl. 5.4. *Testiranje koda za stvaranja Merkleovog stabla*

```
 VZrq0IJk1XldOQlxjN0Fq9SVcuhP5VWQ7vMaiKCP3/0=

 335w5QIVRPSDS77mSp43if68S+gUcN9inK1t2wMyClw=

 ayPA1fNdGxH5toPwsKYXNV3rESd9ka4JHTmcZVuHlA0=

 PznVw0jlt50G6ELBFObMVxWDu/ROSw6/2hoB7AV0XUM=

 qfUVZr1nBffqatVLud60SfeVWC1lKaDiIge4mBIz7Fg=

New node added with the hash value of Y5VvDOSO3Eig1SjLC11Y5NYlr7FNY8obuZUOtlfWH0A=
New node added with the hash value of mKL7/dvHRtcCIiEB/wj8bwxg6cyejAvHI42+2nQzORw=
New node added with the hash value of qfUVZr1nBffqatVLud60SfeVWC1lKaDiIge4mBIz7Fg=
New node added with the hash value of Gz+qP8xe1QzYWS+AXG+PzpdrhYLHObJqbzYTt/mxNhc=
New node added with the hash value of qfUVZr1nBffqatVLud60SfeVWC1lKaDiIge4mBIz7Fg=
New node added with the hash value of jl7rb3O1xHXbzY6oKIpiB15kRkJUmZ0axZs6CtOY7DU=

 Merkle root hash value is jl7rb3O1xHXbzY6oKIpiB15kRkJUmZ0axZs6CtOY7DU=
```

Sl. 5.5. *Ispis konzole pri stvaranju Merkleovog stabla*

5.5. Testiranje provjere podataka

Svojstvo provjere podataka u *hash* stablu predstavlja mogućnost da se u bilo kojem trenutku može provjeriti autentičnost podataka. Uz pomoć okolnih čvorova možemo efikasno odrediti točnost podataka. Na slici 5.6. je prikazana klasa kojom ćemo opisati čvorove potrebne za našu provjeru određenog seta podataka.

```
class ProofHash(var hash: Hash, var direction: Branch) {

    enum class Branch {
        LEFT,
        RIGHT,
        OLD_ROOT
    }

    override fun toString(): String {
        val hash = this.hash.toString()
        val direction = this.direction.toString()
        return "$hash is $direction Child"
    }
}
```

Sl. 5.6. *Klasa za prikaz provjere podataka*

```
fun auditProof(leafHash: Hash): List<ProofHash> {
    val auditTrail = ArrayList<ProofHash>()

    val leafNode = this.findLeaf(leafHash)

    if (leafNode != null) {
        if (leafNode.getParent() == null) throw InvalidParameterException("Expected leaf to have a parent!")
        val parent = leafNode.getParent()
        this.buildAuditTrail(auditTrail, parent, leafNode)
    }

    return auditTrail
}

 fun findLeaf(hash: Hash): Node? {
    return this.leaves.stream()
        .filter { leaf -> leaf.hash == hash }
        .findFirst()
        .orElse(null)
}

private fun buildAuditTrail(auditTrail: MutableList<ProofHash>, parent: Node?, child: Node?) {
    if (parent != null) {
        if (child!!.getParent() != parent) {
            throw InvalidParameterException("Parent of child is not expected parent!")
        }

        val nextChild = if (parent.getLeftNode() == child) parent.getRightNode() else parent.getLeftNode()
        val direction = if (parent.getLeftNode() == child)
            ProofHash.Branch.RIGHT
        else
            ProofHash.Branch.LEFT

        if (nextChild != null) auditTrail.add(ProofHash(nextChild.hash, direction))

        this.buildAuditTrail(auditTrail, parent.getParent(), child.getParent())
    }
}
```

Sl. 5.7. *Funkcije za izvođenje provjere podataka*

Potrebnu putanju ili set podataka koje moramo imati želimo li izvršiti provjeru podataka možemo dobiti koristeći funkciju *auditProof na* slici 5.7.. Njen zadatak je parsirati stablo, poznavajući samo određeni list istoga stabla, i pronaći posredne čvorove s kojima možemo stvoriti identičnu kopiju stabla.

Funkciji na slici 5.8. predajemo sve potrebne informacije koje su potrebne kako bi utvrdili je li provjeravani set podataka autentičan. Funkcija *vertifyAudit* prima korijen hasha, podatak kojeg provjeravamo i okolne čvorove s kojima bi, ako je podatak autentičan, trebao tvoriti identični hash Merkleovog korijena. Tako stablo vrši provjeru svojih podataka, pokušavajući rekreirati stablo na najefikasniji mogući način.

```
fun verifyAudit(rootHash: Hash, leafHash: Hash, auditTrail: List<ProofHash>): Boolean {
    if (auditTrail.isEmpty()) throw InvalidParameterException("Audit trail cannot be empty!")

    var testHash = leafHash

    for (auditHash in auditTrail) {
        testHash = if (auditHash.direction === ProofHash.Branch.RIGHT)
            Hash.create(testHash, auditHash.hash)
        else
            Hash.create(auditHash.hash, testHash)
    }

    return testHash.equals(rootHash)
}
```

Sl. 5.8. *Funkcija za provjeru podataka*

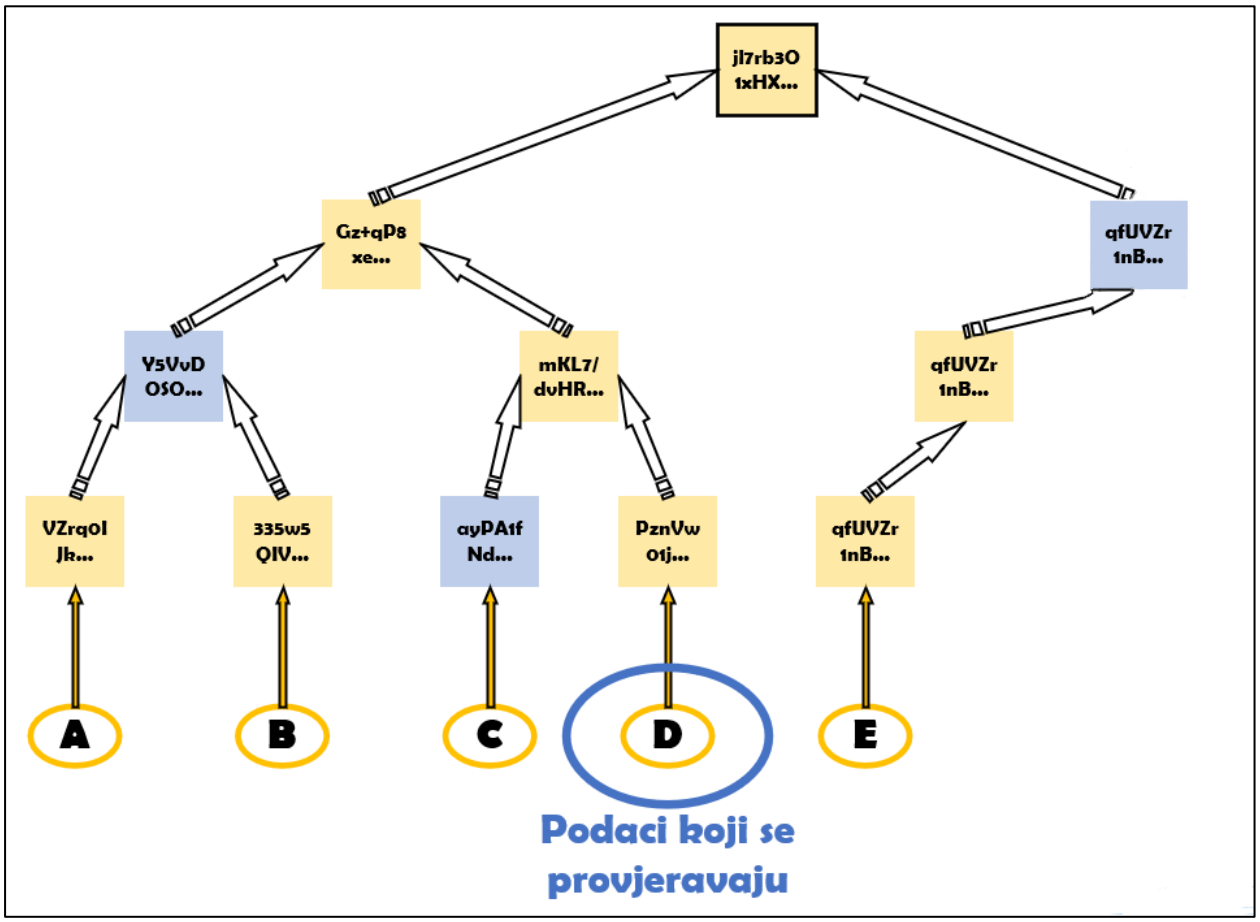
Kao provjeru funkcije za verifikaciju podataka, promatrat ćemo set podataka D iz prijašnjeg stabla. Funkcija *auditProof* će nam ispisati čvorove s kojima možemo rekreirati stablo dok će funkcija *verifyAudit* potvrditi možemo li rekreirati stablo, odnosno posjedujemo li točan i autentičan set podataka.

```
val auditProof = merkleTree.auditProof(dataHashD)
println(auditProof)
println(MerkleTree.verifyAudit(merkleTree.root.hash, dataHashD, auditProof))
```

Sl. 5.9. *Poziv funkcije za provjeru podataka*

```
[ayPA1fNdGxH5toPwsKYXNV3rESd9ka4JHTmcZVuHlA0= is LEFT Child,
 Y5VvDOSO3Eig1SjLC11Y5NYlr7FNY8obuZUOtlfWH0A= is LEFT Child,
 qfUVZr1nBffqatVLud60SfeVWC1lKaDiIge4mBIz7Fg= is RIGHT Child]
true
```

Sl. 5.10. *Ispis konzole nakon provjere podataka*



Sl. 5.11. *Grafički prikaz primjera provjere podataka*

Izvedeni test je potvrdan, što znači da možemo zajedno s provjeravanim čvorom i 3 ključna čvora uspješno rekreirati stablo, to jest da je blok podataka D, član stabla.

6. ZAKLJUČAK

Korisnost Merkleovog stabla u distribuiranim sustavima proizlazi iz toga što je neučinkovito provjeravati cjelokupnu datoteku kako bi provjerili ima li datoteka grešaka. Dok se u nepouzdanim sustavima, poput peer-to-peer sustava, Merkleovo stablo cijeni zbog njegovog svojstva potvrde informacija. Iako je koncept Merkleovog stabla bio poznat i prije, pojavom Bitcoina je došao u centar pažnje. Ova tehnologija ne ide nigdje i tek je u začecima. Trenutno se pretežno koristi u kriptovalutama dok u bliskoj budućnosti možemo očekivati implementaciju blockchaina u mnogim djelatnostima.

Tehnologija blockchaina nam omogućava stvaranje velikih sustava u kojemu nema vladajućeg autoriteta koji upravlja sustavom, već je upravo sustav taj koji regulira i ima autoritet. Primjerice, tehnologiju blockchaina je moguće implementirati u sustav glasanja čime bi izbjegli potencijalno mijenjanje i negiranje glasova, te tako osigurali integritet glasačkog tijela. Ovakav sustav možemo implementirati i u medicini gdje bi se unutar blockchaina skrivale povijest bolesti pacijenata. Tako bi osoba imala uvid na sve promjene u realnom vremenu te bi se spriječilo potencijalno zloupotrebljavanje povjerljivih informacija.

LITERATURA

[1] Robert Manger, Miljenko Marušić, *„Strukture podataka i algoritmi“,* rujan 2003, *http://lnr.irb.hr/soya/nastava/spa-skripta.pdf* , *pristup ostvaren 14.09.2018.*

[2] *„Merkle tree“*, https://en.wikipedia.org/wiki/Merkle_tree , *pristup ostvaren 14.09.2018.*

[3] *„Hashing“,* https://www.techopedia.com/definition/14316/hashing , pristup ostvaren 14.09.2018.

[4] Brian Curran, *„What is a Merkle Tree? Beginner's Guide to this Blockchain Component“*, 9. Srpanj, https://blockonomi.com/merkle-tree/ , *pristup ostvaren 14.09.2018.*

[5] *„SHA256 Hash Generator“*, https://passwordsgenerator.net/sha256-hash-generator/ , *pristup ostvaren 14.09.2018.*

[6]Marc Clifton, *„Understanding Merkle Trees - Why use them, who uses them, and how to use them“*, 3.ožujak2017,

https://www.codeproject.com/Articles/1176140/UnderstandingMerkleTreesWhy-use-them-who-uses-t ,Code Project, *pristup ostvaren 14.09.2018.*

[7] *„Certificate Transparency, How Log Proofs Work“*, http://www.certificate-transparency.org/log-proofs-work , *pristup ostvaren 14.09.2018.*

[8] Alex Chumbley, Karleigh Moore, and Jimin Khim, *„Merkle Tree“*, Brilliant, https://brilliant.org/wiki/merkle-tree/ , *14.09.2018.*

[9] Boris Agatić, *„„Osnove Blockchaina (prvi dio)“,* 8. Kolovoz 2017., Bitcoin radionica, https://bitcoin-radionica.com/osnove-blockchaina/ , *pristup ostvaren 14.09.2018.*

[10] Bhaskar, Nirupama Devi; Chuen, David Lee Kuo, *„Bitcoin Mining Technology“,* 2015.,

[11] Maryanne Murray, *„Blockchain explained“, 15. Lipanj 2018., Reuters graohics,* http://graphics.reuters.com/TECHNOLOGY-BLOCKCHAIN/010070P11GN/index.html , *pristup ostvaren 14.09.2018.*

[12] 482.solutions, „Distributed Ledger Technology and it's types“, 6. Veljača 2018., https://blog.482.solutions/distributed-ledger-technology-and-its-types-ad76565ae76, *pristup ostvaren 14.09.2018.*

[13] Diogo Mónica, „Bitcoin hard-forks and replay attacks“, 25. Rujan 2017, https://diogomonica.com/2017/09/25/what-to-do-before-a-blockchain-hard-fork/, *pristup ostvaren 14.09.2018.*

[14] Divya Singh, *„Types of consensus algorithms“* , 1. Kolovoz 2018., Blockchain semantics, https://www.blockchainsemantics.com/blog/types-consensus-algorithm/ , *pristup ostvaren 14.09.2018.*

[15] Georgios Konstantopoulos, *„Understanding Blockchain Fundamentals, Part 2: Proof of Work & Proof of Stake“*, 8. Prosinac 2017., Medium, https://medium.com/loom-network/understanding-blockchain-fundamentals-part-2-proof-of-work-proof-of-stake-b6ae907c7edb, *pristup ostvaren 14.09.2018.*

[16] Nikola Rogina, *„Što je konsenzus i koje sve vrste postoje“,* 2. Ožujak 2018., Kriptovaluta.hr, https://www.kriptovaluta.hr/tutorials/sto-je-konsenzus-i-koje-sve-vrste-postoje/ , *pristup ostvaren 14.09.2018.*

[17] *“What is a Merkle Tree and How Does It Help Organize Data On The Bitcoin Blockchain?“,* Bitcoin Australia, 7. Kolovoz 2018., https://bitcoin.com.au/what-is-a-merkle-tree-and-how-does-it-help-organize-data-on-the-bitcoin-blockchain/ , *pristup ostvaren 14.09.2018.*

[18] „*Promotional graphics“*, Bitcoin.it, http://vu.hn/images/bitcoin-symbol-and-logo-originsteps/completed%20bitcoin%20logo.png , *pristup ostvaren 14.09.2018.*

[19] *„What is Ethereum? A Step-by-Step Beginners Guide“,* Blockgeeks, 12. Rujan 2018., https://blockgeeks.com/guides/ethereum/ , *pristup ostvaren 14.09.2018.*

[20] „Ethereum“, Bitcoin wiki, https://en.bitcoinwiki.org/wiki/File:Ethereum11.png , *pristup ostvaren 14.09.2018.*

[21] Vitalik Buterin, „Merkling in Ethereum“, Etherium Foundation Blog, https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/, *pristup ostvaren 15.09.2019.*

[22] Kotlin [online], JetBrains, dostupno na: https://kotlinlang.org/, *pristup ostvaren 15.09.2019.*

[23] IntelliJ, JetBrains, https://www.jetbrains.com/idea/, *pristup ostvaren 15.09.2019.*

SAŽETAK

U radu je opisan način rada kriptografske strukture podataka Merkleovog stabla te se teorijski obrađuju pojmovi poput hashiranja, pojedinih svojstava Merkleovog stabla i blockchaina. Također se otkriva uloga hash funkcija u samom radu blockchaina, odnosno Merkleovog stabla gdje se algoritmima poput provjere podataka i provjere dosljednosti stvara sustav utvrđivanja integriteta podataka. A to je točno što je tehnologiji blockchaina trebalo da se razvije u najpopularniji decentralizirani sustav. Blockchain je otporan na zlonamjerno mijenjanje i brisanje podataka jer je svaka njegova transakcija više puta hashirana s ostalim transakcijama i blokovima međusobno. Bitcoin, koji je i započeo revoluciju kriptovaluta, nije jedini koji koristi ovu tehnologiju. Ethereum i mnoge druge kriptovalute koriste sličan koncept s malim razlikama u setu pravila ponašanja u mreži.

Ključne riječi: Bitcoin, blockchain, hash funkcija, kriptografija, Merkleovo stablo

ABSTRACT

Title: Merkle tree

This paper describes how the cryptographic structure of Merkle tree data works, and theoretically addresses concepts such as hashing, individual properties of the Merkle tree, and blockchain. It also reveals the role of hash functions in the blockchain itself, that is, the Merkle tree, where algorithms such as audit proof and consistency proof create a system for determining data integrity. And that is exactly what blockchain technology needed to become the most popular decentralized system. The blockchain is tamper proof because each of its transactions has been hashed multiple times with other transactions and blocks to each other. Bitcoin, which started all of this, is not the only one using this technology. Ethereum and many other cryptocurrencies use a similar concept with small differences in the behavioral rules of the network.

Keywords: Bitcoin, blockchain, cryptography, hash function, Merkle tree

ŽIVOTOPIS

Tomislav Stipanić je rođen 15. prosinca 1996. u Osijeku. Osnovnu školu završava u Bilju 2011. godine i upisuje Elektrotehničku i prometnu školu u Osijeku, smjer Elektrotehničar. 2015. godine Upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

Potpis:

_____

PRILOG

Na optičkom disku priloženom uz ovaj završni rad nalazi se digitalna verzija rada u .docx i .pdf formatu. Također, nalazi se i izvorni kod primjera kreiranja Merkleovog stabla.