Универзитет у Београду
Математички факултет
Београд, 2020.

Криптографски алгоритми у систему Биткоин

Аутор: Давид Ивић
Ментор: др Миодраг В. Живковић

Чланови комисије:
др Филип Марић
др Саша Малков

Наслов мастер рада: Криптографски алгоритми у систему Биткоин

Резиме: У раду су описани делови система Биткоин намењеног преносу новца од једне особе ка другој без посредства банке. Појашњени су шифарски алгоритми који се користе у његовој основи, алгоритми криптографских хеш функција и структура трансакције којима се новац преноси. Како је цео систем везан за новац, посебна пажња посвећена је истраживању његовог настанка, структуре као и сигурности. Део рада су и потенцијални напади који могу да наруше функционисање мреже Биткоин. Разматрани су различити начини обављања преноса новца кроз систем као и његове предности и мане. Појашњено је како особа може да буде активни учесник система и допринесе наставку његовог развијања уз новчану надокнаду. Последње поглавље рада садржи анализу кôда имплементације Биткоин протокола, експерименталну модификацију кôда као и експеримент спроведен над новчаником.

Кључне речи: трансакције, биткоин, хеш вредност, ланац блокова, новчаник

Садржај:

1 Увод

Криптовалуте као новац постоје само у дигиталном облику. Давид Чом (David Chaum) [4] је први представио концепт криптовалута, објашњавајући систем анонимних трансакција. Криптовалуте су постале популарне тек након појаве валуте под називом биткоин.

Биткоин је настао 2008. године као пројекат отвореног кôда који је покренуо човек или група под псеудонимом (његов/њихов индентитет још увек није познат) Сатоши Накамото (Satoshi Nakamoto). Управо тада представљена је технологија ланаца блокова (енг. blockchain) кроз Накамотов рад *The White Paper* [11]. Систем ланаца блокова је основа већине других криптовалута данашњице (називају се још и алткоини, јер су алтернатива биткоину). Биткоин је криптовалута која је câма по себи децентрализована, односно не захтева трећу особу или ентитет (нпр. банку) који ће да надгледа трансакцију која се обавља између две особе. Децентрализација и целокупно постојање система само кроз интернет, разликује се од традиционалног система размене новца. Тренутно постоји преко 4000 активних криптовалута, а од њих је најпознатија, односно има највећу укупну вредност, управо биткоин са преко 18 милиона активних новчића - биткоина, са ознаком валуте BTC и са вредношћу преко 9 900$ за један BTC (јул 2020. године). У систему Биткоин, 1BTC састоји се од 100 000 000 сатошија (енг. satoshi). Следећа по важности је валута под називом итиријум (енг. Ethereum), где 1ETH $\approx$ 300$.

Биткоин има изгледа да постане замена за новац (на начин на који га данас познајемо) јер обезбеђује велики ниво заштите. Све је већи број компанија које имају подршку за плаћање производа користећи биткоине (Microsoft, BMW, KFC итд.). Биткоин има велики број предности у односу на централизован систем плаћања, као што су: анонимност, смањени трошкови трансакције, већи ниво заштите новца итд. Оно што тренутно спречава Биткоин да у потпуности избаци употребу класичног новца (поред политичких разлога) је време обраде трансакције. Обрада једне трансакције између два чвора траје просечно 10 минута, али то може да траје и до 24 сата, што практично искључује ову валуту за свакодневне потребе плаћања. Додатна карактеристика система Биткоин (која се може третирати као додатни проблем) јесте нестабилност биткоина као криптовалуте. Највећи икада забележан скок вредности једног биткоина десио се средином децембра 2017. године, када је 1BTC вредео приближно 19 000$, али се већ крајем јануара вредност биткоина троструко смањила.

Рад је организован тако да глава 2 садржи техничку позадину система Биткоин. Представљени су шифарски системи потребни за функционисање мреже као и начин аутентикације корисника. Глава 3 садржи детаље структуре једне трансакције, која се обавља између два корисника, као и типове различитих формата трансакције. Технологија ланаца блокова представљена је у глави 4. У оквиру исте главе, обрађени су различити типови напада на које систем треба да буде отпоран. Глава 5 садржи преглед различитих приступа чувању најбитнијих корисничких информација (попут приватног кључа). Начин на који корисник може да учествује на проширивању ланца блокова дат је у глави 6. Глава 7 даје преглед имплементације појединих делова протокола система као и његову практичну примену.

2 Биткоин - криптографски елементи

У овом поглављу биће представљена криптографска и техничка решења која се користе за функционисање мреже Биткоин.

2.1 Симетрични шифарски системи

Симетрични шифарски системи представљају један од најстаријих криптографских алата који се користи у комуникацији која треба да буде заштићена. Циљ оваквих система је да шифрују поруку приватним кључем који је претходно размењен између особа које комуницирају, тако да се коришћењем само тог кључа може и дешифровати.

Пример једног симетричног система под називом *Цезарова шифра*, заснива се на идеји да порука буде шифрована тако да се свако њено слово замени са n-тим следећим унутар језичког писма на коме је порука написана. У овом случају улогу приватног кључа игра број n, јер се порука дешифрује враћањем сваког слова шифроване поруке уназад у језичком писму за n места. Назив је потекао одатле што је овакав систем користио Јулије Цезар, у комуникацији са својим генералима, са вредношћу приватног кључа $n = 3$.
Постоји неколико проблема са шифратима који се заснивају на овој идеји:

- број различитих приватних кључева је ограничен. На пример за поруку на енглеском језику постоји само 26 различитих приватних кључева;
- ако би се тај систем користио у енглеском језику, фреквенције предлога THE, A, AN су велике, па може лако да се дође до закључка о вредности приватног кључа (дешифрујући слова предлога). Комплексност шифрата може да се повећа уклањањем белина између речи унутар поруке;
- вредност кључа између особа које комуницирају мора да буде размењена неким сигурним каналом пре него што та шифрована комуникација уопште започне.

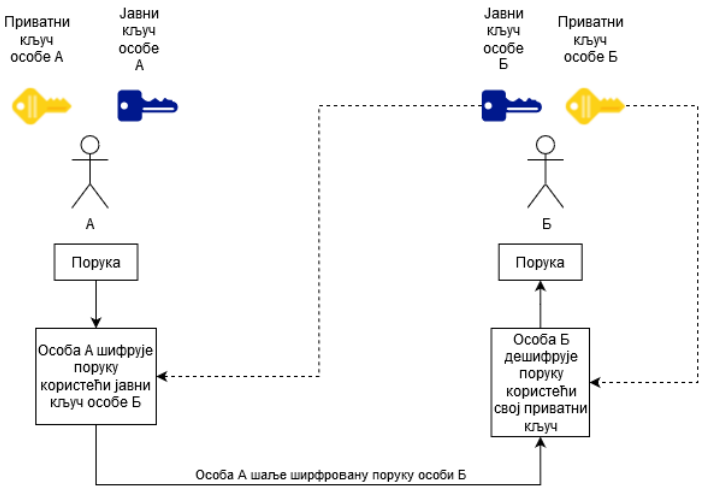
2.2 Системи са јавним кључем

Систем са јавним кључем (асиметрични систем) даје одговор на питање које се намеће код коришћења симетричних шифрата: Шта ако не постоји сигуран канал за размену приватног кључа? С обзиром на то да се данашња комуникација обавља кроз интернет који је несигуран канал, одговор на ово питање добија више на значају. У систему се користе два кључа, приватни и јавни, који су математички повезани (прво се генерише приватни кључ, а на основу њега јавни), видети на пример књигу [7]. Особа у комуникацији, која је заштићена системом са јавним кључем, у поседу има свој приватни и јавни кључ. **Приватни кључ** је сакривен и тај кључ би требало да се чува у тајности и да само особа, која је власник кључа, може да му приступи. **Јавни кључ** се објављује и све особе у систему комуникације знају за њега.
Нека је функција f таква да:

за дато x, израчунавања $y = f(x)$ је лако изводљиво,
за дато y, време за израчунавање $x = f^{-1}(y)$ је непрактично велико.

У систему шифровања јавним кључем, шифарска трансформација је $f(x)$, јавни кључ је параметар од кога функција зависи и свима је познат, оригинална порука је x, док је шифрована порука y. Тајни кључ је параметар од кога зависи инверзна функција. Тајни кључ је одређен јавним кључем али његово ефективно израчунавање полазећи од јавног кључа тражи извршавање неприхватљиво великог броја операција.

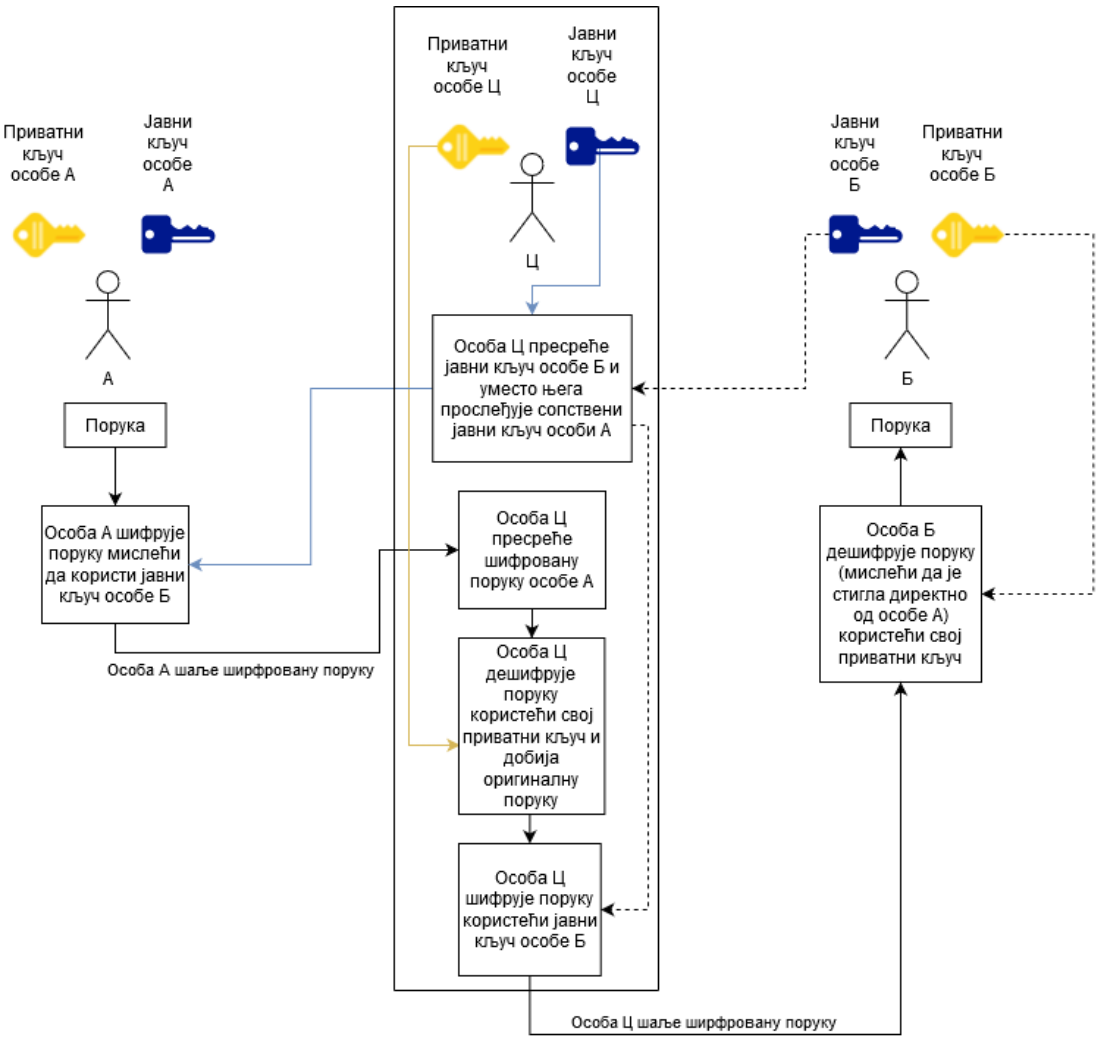
Пример 2.2.1. Особа А жели да пошаље шифровану поруку кроз несигурни комуникациони канал особи Б, која претходно треба да пошаље свој јавни кључ особи А (то је у реду, јер јавни кључ и треба да буде доступан свима у мрежи). Особа А поруку, која је намењена особи Б, шифрује функцијом f користећи јавни кључ који је претходно добила од особе Б. Након пријема поруке, особа Б користи свој приватни кључ и дешифрује примљену поруку. Чињеница да само особа Б зна вредност свог приватног кључа, спречава и особу Ц, која пресреће поруку (након примењене функције f над поруком) да поруку дешифрује и прочита. Приказ овакве комуникације дат је на слици 2.1.



Слика 2.1: Комуникација у шифарском систему са јавним кључем

Овакав тип комуникације подложан је нападу под називом „напад са човеком у средини” (енг. Man-in-the-Middle attack), који користи чињеницу да се јавни кључ прво шаље кроз потенцијално несигурни комуникациони канал. Опис овог напада илустрован је примером 2.2.2.

Пример 2.2.2. Особа Б шаље свој јавни кључ особи А и кључ пресреће особа Ц, која контролише канал. Особа Ц генерише сопствени пар кључева (приватни и јавни) и прослеђује свој јавни кључ особи А, док јавни кључ особе Б задржава за себе. Особа А шифрује поруку јавним кључем који је добила, мислећи да га шифрује јавним кључем особе Б. Након пресретања шифроване поруке, особа Ц може лако поруку да дешифрује користећи свој приватни кључ. Након што има оригиналну поруку (или након што је оригинална порука измењена), особа Ц шифрује поруку користећи претходно пресретнут јавни кључ особе Б и прослеђује је особи Б. Особа Б дешифрује поруку мислећи да је она дошла директно од особе А. Ни особа А, ни Б нису свесни напада. Овај пример је и показатељ да асиметрични систем не решава директно проблем успоставе комуникације без сигурног канала. Приказ напада дат је на слици 2.2.



Слика 2.2: Напад са човеком у средини

Описани проблем може се решити на више начина:

- могуће је слање јавног кључа кроз проверен и сигуран комуникациони канал;
- метод који се заснива на кругу људи који међусобно могу да гарантују један другом да су то заиста њихови јавни кључеви, назива се „мрежа поверења” (енг. web of trust). Сваки корисник има круг особа за чије јавне кључеве је сигуран да њима припадају (упознали су се лично, на пример). Уколико особа жели да комуницира са неким ван свог круга, једноставно пита за јавни кључ те особе кориснике из свог круга повереника, а онда они проверавају свој круг, све док се не дође до потврде о јавном кључу те особе;
- проширење метода мреже поверења је да постоји сертификационо тело које повезује особе и њихове јавне кључеве (више речи о овоме биће у оквиру тачке 2.4).

2.3 Криптографске хеш функције

Криптографске хеш функције су функције које за улаз добијају податке произвољне дужине, а на излазу производе број; тај број се може представити стрингом знакова из неке азбуке фиксне дужине и назива се хеш вредност. Од криптографске хеш функције H, захтева се да:

- вредност функције буде лако израчунљива,
- обезбеди једносмерност, односно да за дато y буде тешко одредити x, тако да важи $\mathrm{H}(x) = y$,
- обезбеди *основну отпорност на колизију*, односно да за дато x буде тешко одредити x', тако да важи $\mathrm{H}(x) = \mathrm{H}(x')$.

Вероватноћа појављивања колизије између n насумично одабраних порука зависи од дужине излаза функције. Нека је број различитих излаза функције означен са K; ако је излаз дужине 128 битова, онда је $K = 2^{128}$. Нека је n број већ хешираних вредности. Вероватноћа да две израчунате хеш вредности буду једнаке је:

$$p = 1 - e^{n^2/(2K)} \sim \frac{n^2}{2K} \text{ (за мале вредности овог израза).}$$

Израз одговара изразу за вероватноћу унутар проблема под називом „рођендански парадокс"[1].

2.3.1 SHA256

Криптографска хеш функција SHA256 је из фамилије функција SHA-2 (енг. Secure Hash Algorithm 2)[2] развијена од стране Агенције за националну безбедност (енг. National security agency, NSA), а објављена од стране института за стандарде и технологију (енг. National Institute of Standards and Technology).

Операције

Операције се примењују на 32-битне речи и то су:

- логичке операције; конјукција - '$\wedge$', дисјункција - '$\vee$', ексклузивно или - '$\oplus$', негација - '$\overline{\ }$'. Како се операције примењују на 32-битне речи, операције се извршавају 32 пута са одговарајућим битовима два аргумента,
- сабирање по модулу 2^{32},
- померање удесно, означено је са $ShR(A, n)$ - помера битове речи A у десно за n позиција,
- ротација удесно означена је са $RotR(A, n)$ - ротира битове речи A у десно за n позиција. За разлику од шифтовања, након што битови пређу на позицију већу од дужине речи, бит се не губи већ се смешта на нулти индекс позиције битовске речи,
- конкатенација; у ознаци A||B - надовезује две речи.

Функције

Комбинацијом претходно дефинисаних операција и речи X, Y, Z, дефинишемо функције:

- $\mathrm{Ch}(X, Y, Z) = (\mathrm{X} \wedge \mathrm{Y}) \oplus (\overline{X} \wedge \mathrm{Z})$ - скраћено од речи *choise* (назив је потекао од особине функције којом се помоћу бита речи X *одлучује* да ли се узима бит из речи Y или речи Z),

[1] http://poincare.matf.bg.ac.rs/%7Eezivkovm//nastava/kripto.pdf
[2] https://en.bitcoinwiki.org/wiki/SHA-256

- Maj(X, Y, Z) = (X $\wedge$ Y) $\oplus$ (X $\wedge$ Z) $\oplus$ (Y $\wedge$ Z) - скраћено од речи *majority* (назив је потекао од особине функције која за резултат има бит који се највише јавља у скупу три улазна бита),
- $\Sigma_0(X)$ = RotR(X, 2) $\oplus$ RotR(X, 13) $\oplus$ RotR(X, 22),
- $\Sigma_1(X)$ = RotR(X, 6) $\oplus$ RotR(X, 11) $\oplus$ RotR(X, 25),
- $\sigma_0(X)$ = RotR(X, 7) $\oplus$ RotR(X, 18) $\oplus$ ShR(X, 3),
- $\sigma_1(X)$ = RotR(X, 17) $\oplus$ RotR(X, 19) $\oplus$ ShR(X, 10).

Функције Σ i σ су линеарне за разлику од функција Ch и Maj.

Константе

Криптографска хеш функција SHA256 користи скуп константи у оквиру својих израчунавања. Константе су добијене као део иза децималног зареза трећег корена прва 64 проста броја и ознака им је: $K_1, K_2, ..., K_{64}$.

Проширивање поруке

Пре самог хеширања, порука се проширује тако да дужина записа буде множилац броја 512. Прво се на поруку надовезује бит 1, који је индикатор краја поруке и почетка надовезивања. Након тога се додаје одговарајући број нула тако да допуњена дужина при дељењу са 512 даје остатак 448. Преостали део од 64 бита попуњава се податком о дужини поруке пре него што је дошло до надовезивања. Битовски запис поруке проширује се чак и када њена оригинална дужина већ одговара множиоцу броја 512. Циљ проширивања јесте подела поруке на једнаке блокове дужине 512.

Реализација

Блокови се даље разбијају на речи у ознаци W_i, дужине 32 бита, $i = 1, 2, \ldots, 16$. На основу ових 16 речи израчунава се додатних 48 речи W_i на основу израза:

$$W_i = \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16}, \text{ где је } i,\ 17 \leqslant i \leqslant 64.$$

У наставку израчунавања користи се осам 32-битних регистара $H_1, H_2, ..., H_8$. Почетни садржај ових регистара (IHV, Initial Hash Value) су делови иза децималног зареза квадратног корена првих 8 простих бројева. Почетне вредности регистара дате су у наставку:

$$H_1^{(0)} = 0x6a09e667\ H_2^{(0)} = 0xbb67ae85\ H_3^{(0)} = 0x3c6ef372\ H_4^{(0)} = 0xa54ff53a$$
$$H_5^{(0)} = 0x510e527f\ H_6^{(0)} = 0xa9b05688c\ H_7^{(0)} = 0x1f83d9ab\ H_8^{(0)} = 0x5be0cd19$$

У наставку се користи осам 32-битних регистара a, b, c, d, e, f, g, h и две привремене речи T_1, T_2 чије се вредности ажурирају у свакој итерацији. На почетку сваке итерације t ($1 \leq t \leq N$, где је N укупан број блокова), вредности регистара a, b, c, d, e, f, g, h, се иницијализују на тренутну вредност регистара IHV. Потом се у оквиру додатних 64 понављања ажурирају регистри a, b, c, d, e, f, g, h и привремене речи T_1, T_2 на следећи начин:

$T1 = h + \Sigma_1(e) + Ch(e, f, g) + K_i + W_i,$
$T2 = \Sigma_0(a) + Maj(a, b, c),$

$h = g$,
$g = f$,
$f = e$,
$e = d + T_1$,
$d = c$,
$c = b$,
$b = a$,
$a = T_1 + T_2$.

Вредности регистара IHV у итерацији t се потом ажурирају на следећи начин:

$H_1^{(t)} = H_1^{(t-1)} + a$
$H_2^{(t)} = H_2^{(t-1)} + b$
$H_3^{(t)} = H_3^{(t-1)} + c$
$H_4^{(t)} = H_4^{(t-1)} + d$
$H_5^{(t)} = H_5^{(t-1)} + e$
$H_6^{(t)} = H_6^{(t-1)} + f$
$H_7^{(t)} = H_7^{(t-1)} + g$
$H_8^{(t)} = H_8^{(t-1)} + h$
Криптографска хеш вредност се на крају рачуна као конкатенација регистара IHV након последње итерације t.

2.3.2 RIPEMD160

Криптографска хеш функција RIPEMD160 је из фамилије RIPEMD (енг. RACE Integrity Primitives Evaluation Message Digest) [1]. Функција на улазу прима поруку коју проширује тако да њена дужина буде умножак броја 512. Проширивање се изводи на исти начин као у случају функције SHA256 (тачка 2.3.1). Проширена порука дели се у блокове дужине 512 битова. Сваки блок даље се дели на 16 речи дужине 32 бита над којима се примењују операције које су груписане као операције леве и десне гране. Унутар и леве и десне гране операције се примењују у оквиру 5 рунди. Свака рунда се састоји од 16 понављања. Излаз је криптографска хеш вредност дужине 160 битова.

Нотација

Нотација којом ће бити описан алгоритам дата је у наставку:

- порука у ознаци $M = (m_0, m_1, ..., m_{15})$, где су $m_0, m_1, ..., m_{15}$, речи поруке,
- регистар у ком је резултат операције леве гране у итерацији i, означен је са X_i, где је $1 \leq i \leq 80$,
- регистар у ком је резултат операције десне гране у итерацији i, означен је са Y_i, где је $1 \leq i \leq 80$,
- ротација улево; X $\lll s$ означава ротацију речи улево за s позиција,
- ротација удесно ; X $\ggg s$; означава ротацију речи удесно за s позиција,
- сабирање по модулу 2^{32}, ознака $+$.

Функције

Функције које се користе су:

- $F_1(X, Y, Z) = X \oplus Y \oplus Z$,
- $F_2(X, Y, Z) = (X \wedge Y) \vee (\overline{X} \wedge Z)$,
- $F_3(X, Y, Z) = (X \vee \overline{Y}) \oplus Z$,
- $F_4(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \overline{Z})$,
- $F_5(X, Y, Z) = X \oplus (Y \vee \overline{Z})$,

где су X, Y, Z речи дужине 32 бита.

Константе

Као константе се, у зависности од тренутне рунде j и тренутне гране операције (леве - l, десне - r), узимају делови броја пре децималног зареза од следећих израза:

Грана l/r	j = 1	j = 2	j = 3	j = 4	j = 5
k_j^l	0	$2^{30}\sqrt{2}$	$2^{30}\sqrt{3}$	$2^{30}\sqrt{5}$	$2^{30}\sqrt{7}$
k_j^r	$2^{30}\sqrt[3]{2}$	$2^{30}\sqrt[3]{3}$	$2^{30}\sqrt[3]{5}$	$2^{30}\sqrt[3]{7}$	0

Реализација

Користи се пет регистара 5 регистара $cv_0, cv_1, cv_2, cv_3, cv_4$ са инцијалним вредностима $cv_0 = 0x67452301, cv_1 = 0xEFCDAB89, cv_2 = 98BADCFE, cv_3 = 1032547, cv_4 = C3D2E1F0$. Пошто се иницијализују регистри X_i, Y_i (где је $i = \{-4, -3, -2, -1, 0\}$) као:
$X_{-4} = Y_{-4} = cv_0 \ggg 10$,
$X_{-3} = Y_{-3} = cv_4 \ggg 10$,
$X_{-2} = Y_{-2} = cv_3 \ggg 10$,
$X_{-1} = Y_{-1} = cv_2$,
$X_0 = Y_0 = cv_1$,
у оквиру сваке рунде j (где је $1 \leq j \leq 5$), вредности X_i, Y_i (где је $1 \leq i \leq 80$) се рачунају на следећи начин:

$$X_i = (X_{i-4} \lll 10) + \Big((X_{i-5} \lll 10) + F_j(X_{i-1}, X_{i-2}, (X_{i-3} \lll 10)) + m_{\pi^l(i)} + k_j^l\Big) \lll s_i^l,$$

$$Y_i = (Y_{i-4} \lll 10) + \Big((Y_{i-5} \lll 10 + F_{6-j}(Y_{i-1}, Y_{i-2}, (Y_{i-3} \lll 10)) + m_{\pi^r(i)} + k_j^r\Big) \lll s_i^r$$

За вредности параметара π и s користи се табела 2.1. Табела је организована тако да је подељена у 5 различитих група, где свака група означава једну рунду. Вредности параметра π одређује која ће се реч m_i користити у итерацији, у зависности од тога да ли се операција примењује у левој или десној грани. Вредност параметра s одређује број места за колико ће се извршити ротација битова, у зависности да ли се ради о операцији леве или десне гране. Криптографска хеш вредност рачуна се на крају помоћу иницијалне вредности регистара и резултата унутар регистара X и Y и то на следећи начин:
$cv'_0 = cv_1 + X_{79} + Y_{79} + (Y_{78} \lll 10)$,
$cv'_1 = cv_2 + (X_{78} \lll 10) + (Y_{77} \lll 10)$,
$cv'_2 = cv_3 + (X_{77} \lll 10) + (Y_{76} \lll 10)$,
$cv'_3 = cv_4 + (X_{76} \lll 10) + Y_{80}$,
$cv'_4 = cv_0 + X_{80} + Y_{79}$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\pi^l(i)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi^r(i)$	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12
s_i^l	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8
s_i^r	8	9	9	11	13	15	15	5	7	7	8	11	14	14	12	6
i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
$\pi^l(i)$	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
$\pi^r(i)$	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2
s_i^l	7	6	8	13	11	9	7	15	7	12	15	9	11	7	13	12
s_i^r	9	13	15	7	12	8	9	11	7	7	12	7	6	15	13	11
i	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
$\pi^l(i)$	3	10	14	4	9	15	8	1	2	7	0	6	13	11	5	12
$\pi^r(i)$	15	5	1	3	7	14	6	9	11	8	12	2	10	0	4	13
s_i^l	11	13	6	7	14	9	13	15	14	8	13	6	5	12	7	5
s_i^r	9	7	15	11	8	6	6	14	12	13	5	14	13	13	7	5
i	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
$\pi^l(i)$	1	9	11	10	0	8	12	4	13	3	7	15	14	5	6	2
$\pi^r(i)$	8	6	4	1	3	11	15	0	5	12	2	13	9	7	10	14
s_i^l	11	12	14	15	14	15	9	8	9	14	5	6	8	6	5	12
s_i^r	15	5	8	11	14	14	6	14	6	9	12	9	12	5	15	8
i	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
$\pi^l(i)$	4	0	5	9	7	12	2	10	14	1	3	8	11	6	15	13
$\pi^r(i)$	12	15	10	4	1	5	8	7	6	2	13	14	0	3	9	11
s_i^l	9	15	5	11	6	8	13	12	5	12	13	14	11	8	5	6
s_i^r	8	5	12	9	12	5	14	6	8	13	6	5	15	13	11	11

Табела 2.1: Редослед порука и вредност ротације

2.4 Дигитални потпис

Дигитални потпис повезује поруку са особом која је шаље, односно обезбеђује гаранцију да порука заиста долази од особе која ју је и потписала. Обично се заснива на систему асиметричног шифровања (тачка 2.2) тиме што се сâм процес потписивања обавља помоћу приватног кључа особе која потписује поруку. Приватни кључ сâм по себи је јединствен и само особа која је власник кључа може и да потпише поруку, из чега следи да је порука заиста послата од стране искључиво те особе. Објашњење прве верзије потписа илустровано је примером 2.4.1, док је начин на који се поруке дигитално потписују у систему Биткоин дат у тачки 2.5.3.

Пример 2.4.1. Особа А жели да пошаље поруку особи Б кроз комуникациони канал који је несигуран, односно потенцијално подложан нападима. На поруку која се шаље примењује се хеш функција, која за циљ има да произвољну дужину поруке сведе на фиксну (детаљно објашњење хеш функција дато је у тачки 2.3), чиме се олакшава систем њеног потписивања. У супротном трајање тог процеса варирало би у зависности од дужине поруке. Након хеширања, потписује се (шифрује) добијена вредност приватним кључем особе А и порука заједно са њеном потписаном хеш вредношћу шаље се кроз канал. Особа Б дешифрује добијену хеш вредност поруке јавним кључем особе А. Након тога, особа Б примењује алгоритам хеширања на поруку коју је добила и те две хеш вредности пореди. Уколико су хешеви исти, порука није мењана.

Предности коришћења дигиталних потписа су следеће:

- аутентикација порука; као што је објашњено кроз пример 2.4.1, потписивање поруке је важно, јер особа Б из претходног примера може да одбаци добијену поруку уколико верификациони алгоритам не потврди њену аутентичност,

- немогућност порицања; особа А из примера 2.4.1, након слања поруке и потписивања, не може да оспори њено постојање и порекло. Особи Б је довољно да као доказ приложи поруку, потпис и јавни кључ. Ово је посебно корисно унутар мреже Биткоин где се новац након слања не може повратити,

- интегритет; најмања измена поруке након слања проузрукује велику промену у хеш вредности у односу на оригиналну вредност хеша израчунате на основу оригиналне поруке, чиме лако може да се провери интегритет поруке.

Дигитални потпис може да повезује и јавни кључ са особом. На томе се базира рад **сертификационог тела** (енг. Certificate Authority, CA). Када особа А генерише свој пар кључева, она свој јавни кључ даје сертификационом телу на дигитално потписивање. Улога тела је да веже идентитет особе са њеним јавним кључем. У будућој комуникацији особе А са особом Б, особа А уз свој јавни кључ доставља и дигитални потпис сертификационог тела као потврду да је она власник јавног кључа.

2.5 Шифарски систем са елиптичком кривом

У овом поглављу биће речи о примени елиптичке криве у сврхе система Биткоин. У првом делу уводе се појмови о елиптичким кривама над неким пољем, а у другом делу приказује се примена конкретне криве у систему Биткоин.

2.5.1 Елиптичка крива

Крива која је описана једначином $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ назива се **елиптичка крива**. Елиптичка крива је скуп тачака, односно парова (x, y) који задовољавају ову једначину. За операцију сабирања тачака на кривој неутрални елемент је тзв. бесконачно далека тачка, у ознаци $\emptyset$. У овој тачки се спајају горњи и доњи крајеви[3] вертикалних правих. Другим речима за произвољну тачку P, важи $P + \emptyset = \emptyset + P = P$. Права $x = x_0$ кроз тачку P сече криву у још једној тачки Q (y-координате тачака P і Q су решења квадратне једначине; ако је један корен квадратне једначине реалан, онда је и други). По дефиницији $P + Q = \emptyset$, односно $Q = -P$, односно тачка Q је супротна тачки P. Збир неке две тачке P и Q које припадају елиптичкој кривој дефинише се на следећи начин. Нека је R трећа тачка пресека праве $y = ax + b$ која пролази кроз тачке P и Q (x-координата тачке R је трећи корен кубне једначине која се добија заменом $y = ax + b$ у једначину криве; пошто су два корена те једначине, x-координате тачака P и Q реални, њен трећи корен је такође реалан). Тада је по дефиницији $P + Q = -R$.
Нека су задате тачке $P(x_1, y_1)$ и $Q(x_2, y_2)$, где је $P \neq -Q$. За израчунавање тачке $R(x_3, y_3) = P + Q$, када је $x_1 \neq x_2$, претходно морају да се израчунају вредности следећих израза:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}; \; \nu = \frac{y_1x_2 - y_2x_1}{x_2 - x_1}.$$

После тога се могу израчунати координате тачке R на основу израза:

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2$$
$$y_3 = -(\lambda + a_1)x_3 - \nu - a_3.$$

[3]Ово није у потпуности исправан израз с обзиром на то да праве иду у бесконачност.

Уколико важи да је $x_1 = x_2$, мења се начин израчунавања вредности λ и ν:

$$\lambda = \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3}; \quad \nu = \frac{-x_1^3 + a_4x_1 + 2a_6 - a_3y_1}{2y_1 + a_1x + a_3},$$

док су изрази за израчунавање координата тачке R исти као у случају када је $x_1 \neq x_2$. Скуп тачака елиптичке криве је група за операцију сабирања тачака.

Пример 2.5.1. Одредити тачку R на кривој $y^2 + y = x^3 - x$, која је добијена као $R = P + Q$, где је $P = (1, 0)$, а $Q = (2, -3)$. На основу једначине елиптичке криве, закључујемо:
$a_1 = 0$, $a_2 = 0$, $a_3 = 1$, $a_4 = -1$, $a_6 = 0$, $x_1 = 1$, $y_1 = 0$, $x_2 = 2$, $y_2 = -3$, ове податке замењујемо у горњим изразима и добијамо,

$$\lambda = \frac{-3 - 0}{2 - 1} = -3; \quad \nu = \frac{0 * 2 - (-3) * 1}{2 - 1} = 3.$$

Даље,

$$x_3 = (-3)^2 + 0 * (-3) - 1 - 2 = 6$$
$$y_3 = -(-3 + 0) * 6 - 3 - 1 = 14$$

Дакле, резултат сабирања је тачка $R = (6, 14)$.

2.5.2 Елиптичка крива у коначном пољу F_p

Уместо поља реалних бројева у ком изводимо операције над тачкама криве, користи се коначно поље F_p, чији су елементи остаци при дељењу простим бројем p применом операције mod p. Генератор групе тачака на елиптичкој кривој над пољем F_p је таква тачка G од које се множењем целим бројевима могу добити све тачке на кривој. Понекад не постоји једна тачка која генерише све тачке на кривој. Тада је тачка G генератор цикличне групе тачака криве. Ред криве n је број тачака на њој. За рачунање је важнији ред цикличне групе генерисане тачком G, односно најмањи број n за који је $nG = \emptyset$.

Пример 2.5.2. Нека је дата крива $y^2 = x^3 + 1$ у пољу F_5. Показује се да је тачка $G = (2, 3)$ генератор криве.
Прво рачунамо тачку $2G$, коју добијамо као $2G = G + G$. На основу дефинисаних формула (тачка 2.5.1), долазимо до резултата да је $\lambda = 1$, $\nu = 1$, одакле следи да је $x_3 = -1 \bmod 5 = 4$, док је $y_3 = 0$, тј. $2G = (4, 0)$. Даље, $3G = 2G + G = (4, 0)$, док је $4G = 2G + 2G = (0, 4)$. Одавде можемо да закључимо да тачка $4G$ и тачка $2G$ имају исту x координату, што значи да су то супротне тачке, односно да важи $4G = -2G$, одакле следи да је $4G + 2G = 6G = \emptyset$. Према томе, циклична група генерисана тачком G има 6 тачака.

За задату тачку G и број d, тачка $T = dG$ може се израчунати помоћу $O(\log n)$ сабирања тачака криве алгоритмом који се састоји од низа удвостручавања и сабирања. Овај алгоритам назива се **дуплирај и додај** [12]. Алгоритам иницијализује резултат на вредност тачке генератора и конвертује вредност n у његову бинарну репрезентацију. Итерација почиње од бита највеће ка биту најмање тежине и у сваком кораку, чувајући међурезултат који се на почетку итерације дуплира, додаје вредност међурезултата уколико је вредност бита у тој итерацији 1. Као што је већ напоменуто, операције се све изводе у коначном пољу F_p, тј. сви међурезултати

своде се по модулу p. Детаљан приказ алгоритма дат је у наставку:

Улаз: $n = \sum_{i=1}^{t} d_i 2^i$, G
Излаз: Вредност T, где је $T = dG$
$T = G$;
for $i = t$ - *1* ***downto*** *0* **do**
  $T = T + T$;
  if $n_i = 1$ **then**
    $T = T + G$;
return T

Супротно томе, уколико се зна T и G, одређивање броја d је тзв. проблем дискретног логаритма са елиптичком кривом[4]. За овај проблем се у општем случају не зна алгоритам који би био битно ефикаснији од прегледања свих тачака на елиптичкој кривој.

2.5.3 Елиптичка крива у систему Биткоин

Крива која се користи у систему Биткоин назива се *secp256k1* и део је групе под називом апстрактне Коблицове (Neal Coblitz) криве [3]. Дефиниција криве у коначном пољу F_p је:

$$y^2 = x^3 + ax + b,$$

Карактеристично за све Коблицове криве је да су њихови параметри унапред дефинисани. Коришћење унапред дефинисаних параметара доприноси ефикаснијим сабирању тачака криве. Конкретно *secp256k1* има следеће параметре [13]:

- крактеристика поља је p = 0xFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F,
- константе су $a = 0$, $b = 7$,
- ред групе је n = 0xFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141,
- генератор криве је G = (0x79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798, 0x483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8).

У систему Биткоин приватни (тајни) кључ је неки број d, $0 < d < n$. Приватном кључу одговара јавни кључ, односно тачка $T = dG$.
Улога елиптичке криве у систему Биткоин је начин на који особа А потписује поруку упућену особи Б. Дигитални потпис помоћу елиптичке криве (енг. Elliptic Curve Digital Signature Algorithm, ECDSA) је пар у ознаци (r, s), чији поступак генерисања је дат у наставку.

Генерисање потписа

Најпре се бира псеудо-случајан број k за рачунање прве компоненте пара који дефинише потпис на следећи начин:

$$(x_1, y_1) = kG$$
$$r = x_1 (\text{mod } n),$$

[4]http://poincare.matf.bg.ac.rs/%7Eezivkovm//nastava/kripto.pdf

где је G претходно дефинисан генератор групе *secp256k1*, а n ред криве. Други елемент пара s рачуна се као:

$$s = (H + r\ d)k^{-1} (\text{mod } n),$$

где је H криптографска хеш вредност поруке, а d вредност приватног кључа особе која потписује поруку. Савет је да се приликом сваког новог потписивања бира увек друга вредност за параметар k. Начин на који особа, којој је порука упућена, верификује дигитални потпис и утврђује да ли је исправан, дат је у наставку.

Провера потписа

Након пријема пара (r, s), особа која проверава дигитални потпис рачуна:

$$w = s^{-1} (\text{mod } n)$$
$$u_1 = H\ w (\text{mod } n)$$
$$u_2 = r\ w (\text{mod } n)$$
$$(x_2, y_2) = u_1 G + u_2 T.$$

Особа рачуна криптографску хеш вредност примљене поруке и то је H. Јавни кључ у ознаци T, особе која је потписала поруку, свима је познат. Генератор групе је G, док је ред групе n. Уколико је $r = x_2$ дигитални потпис се сматра исправним. Порука у систему Биткоин одговара једној трансакцији (глава 3).

2.6 Kôд Base58

Kôд Base58 служи за представљање великих бројева (или ASCII стрингова, који се такође могу схватити као бројеви записани у систему са основом 256) стрингом знакова из специјалне азбуке од 58 знакова.
Азбуку коју користи Base58 чине:

- карактери од 'A' до 'Z', осим слова: 'O', слова 'I',
- карактери од 'a' до 'z', осим слова 'l',
- цифре од 1 - 9 (цифра 0 се не користи).

Разлог искључивања појединих карактера је то што имају исту или јако сличну репрезентацију. Овај скуп карактера дат је у табели 2.2. Алгоритам кодирања примењен на стринг s[0,..n-1] замењује сваки знак s[i] (где је $0 \leq$ i $< n$) одговарајућом вредношћу из ASCII табеле помноженом са бројем 256^i. Кодирање се завршава представљањем добијеног броја у систему са основом 58 и заменом добијених цифара развоја знацима из табеле 2.2.

Пример 2.6.1. За кодирање ASCII стринга „Bit", потребно је одредити бројевну вредност њених карактера (вредности из ASCII табеле):
B - 66,
i - 105,
t - 116,
потом се свака од тих вредности множи са бројем 256^i, где је i позиција слова у речи, па је вредност слова „B" $66 * 256^2 = 66 * 65536 = 4325376$, слова „i" $105 * 256^1 = 105 * 256 = 26880$, док је вредност слова „t" $116 * 256^0 = 116$. Сума свих ових вредности је 4352372. Даље, одређују се цифре ове вредности у систему са основом 58. Вредност се дели бројем 58 и памте се остаци при том дељењу.

Вредност	Карактер	Вредност	Карактер	Вредност	Карактер	Вредност	Карактер
0	1	1	2	2	3	3	4
4	5	5	6	6	7	7	8
8	9	9	A	10	B	11	C
12	D	13	E	14	F	15	G
16	H	17	J	18	K	19	L
20	M	21	N	22	P	23	Q
24	R	25	S	26	T	27	U
28	V	29	W	30	X	31	Y
32	Z	33	a	34	b	35	c
36	d	37	e	38	f	39	g
40	h	41	i	42	j	43	k
44	m	45	n	46	o	47	p
48	q	49	r	50	s	51	t
52	u	53	v	54	w	55	x
56	y	57	z				

Табела 2.2: Скуп карактера за кôд Base58

4352372 = 58*75040 + 52
75040 = 58*1293 + 46
1293 = 58*22 + 17
22 = 58*0 + 22
Вредност добијених остатака 22, 17, 46, 52 одговара неким карактерима из табеле 2.2. Посматрамо колону „Вредност" и у зависности од остатка тражимо њему одговарајући карактер. Дакле, кôд Base58 стринга „Bit" је: „PJou"[5].

2.7 Јавни кључ и адреса у систему Биткоин

Јавни кључ (добијен од приватног, поступком датим у тачки 2.5.2) корисника у систему Биткоин користи се за израчунавање корисничке Биткоин адресе на основу корака описаних у наставку:

- јавни кључ састоји се од тачке на елиптичкој кривој са X и Y координатом. Ова два податка трансформишу[6] се у хексадекадни запис дужине 64 бајта (32 бајта за сваку од координата). У запис улази и додатни бајт који носи информацију да ли је запис координата компримован. Некомпримовани запис је заправо бинарни запис управо X и Y координата, док компримован представља запис X координате помоћу које се, решавањем квадратне једначине (тачка 2.5), може доћи до вредности Y координате;

- координате се надовезују једна на другу и тако добијена вредност се криптографски хешира. У ту сврху се прво користи функција SHA256 (тачка 2.3.1), а потом функција RIPEMD160 (тачка 2.3.2). Улога првог хеширања је заштита

[5]Ово се може проверити неким од јавно доступних алата за Base58 кодирање и декодирање, као што је `https://www.appdevtools.com/base58-encoder-decoder`.

[6]Функције за трансформацију су из библиотеке OpenSSL[7].

информација о правим вредностима координата X и Y, док је сврха другог хеширања додатно сакривање као и компримовање података. Дужина записа хеш вредности на крају овог корака је 20 бајтова;

- резултат претходног корака користи се као улаз хеш вредности функције SHA256^2 [8]. Чува се првих 4 бајта резултујуће хеш вредности, као контролна сума (заштитни кôд). Овај корак има за циљ смањење грешака код уноса неке јавне адресе (приликом трансакције) једноставним прерачунавањем контролне вредности. На овако добијен стринг, додаје се вредност 0x0 на почетак записа;

- резултат претходног корака користи се као аргумент алгоритма Base58 (тачка 2.6) и резултујућа вредност је Биткоин адреса.

Вероватноћа појављивања две исте Биткоин адресе одговара вероватноћи колизије хеш функције (објашњена у тачки 2.3). Подсетимо, формула за колизију хеш функције гласи:

$$p = \frac{n^2}{2K}$$

Вредност броја K одговара броју битова на излазу хеш функције, што је у случају генерисања Биткоин адресе 160, одакле закључујемо да укупан број Биткоин адреса има $K = 2^{160}$. Рецимо да ће за неколико година на планети Земљи живети укупно 8 милијарди становника и рецимо да сваки становник има 10 милиона Биткоин адреса. Укупан број активних адреса, односно већ пронађених хеш вредности биће $n = 80\ 000\ 000\ 000\ 000\ 000 \sim 2^{57}$. Вероватноћа p за постојање две исте Биткоин адресе према горњем изразу биће једнака:

$$p = \frac{2^{114}}{2^{161}} = 2^{-47},$$

што је занемарљиво мало.

2.8 Систем Биткоин

Криптографска решења објашњена у претходним тачкама су срж система Биткоин. Поруке које се размењују између корисника унутар система чине размену биткоина у виду трансакције. Свака трансакција је хеширана (тачка 2.3), дигитално потписана (тачка 2.4) и постоји као податак у систему. Специјализовани корисници (тачка 4.2) обједињују више трансакција у блок. Сваки блок има јединствену хеш вредност добијену над подацима карактеристичним за тај блок и за тренутак када је формиран (тачка 4.3). Хеширање доприноси очувању интегритета података, односно онемогућава злонамерном кориснику да их мења. Више тако образованих блокова уланчавају се један на други и формирају ланац (глава 4). Разлог уланчавања је додатни ниво очувања интегритета и сигурности, јер хеш вредност блока директно зависи од хеш вредности претходног блока на који је уланчан. Како сваки корисник чува своју копију ланца блокова (чиме се добија на децентрализацији система), злонамерном кориснику је практично[9] онемогућено да промени било шта унутар ланца. Мењањем било ког податка у средини ланца проузрукује неисправно стање ланца од блока који је мењан, па све до врха ланца. На тај начин злонамерни корисник и његов злонамеран поступак се брзо детектују, јер су подаци у његовом ланцу другачији у односу на податке који имају сви остали корисници у мрежи.

[8] Ознака „^2” одговара примени криптографске хеш функције два пута.

[9] Теоријски јесте могуће, како је објашњено у тачки 4.5

3 Трансакције

Улога мреже Биткоин је чување свих икада обављених трансакција унутар ње. Систем се састоји од ланаца блокова (глава 4), где сваки блок у ланцу садржи више формираних трансакција. Свака трансакција се састоји од улазног дела **TxIn** и излазног дела **TxOut**. Излаз трансакције садржи информације о Биткоин адреси примаоца, као и о количини новца коју прималац добија по завршетку трансакције. Након обављене трансакције прималац може касније да користи примљени новац само уколико поседује приватни кључ коришћен у генерисању њему одговарајућег јавног кључа, односно његове Биткоин адресе ка којој је претходно послат новац (тачка 2.7). Улазни део трансакције (поред других података, тачка 3.1) садржи референцу на претходну криптографску хеш вредност излаза претходне трансакције (која покрива износ на улазу те трансакције) и потписану хеш вредност те трансакције. Важност чувања приватног кључа особе испољава се у тренутку када особа жели да започне трансакцију са неким другим у мрежи користећи претходно примљене биткоине. Губитак приватног кључа онемогућава особу да дигитално потпише трансакцију коју креира, односно она не може да докаже власништво над биткоинима. Особа за коју се каже да поседује неки произвољан број биткоина, заправо поседује једну или више референци ка претходним трансакцијама које показују како је она дошла у посед тих биткоина. Трансакција може да има више улаза и излаза.

Биткоин новчаник (више речи о новчаницима биће унутар главе 5) је програм помоћу ког се шаљу и примају биткоини и тиме омогућава особи (власнику новчаника) да буде укључен у трансакције које се обављају. Додатна улога новчаника је одржавање базе података о референцама на претходне трансакције за које још увек није дефинисан излаз, односно чување свих биткоина који још увек нису потрошени. Биткоини који ће бити део будућих трансакција, чувају се у бази под називом **UTXO** (енг. Unspent Transaction Output). Сваки чвор у мрежи чува своју копију UTXO базе и ажурира је са појавом нових блокова у ланцу. Систем ефикасно функционише када је излаз претходне трансакције у целости искоришћен као улаз друге, али то није увек случај. Следећи пример илуструје ситуацију када се троши само део биткоина неке претходне трансакције.

Пример 3.0.1. Особа А у својој UTXO бази има x биткоина као резултат неког претходног TxOut-а и креира трансакцију Tx ка особи Б, тако да троши y биткоина, где је $y < x$ за неки производ. Како се претходна трансакција у којој су добијени биткоини, не може разбити на мање делове, идеја је да се особи Б проследе биткоини који су њој намењени, док преостали део биткоина постаје део додатног излаза трансакције Tx који генерише новчаник особе А. Адреса примаоца везана за додати излаз је адреса особе А, а количина биткоина је вредност $x-y$[1] као кусур. Биткоини се враћају назад у базу UTXO као резултат дела трансакције особе А са самом собом. Пракса је да се за Биткоин адресу особе А у том случају користи увек нова адреса под заједничким називом „адреса за кусур” (енг. change address), што повећава сигурност.

Особа помоћу новчаника креира трансакцију Tx и даље је прослеђује осталим новчаницима у мрежи. Новчаници у мрежи реагују на захтев за креирање трансакци-

[1]Ова вредност може додатно бити умањена за вредност провизије на трансакцију, тачка 4.2.1.

је Tx и врше низ провера њене исправности. Поступак провере нове трансакције састоји се од:

- провере да ли је укупна сума биткоина на улазу трансакције већа или једнака од оне на излазу (уколико ова неједнакост није задовољена, трансакција се одбацује као неисправна);
- провере да ли је трансакција исправно потписана (уколико дигитални потпис трансакције не одговара јавном кључу пошиљаоца, трансакција се одбацује као неисправна);
- провере да ли особа која шаље биткоине има довољан број биткоина на располагању. Како сваки чвор има копију базе UTXO, провера може лако да се изврши упитом у ту базу. Она је ефикаснија од проверавања читавог ланца блокова који садржи све трансакције. Додатној ефикасности доприноси и то што је приликом покретања новчаника, база UTXO учитана у RAM уместо на диск, па се провера обавља брже.

Уколико новчаник који креира трансакцију, заједно са другим новчаницима у мрежи којима је трансакција прослеђена на проверу, утврди да је трансакција исправна, сви новчаници који су учествовали у њеној провери исправности смештају је у своју „базу непотврђених трансакција" (енг. unconfiremd transactions memory pool, mempool). Одатле је преузима чвор рудар (тачка 4.2) и укључује је у блок који он формира.

3.1 Скрипте за трансакцију

Власник новчаника, на који се односи једна Биткоин адреса, прима биткоине у склопу трансакције Tx. За даље трошење биткоина, власник мора да потврди да поседује приватни кључ, као и њему одговарајући јавни кључ коришћеног за генерисање Биткоин адресе која је на излазу трансакције Tx, и тиме гарантује власништво над примљеним биткоинима. Поменута трансакција Tx на свом излазу TxOut поставља услове под којим биткоини могу да се даље троше. Услови се односе управо на то да је неопходно да особа докаже власништво над новчаником и приватним кључем који је коришћен за генерисање Биткоин адресе. Услови који се постављају на излазу трансакције, као и поступак доказивања власништва, генеришу се помоћу две скрипте, тзв. „скрипте за трансакцију" (енг. transaction scripts). Скрипте се конструишу помоћу имплементираног програмског језика **скрипт** (енг. script). Језик је базиран на раду са стеком, који је у кôду представљен кроз вектор бројева. Скрипт дефинише скуп оператора и операнада који се над тим операторима примењује. Примена једног оператора над операндима (за неке операторе они нису потребни) чини инструкцију. Како је базиран на раду са стеком, принцип извршавања је да се операнди скидају са стека (узимају са краја вектора), одговарајући оператори их користе, а након извршене инструкције резултат се - или поставља на врх стека (гура на крај вектора) или игнорише (што је показатељ да је инструкција неометано извршена до краја). Скрипт има далеко мањи број оператора у поређењу са данашњим програмским језицима управо због повећања сигурности. Мањак комплексности смањује могућност малверзација помоћу њега. Из истог разлога он није Тјуринг комплетан језик, јер се тиме добија одсуство петљи, чије коришћење би за последицу имало успорење или чак пâд мреже, на пример, у случају бесконачне петље. Део имплементације скрипт језика покривен је тачком 7.1.3.

Две наредне скрипте су део сваке трансакције (њихова практична примена је објашњена у тачки 3.2):

- <scriptPubKey>; поставља услове који морају бити испуњени да би биткоини добијени, као део неке од претходних трансакција, били доступни за њихово трошење као део будуће трансакције,

- <scriptSig>; садржи податке који потенцијално испуњавају услове постављене кроз <scriptPubKey>.

Протокол налаже дефинисање ових скрипти тако да се скрипта <scriptPubKey> налази на излазном делу TxOut, а скрипта <scriptSig> на улазном делу TxIn. Срж скрипте <scriptSig> је дигитални потпис трансакције. Срж скрипте <scriptPubKey> је низ инструкција које постављају услове о даљем трошењу послатих биткоина. Поступак дигиталног потписивања и испитивања исправности тог потписа дат је у оквиру тачке 2.5.3. Провера исправности (коју врше други новчаници у мрежи, као што је већ речено на почетку ове главе) креиране трансакције врши се тако што се прво скрипта <scriptSig> и скрипта <scriptPubKey> редом надовежу, а потом се извршавају инструкције унутар њих. Уколико је на врху стека, након свих завршених инструкција, вредност *тачно* - иницирана трансакција сматра се исправно потписаном. Део инструкција које се могу користити унутар скрипти дате су кроз преглед типа трансакције „плаћања ка адреси", тачка 3.2.

Као што је већ речено трансакција може имати више улаза и излаза, а дигитални потпис трансакције не мора да се односи на сваки од њих. У сврху означавања који од улаза је потписан, користи се индикатор под називом **тип хеша** (енг. hashtype). Индикатор се ставља на крај скрипте <scriptSig> и може имати различите вредности:

- SIGHASH_ALL; коришћењем овог индикатора гарантује се да се дигитални потпис односи на све улазе и све излазе транаскције,

- SIGHASH_NONE; коришћењем овог индикатора гарантује се да се дигитални потпис односи на све улазе, али ни на један излаз трансакције. Интерпретација за потребе коришћења овог типа хеша је да особа, која започиње трансакцију, шаље биткоине али не дефинише излаз трансакције, односно особи није битно како ће се они даље трошити,

- SIGHASH_SINGLE; начин на који се неки улаз реферише у неком излазу је кроз индекс, где улазу индекса i, одговара излаз индекса i. Коришћењем овог индикатора гарантује се да је улаз i дигитално потписан заједно са њему одговарајућим излазом (излаз са истим индексом i као и улаз)[2].

У табели 3.2 налази се списак поља садржан унутар TxIn, а у табели 3.1 списак поља садржан унутар TxOut.
Вредност поља на улазу трансакције TxIn под називом *Број секвенце*, унутар табеле 3.2, означава трансакцију која није финална. Уколико је вредност поља *Број секвенце* различита од 0xFFFFFFFF, то представља индикатор да ће се неки од улаза трансакције убудуће мењати. Трансакција која није финална, у старијим верзијама протокола, могла је да се нађе унутар базе непотврђених трансакција, али је рудари не би убацивали ни у један блок док не буде финална, односно док не буде имала вредност 0xFFFFFFFF. Са новијом верзијом протокола таква трансакција

[2]Ово је под условом да се индикатор користи у комбинацији са још једним типом хеша SIGHASH_ANYONECANPAY, чија је улога да модификује очекивано понашање типа хеша.

Поље	**Опис поља**	**Величина поља**
Број биткоина	Ненегативан број који одговара количини биткоина (сатошија) која се шаље на Биткоин адресу	8 бајтова
Дужина скрипте <scriptPubKey>	Ненегативан број који одговара дужини скрипте која дефинише услове даљег коришћења биткоина као део будуће трансакције	1 - 9 бајтова
<scriptPubKey>	Низ инструкција скрипт језика који поставља услове на излазу тренутне трансакције	Зависи од величине скрипте

Табела 3.1: Изглед поља TxOut

Поље	**Опис поља**	**Величина поља**
Хеш вредност претходне трансакције	Референца на претходну трансакцију добијену на Биткоин адресу чији се износ прослеђује као део нове трансакције	32 бајта
Индекс претходне трансакције	Ненегативан број који одговара индексу једног излаза (уколико их има више) претходне трансакције.	4 бајта
Дужина скрипте <scriptSig>	Ненегативан број који одговара дужини скрипте <scriptSig>	1 - 9 бајтова
<scriptSig>	Низ инструкција скрипт језика који проверава испуњеност услова постављених на излазу претходне трансакције скриптом <scriptPubKey>	Зависи од величине скрипте
Број секвенце	Обично број 0xFFFFFFFF	4 бајта

Табела 3.2: Изглед поља TxIn

не сматра се допустивом, јер би на тај начин могао да се изврши напад на мрежу. Потенцијални напад може да буде такав да се база непотврђених трансакција затрпа трансакцијама које нису финалне и тако добије успорење унутар мреже.

Свака трансакција има јединствену ознаку у ланцу **TXID**. Ознака се добија конкатенацијом тренутне верзије протокола, свих улаза и излаза трансакције и податаком о броју секвенце. Овако надовезани подаци се хеширају коришћењем криптографске хеш функције SHA256^2 и добијена хеш вредност се потписује приватним кључем особе која креира трансакцију. TXID се користи као референца која означава која претходна трансакција, односно њен излаз постаје улаз нове трансакције. Додатна улога хеширања и потписивања је очување интегритета трансакције, односно спречавања могућности њеног мењања након што постане део мреже.

3.2 Плаћање ка хешираном јавном кључу (плаћање ка адреси)

Плаћање ка хешираном јавном кључу је тип трансакције који се најчешће[3] користи унутар мреже. Као идентификација примаоца трансакције наводи се његова Биткоин адреса, која се након низа корака (објашњених у оквиру тачке 2.7), генерише на основу јавног кључа примаоца. Као што је већ објашњено у оквиру тачке 3.1, да би особа имала право на трошење биткоина (које је примила кроз неку од претходних трансакција), она мора да испуњава услове претходне трансакције дате кроз скрипту <scriptPubKey>, док скрипта <scriptSig> потенцијално доказује испуњеност тих услова (на пример, провера исправности дигиталног потписа, тачка 2.5.3).

Садржај скрипти

Скрипта <scriptSig> састоји се од:

- <sig>; дигитални потпис нове трансакције у којој особа покушава да потроши претходно примљене биткоине. Трансакција се дигитално потписује коришћењем приватног кључа (тачка 2.5.3),
- <pubKey>; јавни кључ који одговара приватном кључу коришћеном за генерисање дигиталног потписа <sig>.

У наставку су дати оператори (и њихов опис) који се користе у оквиру скрипте <scriptPubKey> као део трансакције типа „плаћање ка хешираном јавном кључу”:

- OP_DUP; копира последњу вредност са стека и ставља је на стек. Оператор не захтева операнд;
- OP_HASH160; резултат примене оператора је хеш вредност над датим операндом и постављање резултата на врх стека. Хеш вредност се рачуна применом криптографских хеш функција SHA256 (тачка 2.3.1) и RIPEMD160 (тачка 2.3.2) у наведеном редоследу. Оператор захтева један операнд са врха стека, а у случају трансакције типа „плаћање ка хешираном јавном кључу”, вредност операнда одговара вредности јавног кључа примаоца биткоина;
- OP_EQUALVERIFY; примена овог оператора захтева два операнда. Са врха стека скидају се две последње вредности и пореде се. Уколико су вредности операнада једнаке, резултат поређења се не ставља на стек. У супротном прекида се парсирање скрипте (трансакција је неисправна);
- OP_CHECKSIG; оператор проверава исправност дигиталног потписа датог унутар скрипте <scriptSig>. Оператор захтева два операнда, а то су два податка унутар скрипте <scriptSig>, дигитални потпис и јавни кључ особе која касније буде трошила биткоине[4]. Уколико је нова трансакција исправно ди-

[3]Према подацима преузетим 25.07.2020. године са сајта `https://www.quantabytes.com/articles/a-survey-of-bitcoin-transaction-types` преко ког се могу пратити и анализирати подаци мреже Биткоин постоји преко 86 000 000 појављивања трансакције типа „Плаћање ка хешираном јавном кључу” унутар мреже.

[4]Особа која поставља услове не зна оригиналну вредност јавног кључа особе која ће касније трошити послате биткоине, већ зна само хеш Биткоин адресу на коју биткоине шаље, односно хеширану вредност јавног кључа.

гитално потписана, резултат оператора поставља се на врх стека као вредност *тачно*. У супротном, прекида се даље парсирање скрипте (трансакција је неисправна).

Додатно, скрипта <scriptPubKey> садржи и као податак <pubKeyHash>, чија вредност одговара криптографској хеш вредности јавног кључа њему одговарајуће Биткоин адресе као примаоца трансакције. Добијање хеш вредности јавног кључа одговарајуће Биткоин адресе, састоји се из декодирања Base58 адресе, а потом уклањања 0x0 са почетка добијеног стринга и контролне суме са његовог краја. Кораци су инверзни у односу на последња два корака дата унутар објашњења како се од јавног кључа добија Биткоин адреса (тачка 2.7).

Реализација протокола

Као што је већ речено у тачки 3.1, за проверу исправности трансакције скрипта <scriptSig> и скрипта <scriptPubKey> надовезују се једна на другу, у наведеном редоследу, и тако добијена скрипта се парсира. У табели 3.3 дат је списак инструкција унутар друге колоне и стање стека у време извршавања инструкције унутар прве колоне. Празан стек се попуњава подацима из скрипте <scriptSig>, а то су дигитални потпис <sig> који је генерисан коришћењем приватног кључа и њему одговарајућег јавног кључа, <pubKey>. Потом се примењује оператор OP_DUP који за операнд има вредност која је тренутно на врху стека, <pubKey> и копију операнда ставља на врх стека као резултат. Затим следи примена оператора OP_HASH160 чијем операнду одговара јавни кључ, <pubKey>. Резултат је криптографска хеш вредност која се поставља на врх стека и то је <pubKeyHashA>. Затим се на врх стека ставља податак <pubKeyHash>. Последње две вредности са стека, <pubKeyHash> и <pubHashA>, скидају се и над њима се примењује оператор OP_EQUALVERIFY. Уколико су операнди једнаки, наставља се парсирање скрипте, а на стек се не ставља резултат поређења. Последњи оператор OP_CHECKSIG провера потпис трансакције користећи претходни податак са стека, <sig>. Уколико је дигитални потпис исправан, на врх стека се поставља вредност *тачно* и испуњен је услов за трошење биткоина.

3.3 Трансакција са више потписа (m од n)

Новије верзије Биткоин протокола подржале су трансакцију која захтева да она буде потписана са више различитих дигиталних потписа. Трансакција овог типа реализује се тако што се скрипте <scriptPubKey> и <scriptSig> модификују тако да <scriptPubKey> садржи n (где је $n > 1$) различитих јавних кључева, док скрипта <scriptSig> садржи минималан број m (где је $0 < m \leq n$) исправних дигиталних потписа генерисаних помоћу приватних кључева (њима одговарајући јавни кључеви налазе се у скрипти <scriptPubKey>). Трансакција оваквог типа назива се „трансакција са више потписа" („трансакција m од n"). Предност коришћења овог протокола лежи у томе што је за обављање трансакције потребно више различитих приватних кључева и тиме се отежава могућност неовлашћеног трошења биткоина. У трансакцији m од n прималац биткоина мора претходно да проследи информације о јавном кључу и минималном броју потписа потребних за проверу исправности трансакције пошиљаоцу. Практична примена трансакције која захтева више потписа дата је на крају ове тачке кроз различите примере.

Провера исправности трансакције врши се надовезивањем скрипти <scriptSig> и <scriptPubKey> у датом редоследу. Затим се над тако надовезаним подацима уну-

Стек	Скрипт команда
Празан стек	<sig><pubKey>OP_DUP OP_HASH160<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
<sig> <pubKey>	OP_DUP OP_HASH160<pubKeyHash>OP_EQUALVERIFY OP_CHECKSIG
<sig> <pubKey> <pubKey>	OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
<sig> <pubKey> <pubHashA>	<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
<sig> <pubKey> <pubHashA> <pubKeyHash>	OP_EQUALVERIFY OP_CHECKSIG
<sig> <pubKey>	OP_CHECKSIG
Тачно	

Табела 3.3: Плаћање ка хешираном јавном кључу

тар скрипти примењује оператор скрипт језика OP_CHECKMULTISIGVERIFY. Оператор проверава исправност дигиталног потписа датог унутар скрипте <scriptSig> у односу на јавни кључ дат унутар скрипте <scriptPubKey>. Проверава се први дигитални потпис, са првим јавним кључем, други са другим итд. Уколико неки дигитални потпис не одговара јавном кључу, проверава се следећи по реду и проверавање се наставља докле год не буде пронађен јавни кључ који је одговарајући потпису (претрага највише иде до n-тог). Битно је обратити пажњу на редослед дигиталних потписа и јавних кључева унутар скрипти, јер оператор OP_CHECKMULTISIGVERIFY не проверава оне који су по редоследу иза јавног кључа који није прошао проверу. Уколико се на крају парсирања скрипти испостави да је број исправних дигиталних потписа мањи од m, трансакција се сматра неисправном.

Примери коришћења трансакције са више потписа:

- трансакција са посредником; електронска куповина може да буде рискантна по купца, јер он може бити изложен ризику губитка новца, а да не добије робу коју је захтевао. Улога посредника и „трансакција са више потписа" може да га заштити (илустровано примером 3.3.1),
- заштита новца; сигурнији начин обављања трансакције јесте коришћењем више приватних кључева (илустровано примером 3.3.2),
- 2 од 2 кључа; случај када су неопходна оба потписа, односно кад је $m = n$ (илустровано примером 3.3.2).

Пример 3.3.1. За ситуацију где имамо купца, посредника и продавца, довољно је да имамо 2 од 3 потписа за обављање трансакције. Уколико се купац и продавац договоре, посредник није неопходан за закључење трансакције и она се може обавити без њега. Уколико једна особа покуша да превари другу (на пример продавац пошаље погрешну робу или робу која не одговара спецификацији датој при куповини), посредник одлучује која особа је оштећена и употребом свог приватног кључа и кључа те особе, реализује трансакцију где се враћају биткоини оштећеној особи.

Пример 3.3.2. Особа има три приватна кључа који учествују у потписивању. Један од њих је строго чуван, на пример у сефу. Други може да буде чуван на мобилном телефону или на неком другом медијуму који је власнику кључева највише доступан (који ће најчешће бити коришћен), док трећи може да буде у власништву трећег лица, провајдера. Свака трансакција, која се обавља, подразумева 2 од 3 потписа. Овакав систем штити особу од губитка биткоина у случају да изгуби један од два кључа. Такође, штити га и од самог провајдера који не може да преузме потпуну контролу над биткоинима, јер му за крађу недостаје барем још један приватни кључ.

Пример 3.3.3. Претходни пример је могуће реализовати и без улоге провајдера, већ само коришћењем два медијума за чување (на пример, рачунар и мобилни телефон). Приликом сваке трансакције, неопходна су оба потписа што онемогућава злонамерну особу, која би успела да дође у посед једног од два приватна кључа, да обави било какву трансакцију без другог.

3.4 Плаћање ка хешираној скрипти као адреси

Плаћање ка хешираној скрипти (енг. Pay to script hash, P2SH) је стандардизована трансакција, коју је предложио Гевин Андерсен (Gavin Andresen) 2012. године у оквиру свог предлога о унапређењу система Биткоин BIP16 [14] (дефиницију термина BIP погледати у тачки 4.4), као решење проблема који се појавио приликом употребе трансакције са више потписа m од n (тачка 3.3). Проблем се састоји у структури скрипте <scriptPubKey>, коју генерише новчаник пошиљаоца (особа која иницира трансакцију) ка новчанику који захтева да се трансакција касније обради као „трансакција са више потписа". На тај начин новчаник пошиљаоца поставља услове под којим ће моћи да се даље троше биткоини, иако је заправо прималац тај који треба да има информације колики је најмањи број дигиталних потписа неопходних за даље трошење биткоина и који су све јавни кључеви део трансакције.

Решење које је предложио Гевин је у премештању трансакцијских услова из скрипте <scriptPubKey> у скрипту <scriptSig>, односно премештање одговорности о даљем трошењу биткоина са пошиљаоца на примаоца. У случају трансакције са више потписа, унутар P2SH протокола, скрипта <scriptSig> садржи и дигиталне потписе и јавне кључеве особа који су део трансакције, док скрипта <scriptPubKey> садржи криптографску хеш вредност скрипте <scriptSig> у дужини од 20 бајтова. Како је већ речено, прималац трансакције m од n шаље пошиљаоцу јавне кључеве и вредност броја m, али у случају трансакције P2SH, прималац даје пошиљаоцу само хеш вредност скрипте <scriptSig>. Овај систем за последицу има и то да протокол налаже коришћење посебног типа адресе у називу „скрипт адреса" (енг. script address). Кораци за генерисање ове адресе слични су као унутар тачке 2.7. Разлика је у томе што се уместо јавног кључа користи скрипта <scriptSig>. Додатна

разлика је и то што се на резултат добијен корацима у генерисању Биткоин адресе (објашњеним у тачки 2.7) пре корака кодирања Base58, додаје на почетак децимална вредност 5. Још један разлог за коришћење P2SH протокола уместо трансакције m од n је смањење провизије на трансакцију коју плаћа пошиљалац (објашњење прозовизије на трансакцију дато је у оквиру тачке 4.2.1), а на чију цену утиче сложеност скрипти за трансакцију. Примена P2SH трансакције са више потписа избацила је из употребе трансакцију m од n.

4 Ланац блокова

Разлог великој популарности биткоина (али и алткоинима) какву данас има је технологија на којој се систем заснива и кроз који је представљена, а то је ланац блокова. База података свих трансакција икада обављених у систему организована је у блокове (где се више трансакција налази у оквиру једног блока који има своју хеш вредност) који су међусобно повезани у ланац. Сâма база је децентрализована, односно сваки корисник унутар мреже (у даљем тексту чвор) има копију те базе, учествује у њеном ажурирању као и у синхоронизовању података са осталим чворовима у мрежи. Карактеристика технологије да прати и уписује податке, након чега они не могу бити мењани, а уз то да буду транспарентни, даје широк спектар потенцијалних унапређења постојећих система. Јапански произвођач моторних возила Тојота, ради на истраживању на који начин технологија ланаца блокова може да се искористи у области ауто индустрије. Постоји могућност да ћемо у будућности имати увид у комплетну историју аутомобила који купујемо: колико километара је возило прешло, да ли је било учесник у судару, детаљне информације о деловима од којих је возило састављено итд. Сличан систем може да се примени у индустрији хране, где купац има увид у комплетну историју контроле квалитета неког производа. Оно што овај ланац чини сигурним у систему Биткоин је начин додавања нових блокова од којих сваки мора имати „доказ о раду".

4.1 Доказ о раду

Концепт **доказ о раду** (енг. proof-of-work) представљен је 1993. године као решење за напад који се састојао од масовних слања захтева од стране клијента ка серверу[1]. Пример таквог напада састоји се од слања великог броја мејлова (унутар мреже електронске поште) који садрже небитне информације, рекламе или злонамерне програме. Последица оваквог напада је велико успорење електронске мреже, а потом и њено обарање.

Решење се састоји у томе да сервер клијенту постави проблем који клијент треба да реши. Резултат се потом шаље серверу на проверу. Сâм проблем не треба да буде превише комплексан за решавање, већ за циљ има само да исцрпи компјутерско или корисничко време, али не у великој мери. Управо такав систем спречава злонамерног корисника да масовно шаље захтеве, јер сваки захтев мора да буде пропраћен решењем проблема који је сервер поставио. Са друге стране, верификација решења проблема на серверу је тривијална.[2]

У оквиру свог захтева за доказ о раду, Биткоин чворовима који учествују у формирању блока („рудари", тачка 4.2) представља проблем парцијалне инверзије хеш вредности (енг. partial hash inversion). Проблем се састоји у проналаску такве варијанте садржаја блока да хеш вредност тог садржаја, посматрана као 256-битни број, буде мања од претходно задатог броја у ознаци $nBits$. У специјалном случају $nBits = 2^{255-D}$ захтев постаје да хеш вредност блока почиње са бројем нулâ већим или једнаким од претходно дефинисаног броја D (тачка 6.1). Овако дефинисани бројеви контролишу тежину формирања блока. Практично, проблем се своди на проналажење (тачније погађање) псеудо-случајног броја под називом *нонс* (енг.

[1]Овакав тип напада се назива *DOS* (Denial-of-Service attacks).

[2]Пример оваквог система је *CAPTCHA*, `http://www.captcha.net/`.

nonce, number only used once) који, у комбинацији са осталим подацима (тачка 4.3) у блоку, задовољава наведени услов за хеш вредност. Верификација у овом случају јесте тривијална и своди се на израчунавање једне хеш вредности садржаја блока добијеног од чвора, што је услов да се блок прихвати као наставак ланца. Разлика у односу на класичне системе доказа о раду је одсуство сервера, тачније централног система који врши проверу решења. У овом случају верификација се обавља дистрибуирано; други чворови мреже учествују у верификацији управо кроз проверу да ли вредност хеш функције задовољава дати услов. Сâм проналазак псеудослучајног броја, односно решења проблема, може да буде компјутерски и временски веома захтеван.

4.2 Чворови рудари

Чворови који учествују у уградњи нових, непотврђених транскација у ланац обједињујући их у блокове, називају се **рудари** (енг. miners). Назив је мотивисан чињеницом да се за решавање проблема доказа о раду (објашњеног у тачки 4.1) и уланчавање новог блока у постојећи ланац, они награђују новим биткоинима (од 11. маја 2020. године, та награда износи 6.25 BTC, тачка 7.1.1), који нису претходно постојали унутар мреже (као да су их они „ископали" и открили осталим чворовима). Чворови се награђују и сумом свих провизија од трансакција (тачка 4.2.1) које се укључују у блок. Рудари се такмиче између себе ко ће пре да реши задати проблем[3] и буде награђен, па улажу у своје машине за формирање блока и електричну енергију, не би ли из тог такмичења изашли као победници. У почетку, довољно је било користити кућни рачунар за решавање проблема у вези са доказом о раду, а награда је била 50BTC. Са порастом броја чворова у мрежи повећао се и број конкурената, па су корисници улагали у графичке картице које су већ биле оптимизоване тако да врше бржа израчунавања од процесора. Данас се најчешће користи специјално дизајниран хардвер за формирање блокова који постиже брзину од 14.0 TH/s (тера хеш у секунди), али и који изискују пуно електричне енергије (више речи о овоме биће у оквиру главе 6). Рудари се због тога често организују у групе (енг. mining pools) удружујући своју опрему за хеширање и у зависности од свог удела у раду групе, бивају одговарајуће награђени.

Са порастом чворова и величине улагања у опрему повећава се и брзина формирања нових блокова, али и тежина проналаска псеудо-случајног броја. На сâму комплексност проблема може се лако утицати, једноставним мењањем броја $nBits$ односно најмање потребног броја нулâ потребних на почетку хеш вредности блока. Комплексност израчунавања мења се након сваких 2016 нових блокова (тачка 7.1.1), где се посматра за које време су они пронађени. Уколико је просечно време између два нова блока[4] 10 минута, комплексност остаје иста. Уколико је то време мање, комплексност се повећава. Ако је време рачунања блокова веће, комплексност се смањује[5]. Како се повећава број нових биткоина, награда рударима се смањује. Просечно, на сваке четири године та награда се полови. Последица оваквог решења је да је укупан број биткоина 21 000 000, па се не може десити класична инфлација, односно губитак вредности новца када његова количина у оптицају неконтролисано порасте. Досадашњи број непронађених биткоина који постоје као награда будућим рударима да их „ископају" је 2 560 000[6].

[3]Проблем могу да реше два различита рудара у приближно исто време. Разрешење ове ситуације дато је у тачки 4.3.1.

[4]Сваки блок садржи временски печат.

[5]`https://en.bitcoin.it/wiki/Difficulty`

[6]Подаци су преузети 24.07.2020. године са сајта `https://www.blockchain.com/charts/`

4.2.1 Провизија на трансакцију

Поред тога што учествује у формирању нових блокова, рудар је и регуларан чвор у мрежи и учествује у провери исправности новокреираних трансакција. Рудар користи своју копију базе непотврђених трансакција и одатле бира трансакције које улазе у блок. Како је величина блока ограничена, они не могу да укључе све трансакције у један блок, већ најчешће бирају оне које ће да им донесу највећи профит преко **провизије на трансакцију** (енг. transaction fee). Провизија се наплаћује увек иницијатору трансакције кроз додатни излаз из ње. Особа која креира трансакцију може посредно да утиче на то колико ће брзо трансакција бити укључена у следећи блок дефинишући вредност провизију трансакције. Са повећањем провизије, повећава се и шанса да ће следећи чвор рудар укључити баш његову трансакцију у блок. Уколико особа не дефинише величину провизије, новчаник (преко ког је особа иницирала трансакцију) предложиће јој коју вредност за провизију да постави. Предложена вредност варира у зависности од тога колика је заузетост мреже, односно колико постоји непотврђених трансакција у бази. Како је величина блока ограничена, трансакције које заузимају више простора имају мале шансе да буду укључене у блок уколико не постоји довољно[7] велика надокнада за њих. Величина трансакције варира у зависности од броја улаза и броја излаза трансакције. Додатну улогу у величини има и сложеност скрипти трансакције (тачка 3.1), нпр. трансакција са више потписа (тачка 3.3) сматра се сложеном због њене дужине.

4.3 Структура ланаца блокова

Ланац је структуиран тако да садржи све блокове од почетка функционисања мреже Биткоин, укључујући први блок који је 2009. године убацио Сатоши Накамото. То је почетни блок (тачка 7.1.2). Последњи убачен блок у ланац назива се глава ланца (енг. blockchain head). Укупан број блокова у ланцу, од почетног па све до главе, назива се висина блока (енг. block height). Веза између блокова је таква да се за рачунање хеш вредности новог блока (поред осталих параметара који ће бити објашњени касније) користи хеш вредност тренутне главе ланца на коју се нови блок надовезује. Након надовезивања нови блок је тренутна глава ланца, а блок на који се надовезао назива се његовим родитељем (енг. parent block). Један блок садржи трансакције које је рудар укључио из базе непотврђених трансакција. Блок карактерише и његово заглавље које се састоји од следећих елемената:

- верзија; верзија Биткоина под којом је блок формиран,
- хеш вредност претходног блока; хеш вредност родитељског блока, односно блока на који се формирани блок надовезује,
- корен Мерклеовог стабла (објашњеног у оквиру тачке 4.3.2); јединствена ознака свих трансакција које блок садржи,
- временски печат; тренутак формирања блока израженог као UNIX време (тачка 7.1.2),
- *nBits*; тренутна тежина формирања блока,
- *нонс*; псеудо-случајан број.

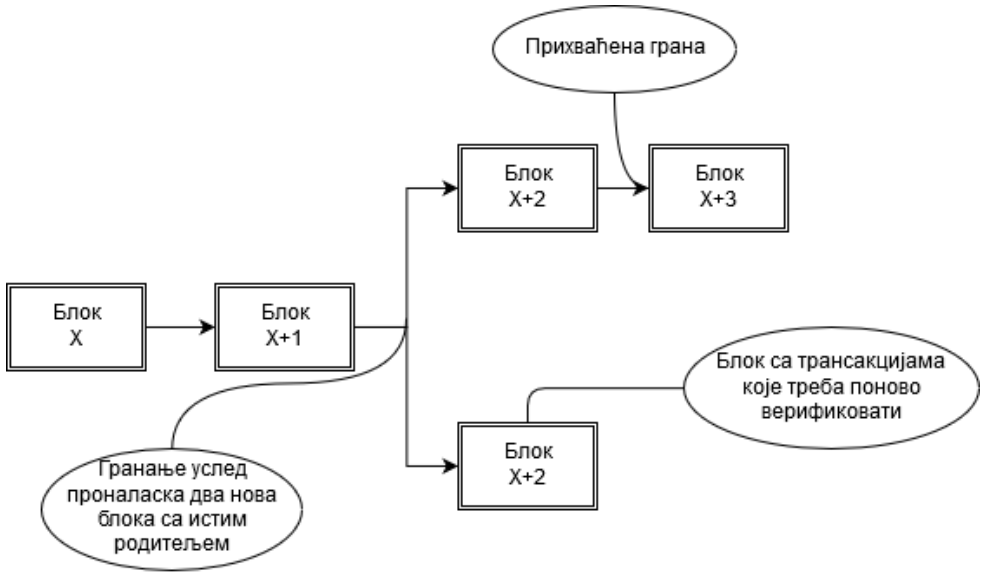
`total-bitcoins`.

[7]Постоји низ сајтова који пружају информацију о томе шта значи довољно велика провизија у зависности од величине трансакције.

Хеш вредност блока одговара вредности која се добија рачунањем SHA256^2 од податка који се добија конкатенацијом података из заглавља блока и та вредност је идентификатор блока у ланцу. Блок садржи и базну трансакцију (енг. coinbase), поред осталих које је рудар претходно укључио из базе непотврђених трансакција. Карактеристично за базну трансакцију је што њен улаз не садржи потпис, пошто нема ниједан излаз претходне трансакције на који се базна односи (поље <sig> унутар скрипте <scriptSig> може да остане празно). Улога такве трансакције јесте да TxOut садржи укупну награду рудару за формирање блока (награда за формирање и сума провизија на трансакције). Додатно, базну трансакцију рудар може да искористи за сигнализирање да је прихватио неки од BIP-ова (тачка 4.4).

4.3.1 Гранање ланца

У случају да два различита регуларна блока буду пронађена у приближно исто време, односно да се унутар мреже пропагирају два блока са истим родитељем, долази до гранања (рачвања) ланца[8]. Тако формирана бочна грана ланца може да потраје неколико блокова, али то најчешће није случај, јер је и само гранање веома ретка ситуација, па је дужина паралелних грана ланца најчешће у дужини једног блока. Оваква ситуација решава се на следећи начин: чим једна од грана постане дужа, односно има већи број настављених блокова, она се сматра прихваћеном, а друга грана се одбацује. Одабир гране практично се своди на ону у оквиру које је уложено више хеширања (и електричне енергије). Трансакције које су биле укључене у блокове бочне гране враћају се у базу непотврђених трансакција. Неке од тих трансакција могу бити део већ верификованих у оквиру прихваћене гране, па се те трансакције одбацују, јер су једном већ верификоване. Овај тип гранања назива се привременим гранањем (енг. Temporary fork). Видети илустрацију на слици 4.1.



Слика 4.1: Гранање ланца

Поред привременог, постоји меко гранање (енг. soft fork) и тврдо гранање (енг. hard fork). Меко гранање настаје у тренутку измене протокола, при чему је очувана компатибилност са претходним верзијама протокола. Било да се блок формира са старом или новом верзијом протокола, он се и даље сматра исправним. Овакав

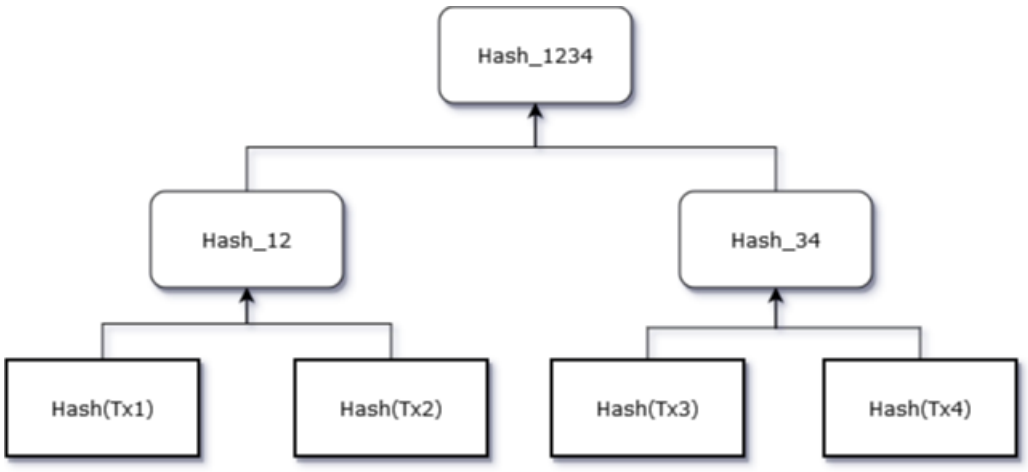
[8]„Приближно исто време" варира у зависности од загушења мреже. До рудара који пропагира свој блок није дошла још информација о постојању другог блока са истим родитељем у односу на онај који је он формирао.

тип гранања настаје када је сврха промене повећање сигурности или нека оптимизација. Код тврдог гранања блок, формиран старијом верзијом, не би се сматрао исправним, односно измена није компатибилна са старијим верзијама протокола. Овај тип гранања јавља се када су измене у протоколу драстичније, на пример измена величине блока у ланцу или креирање система Алткоин на основу система имплементације протокола Биткоин. Формирање блокова унутар система Биткоин као део бочне гране нема смисла, јер ће блок бити одбачен као неисправан. Коришћење новчаника са трансакцијама на бочној грани након меког гранања није препоручљиво (и не практикује се да не би долазило до поделе мреже), али је доступно. Чворови који тренутно нису на активној грани након меког гранања, чине око 1% чворова у мрежи[9] и то су чворови који нису прихватили увођење система SegWit (тачка 4.6.2). Иако, технички, тренутни ланац блокова јесте разгранат, то не утиче ни на који начин на функционисање мреже јер је меко гранање компатибилно са верзијама уназад (тачка 7.1.1).

4.3.2 Мерклеово стабло

Као што је већ речено сваки блок има своју хеш вредност која је израчуната на основу заглавља блока. Организација хеш вредности трансакција унутар блока реализована је кроз бинарно, Мерклеово стабло.

У листовима стабла налазе се хеш вредности појединачних трансакција укључујући и базну. У сваком унутрашњем чвору стабла налази се хеш вредност конкатенације садржаја његових синова. Последња израчуната хеш вредност на основу два унутрашња чвора је Мерклеов корен. Оваква организација битно убрзава проверу да ли нека трансакција припада блоку (тачка 5.8), јер се проверавају само хеш вредности на путу од листа (хеш вредности трансакције) до корена. Пример оваквог стабла унутар блока у који су укључене трансакције Tx1, Tx2, Tx3 и Tx4, дат је на слици 4.2. Чвор Hash_1234 је Мерклеов корен.



Слика 4.2: Мерклеово стабло

4.4 Улога рудара у побољшању система Биткоин

Биткоин, као децентрализован систем који није под контролом ниједног конкретног ентитета, није имао стандардизован начин унапређења свог функциони-

[9]Подаци преузети 24.07.2020. године са сајта `https://luke.dashjr.org/programs/bitcoin/files/charts/software.html`

сања. Амир Таки (Amir Taaki) предложио је решење за овај проблем 2011. године [15], увођењем техничких докумената под називом BIP (Bitcoin improvment proposals)[10] која описују предлог о могућем побољшању делова система. Та документа треба да буду доступна да може да их разматра цела Биткоин заједница укључујући софтверске инжењере који имплементирају измене протокола, рударе и крајње кориснике. Постоје различити типови таквих докумената у зависности од тога на који аспект система је побољшање фокусирано. Документи који утичу на стандард (енг. **Standards Track BIPs**) фокусирани су на измене у оквиру протокола мреже и протокола провере блокова. Информативни документи, (енг. **Informational BIPs**) су документи који пружају смернице о томе како је најбоље користити систем и кроз које може да се укаже на потенцијалне проблеме, не предлажући решења. Процесни документи (енг. **Process BIPs**) садрже предлоге о измени процеса функционисања система који могу да одступају од постојећих процеса. Група програмера која ради на одржавању система Биткоин узима у разматрање документа и мења кôд по угледу на документ који су прихватили. Рудари потом сигнализирају да ли су прихватили измену додајући стринг уместо дела <sig> базне трансакције, чији је <scriptSig> свакако празан. Разлог зашто је битно да рудари искажу своје мишљење о промени кôда на овај начин је то што уколико више од половине рудара након извесног времена[11] не прихвати измену, може се размотрити повлачење промене из протокола. Пример измене коју рудари нису у почетку прихватили је документ BIP141 (тачка 4.6.2), али измена није повучена и данас се примењује.

4.5 Напади на мрежу

У овој тачки ће бити речи о различитим потенцијалним нападима на систем и на који начин је сâм систем конструисан да би се од њих заштитио.

4.5.1 Дупло трошење

Напад дуплог трошења један је од једноставних напада који је систем Биткоин у самом почетку свог постојања морао да реши. Напад се састоји у трошењу истих биткоина у оквиру две или више различитих трансакција. Како је систем Биткоин децентрализован, односно не постоји ентитет који контролише стање биткоина особе која покушава да изврши дупло трошење, проблем је једноставно решен применом правила о најдужем ланцу (објашњеном у оквиру тачке 4.3.1).

Финијев напад

Варијанту напада дуплог трошења представио је Хал Фини[12] (Hal Finney, по ком је напад и добио име), где нападач тајно рудари блок, укључујући у њега и своју трансакцију у којој шаље биткоине самом себи. Након формирања блока, он започиње регуларну трансакцију са другом особом (продавцем) у којој користи исте биткоине и за њих захтева неку робу. Продавац чека неки период времена и

[10]Целокупан преглед досадашњих предлога може се наћи на локацији `https://github.com/bitcoin/bips/blob/master/README.mediawiki`, од којих је први управо рад Амира Такија под ознаком BIP1. Постоји још неколико круцијалних радова као што су BIP91, BIP141, BIP148, итд.

[11]Време се изражава у броју формираних блокова од тренутка када је неки технички документ дат у разматрање. Број блокова варира у завиности од озбиљности измене у протоколу.

[12]Унутар дискусије о ризицима брзог прихватања транскација, Фини је представио и своју идеју о нападу, `https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384`

примећује да у мрежи не постоји друга трансакција која би указала на то да су биткоини које нападач користи за куповину већ коришћени у некој другој трансакцији и шаље робу. Због тога, продавац може да се одлучи да пошаље робу пре формирања блока који садржи трансакцију коју обавља са нападачем. Након тога нападач објављује свој тајно формиран блок и тада се у мрежи налазе две трансакције које троше исте биткоине.

Проблем са овим нападом је што особа којој се шаљу биткоини може да чека да се ланац повећа (у односу на тренутак када је нападач иницирао трансакцију са продавцем) за неколико блокова (тај број је обично 6) и тако буде сигуран да није дошло до дуплог трошења биткоина. Нападач је протрaћио и време и ресурсе рударећи блок, а регуларна трансакција коју обавља са продавцем остаје као неповратан део мреже, што је исправно.

4.5.2 Напад 51%

Једини напад који има шансе да успе је „напад 51%", односно напад познат као „већински напад" (енг. majority attack). Уколико би нападач имао у поседу машине са брзином израчунавања хеш вредности већом од половине целокупне мреже Биткоин (строго већим од 50%, одакле потиче и назив), могао би да је контролише. У том случају, он би могао да утиче на победника приликом гранања ланца, јер са машинама у поседу, вероватноћа да он формира следећи блок је строго већа од 1/2 и на тај начин утиче на креирање најдужег ланца који би по дефиницији протокола био прихваћен. Управо то као последицу има да би могао лако да изведе напад дуплог трошења искључујући друге чворове рударе из мреже и у потпуности преузимајући све нове трансакције на процену њихове исправности.

Проблем са овим нападом је у његовој цени. Нереално је да једна особа може да изведе овај напад, с обзиром на то да мора да буде у поседу мноштво машина са високом брзином израчунавања криптографске хеш вредности. Цена овог напада процењује се у милијардама долара, зато је вероватније да напад организује нека држава. Како временом мрежа постаје већа, она је и стабилнија и нападач би морао да је прати улагањем да би могао да одржи већинско учешће у машинама. Додатни проблем је што у надгледању трансакција и нових блокова учествује цела мрежа, као и инжењери који раде на протоколима мреже Биткоин.

Овакав тип напада никада није изведен од када постоји мрежа Биткоин, али је 2014. године једна група рудара под називом Ghash.io[13], уједињено дошла до високог нивоа брзине израчунавања хеша и били су учесници у 42% новокреираних блокова. Страхујући да би онда били у могућности заједнички да изведу напад 51%, наишли су на притисак биткоин заједнице да напусте ту групу, што је велики број рудара и урадио.

4.5.3 Напад са загушењем мреже трансакцијама

Упркос доказу о раду, напад који се састоји од масовног слања захтева ка мрежи је и даље могућ. Злонамерна особа може да га реализује обављањем мноштва трансакција са собом, пошто би га трансакција са неким другим водила у банкрот због постојања најмање вредности са којом се може обавити трансакција и она износи 546 сатошија (0.0000546BTC).

Приликом реализације овог напада проблем су провизије за трансакције. Да би нападач повећао шансе да трансакције којима покушава да загуши мрежу уђу у следеће блокове, он мора да понуди већу провизију за трансакцију. Тиме овај

[13] `https://en.wikipedia.org/wiki/Ghash.io`

напад после неког времена постаје неисплатив. Загушење мреже може да изазове злонамерни, али и немарни корисник.

4.5.4 Себично формирање блокова

Напад који није усмерен на сâму мрежу, већ на рударе у њој, где злонамерни чвор рудар чешће побеђује у трци формирања новог блока и одатле добија више награда, назива се **себично формирање блокова**. Напад, који се може третирати и као стратегија формирања блокова, описали су Итај Ејла (Ittay Eyal) и Емин Ган Сирера (Emin Gün Sirer) 2013. године у раду [6].

Стратегија користи протокол о гранању и чињеницу да рудар прати најдужи ланац као родитељ блока који он треба да формира. Злонамерни рудар може свој блок да формира у тајности са везом на тренутно активну главу ланца. Злонамерни чвор наставља формирање блокова надовезујући се на своје блокове и за то време прати ланац и његову дужину. У тренутку када се дужина ланца, од момента када је започео формирање блока у тајности, приближи дужини његове приватне гране, он је објављује на мрежу и из угла мреже то је само још једно привремено гранање које се десило. Грана коју је нападач објавио мора да буде дужа од бочне гране која се развијала, док је нападач тајно формирао блокове како би напад био успешан. Остали чворови прате правило о најдужој грани и надовезују се на грану злонамерног рудара. За разлику од напада 51% (тачка 4.5.2) који захтева надмоћ од половине мреже у брзини израчунавња хеш вредности, овај напад је, према рачуну која је изведена у раду [6], изводљив и са 25% укупне брзине израчунавања хеш вредности мреже. Оно што овај напад чини опасним по остале рударе је то што је њихов рад узалудан и бивају ускраћени за награду за блокове које су претходно формирали.

Напад може да се детектује користећи временски печат како је објашњено у раду [8]. Између два регуларно формирана блока, постоји временска разлика која одговара времену од тренутка за које се први блок пропагирао кроз мрежу, до тренутка када је формиран нови блок (укључујући и време за које је формиран). У случају себичног формирања блокова, временска разлика између два тајно формирана блока, биће мања, односно одговараће само времену за које је потребно да се блок формира.

4.5.5 Илегалан садржај унутар ланца

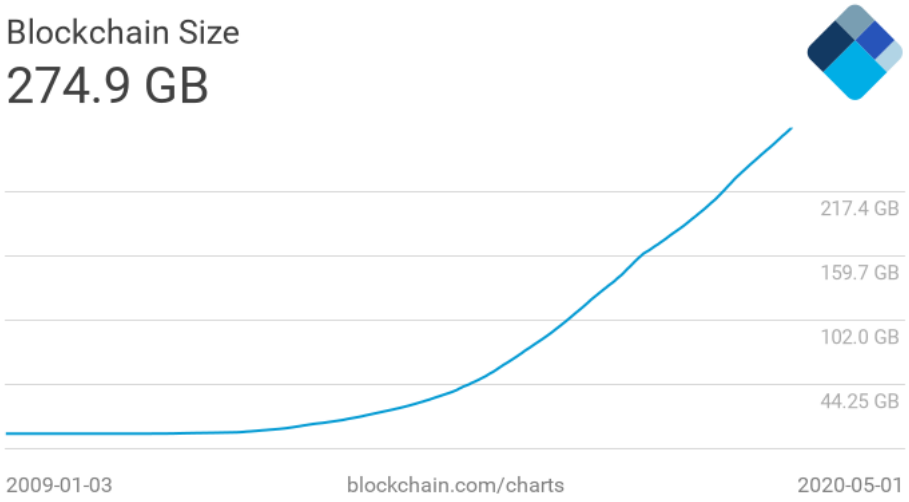
Чињеницу да сваки чвор има своју копију ланца, може да искористи злонамеран корисник тиме што ће унутар формираног блока да заувек укључи садржај који може да буде незаконит, као што су: књиге које нису дозвољене у појединим државама[14], садржај који је под ауторским правима, садржај подложан цензури или неки други. Овај тип података може да се нађе као део јавне адресе трансакције, заглавља блока (на пример у оквиру Мерклеовог корена, тачка 4.3.2) или као део базне трансакције (тачка 4.3).

Мана овог напада је у томе што може лако да се детектује, јер у ком год делу блока да је смештен, он атипично одскаче од њега у односу на регуларне блокове. Међутим, уколико не долази ни до какве провере блока који стигне до чвора, он може проћи као неоткривен. Циљ напада је наношење штете свим чворовима у мрежи, али не у материјалном смислу какав је напад „себично формирање блокова" (тачка 4.5.4)

[14] `https://en.wikipedia.org/wiki/List_of_books_banned_by_governments`

4.6 Слабости ланца блокова

Ланац блокова представља иновативну технологију која доноси велики ниво сигурности криптовалута које се на њему заснивају, као што су биткоин, итиријум (криптовалута споменута у уводу) итд. Идеја на основу које је настао јесте да сваки чвор има своју копију ланца чиме се постиже децентрализација. Унутар система Биткоин, ланац се проширује тако што рудари решавају проблем дат кроз доказ о раду. Систем Итиријум, са друге стране, свој ланац проширује користећи технологију која се заснива на доказу о улогу (тачка 4.6.1) и назива се *Casper*[15]. Проблем унутар обе криптовалуте је што овај ланац константно расте. Расту додатно доприноси и повећање броја чворова у мрежи, а самим тим већи број трансакција које се кроз мрежу обављају. Промена количине меморије коју биткоин ланац захтева (и раст величине ланца), почевши од његовог настанка, приказан је на слици 4.3.



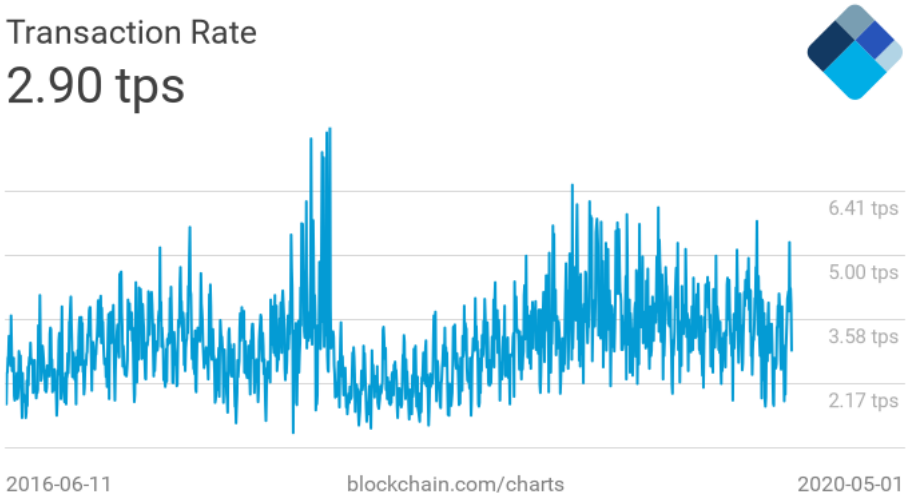
Слика 4.3: Дужина ланца

Због величине целокупног ланца непрактично је за његову обраду користити мобилни телефон. Додатни проблем је време чекања да се нека трансакција изврши, јер време које је потребно да трансакција буде убачена у блок, није мало. То време подразумева проверу исправности трансакције, формирање блока у који ће бити убачена и евентуално чекање на проширење дужине ланца у односу на тренутак започињања трансакције како би се избегао Финијев напад (тачка 4.5.1). На слици 4.4 приказано је кретање броја трансакција у секунди у последњих неколико година. Као што се види, број трансакција у секунди временом се није пуно мењао и данас (мај, 2020.године) просечно се обрађују 3 трансакције у секунди. Ова стопа обраде драстично је мања у односу на плаћања која се обављају помоћу VISA картице, где је број трансакција по секунди 1700[16].

4.6.1 Потрошња електричне енергије

У систему Биткоин принцип по ком се формирају нови блокови састоји се од решавања проблема у вези са доказом о раду постављеним (тачка 4.1). Међутим, то изискује улагања у машине које имају велику брзину израчунавања хеш вредности, у циљу побеђивања у трци за наградом коју, тако формиран блок, доноси.

[15] https://github.com/ethereum/wiki/wiki/Casper-Proof-of-Stake-compendium
[16] https://en.bitcoin.it/wiki/Scalability



Слика 4.4: Стопа трансакција по секунди

Проблем је што те машине имају велику потрошњу електричне енергије. Наиме, истраживања показују да формирање нових блокова у чворовима у мрежи, захтева потрошњу електричне енергије приближно једнаку потрошњи неких држава. На слици 4.5 дат је један пример односа. Подаци[17] показују да је потрошња електричне енергије за само једну трансакцију приближно једнака 700 kWh. Потрошња електричне енергије за обраду 100 000 VISA трансакција приближно износи 180 kWh. Све ово јасно показује да је штетност формирања блока за нашу планету велика, што је довољан разлог да поједине државе не признају биткоин као валуту (поред политичких).
Решење које се фокусира на главни разлог оваквих статистика је замена технологије „доказ о раду" технологијом „доказ о улогу" какву користе неке од алткоина (попут итиријума).

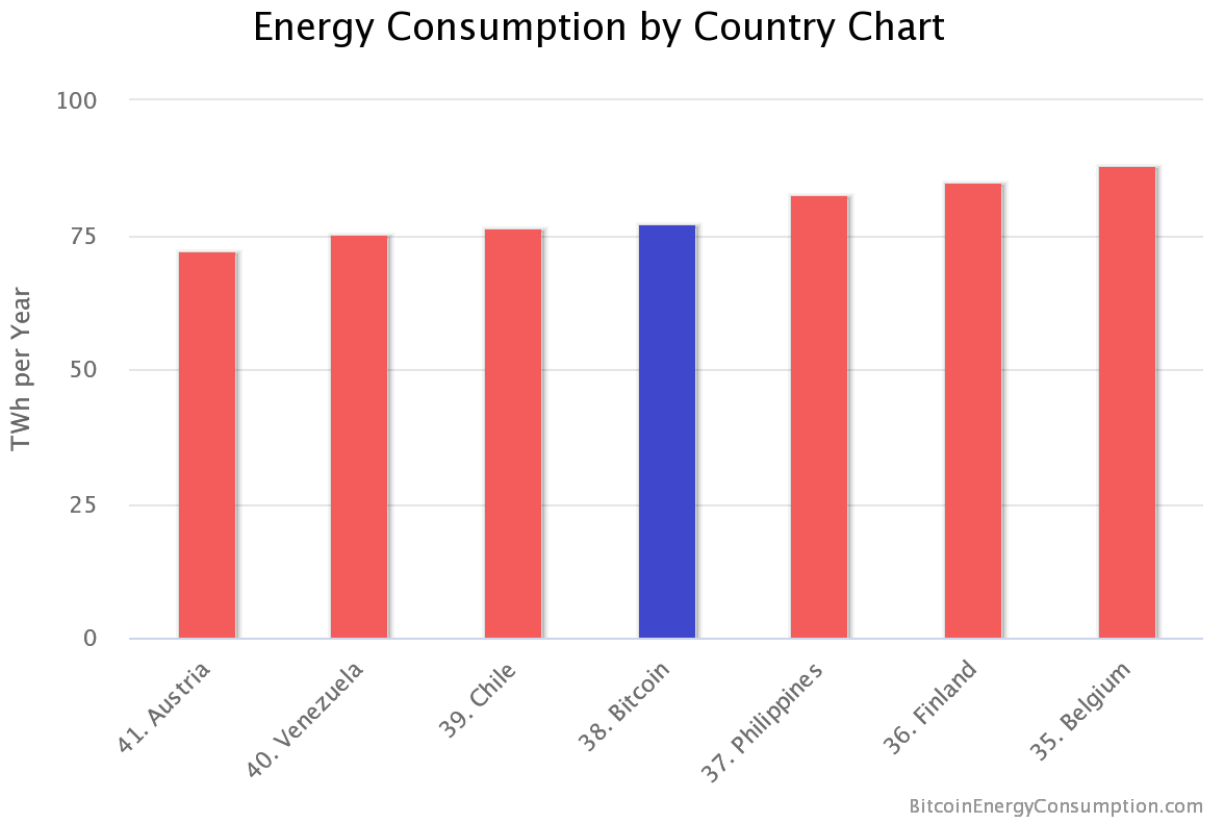
Доказ о улогу

Доказ о улогу представља алтернативу доказа о раду где чворови, уместо да се такмиче између себе ко ће да дода нови блок у ланац, буду изабрани од стране мреже да тај блок створе[18]. Сваки чвор, чија је улога да ствара нове блокове, уложио је одређену своту новца и количина тог новца одређује његову шансу да буде изабран. При томе, што више чвор уложи, веће су шансе да буде изабран. Наизглед делује неправично, јер богатији чворови имају веће шансе да буду изабрани. Међутим, уколико се испостави да је створени блок неисправан, чвор - који је учествовао у његовом стварању, „кажњава" се одузимањем дела улога који је поставио. Уколико чвор креира исправан блок, он је награђен провизијом на трансакције и враћа му се новац постављен као улог.

Осврнућемо се на напад који је теоријски могућ, а може да нанесе највећу штету систему, 51% (тачка 4.5.2). У случају доказа о раду, нападач (реалније, група нападача) мора да има у свом власништву машине које имају строго већу брзину израчунавања хеш вредности од половине мреже. У случају доказа о улогу, нападач мора да уложи количину новца строго већу од половине укупног улога у мрежи,

[17]Прегледани на сајту `https://digiconomist.net/bitcoin-energy-consumption/` у мају 2020. године.

[18]Намерно није коришћен израз *формирају*, јер постоји разлика у терминологији у „доказу о улогу" у односу на ону коришћену унутар „доказа о раду".



Слика 4.5: Однос потрошње струје за формирања блокова и потрошње неких држава

чиме је овај напад још више отежан, али није нереалан. Додатна предност доказа о улогу, утиче да систем остане децентрализован, што није случај унутар технологије доказа о раду, где велики број удружених рудара у једну рударску групу могу да учине систем привидно централизованим.

4.6.2 Побољшања скалабилности

Као што је већ објашњено мрежа Биткоин је спора. Број различитих трансакција унутар блока је мали, а њихова пропагација кроз мрежу траје дуго (тачка 4.6). У овој тачки ће бити објашњена два различита решења са међусобно другачијим приступима за овај проблем. Остала решења могу се наћи у оквиру рада [17].

Одвојени потписи

Горња граница величине блока у ланцу износи свега 1MB што битно ограничава број трансакција у блоку. Промена величине блока изменом кôда имплементације протокола проузроковала би тврдо гранање (које не би било компатибилно са старијим верзијама протокола, тачка 4.3.1), што је непрактично, јер може доћи до поделе чворова у мрежи, а то програмери Биткоина желе да избегну. Додатни разлог је то што се повећање максимума величине блока сматра привременим решењем. Нагло повећање блока унутар система продужило би његово време пропагације кроз мрежу. Временом, како би се повећао број чворова у мрежи, повећао би се и број нових трансакција, чиме би се поново јавила потреба за повећањем

величине блока[19], па се ово решење не сматра дугорочним. Ерик Ломброзо (Eric Lombrozo), Џонсон Лау (Johnson Lau) и Питер Вили (Pieter Wuille) 2015. године представили су кроз BIP141 [16] решење за повећање величине блока извођењем меког гранања.

Структура сваке трансакције своди се на улазни, излазни део и потписе. Унутар трансакције укључене су и скрипте за њену верификацију (тачка 3.1) и оне, заједно са потписима, заузимају највећи њен део. Решење „одвојени потписи"[20] (енг. Segregated Witness, SegWit) базира се на новој структури блока која се састоји од: основног дела - садржи информацију о пошиљаоцу и примаоцу и величине је 1MB (што одговара величини блока унутар старије верзије протокола); дела сведока - садржи све потписе и скрипте. Оваква структура блока помера његову границу максималне величине на 4MB што омогућује већи број трансакција унутар њега. Оно што ово решење чини компатибилним са старијом верзијом протокола је то што чворови, који нису ажурирани на новију верзију, прегледају само основни део блока као и пре, где је <scriptPubKey> празан, а <scriptSig> садржи OP_TRUE. Чвору то делује као трансакција чији је тип хеша SIGHASH_NONE (тачка 3.1) и пропагира га даље као регуларног. Када блок нове структуре дође до чворова са новијом верзијом протокола, они прегледају и основни и надограђени део блока.

Брза мрежа

Приступ са смањеним бројем трансакција, чиме се добија смањење укупне дужине ланца, представљен је под називом **брза мрежа** (енг. Lightning Network) [5]. Брза мрежа је додатни слој поред постојећег система Биткоин, који је одвојен од централног ланца блокова. Идеја је да се главни ланац не затрпава мини-трансакцијама, односно трансакцијама које се свакодневно обављају, већ да се оне директно извршавају између чворова.

Реализација је преко канала плаћања (енг. payment channel), који се успоставља између два чвора брзе мреже и кроз њега се обављају све мини-трансакције. Како се мини-трансакције искључују са главног ланца, смањује се број трансакција који треба да обраде остали чворови главног ланца. Пре отварања канала плаћања, два чвора међусобно остварују трансакцију, где сваки од чворова улаже одређену своту биткоина као депозит. Трансакција је типа 2 од 2 кључа (објашњена у тачки 3.3) и она се укључује у главни ланац. Укупна вредност депозита једне и друге стране представља максимални износ који може да се, у једном тренутку, нађе у оквиру трансакција брзе мреже, тј. капацитет канала плаћања[21]. Трансакција која се обавља унутар брзе мреже назива се „обавезајућа трансакција" (енг. commitment transaction). Баланс између два чвора која обављају трансакцију мора да остане у границама депозита. Након завршетка последње мини-трансакције, канал плаћања између два чвора може да се затвори. Протокол налаже да се направи нова, закључна трансакција са укупним билансом стања биткоина (и једног и другог чвора) која замењује све мини-трансакције кроз канал и која се укључује у блок кога неки чворрудар за главни ланац блокова. На тај начин две трансакције укључене у главни ланац замењују велики број других, регуларних трансакција ван њега.

[19]Са овим се није сложио део Биткоин заједнице у коме се сматрало да ће се проблем скалабилности умногоме ублажити повећањем максималне величине блока на 8MB (чиме се добија 8 пута више од тадашњег броја трансакција унутар блока). Ово је проузроковало тврдо гранање ланца и настанак нове алткоин валуте под називом *Bitcoin-Cash*, која и данас постоји.

[20]Дигитални потписи се често називају сведоци трансакција.

[21]Додатна улога депозита јесте додатни ниво заштите. Уколико се детектује понашање једног од чворова које је ван протокола, целокупан износ његовог деопозита пребацује се другом чвору.

Додатна предност брзе мреже је начин на који су чворови унутар ње повезани. Канал плаћања не мора нужно да постоји између свака два чвора у мрежи. Уколико би један чвор желео да изврши трансакције са другим, оне се могу обављати кроз заједнички чвор са којима оба чвора већ имају успостављен канал плаћања.

5 Новчаници

Биткоин новчаник је термин који се користи за програм који има улогу чувања и праћења стања корисничких биткоина, слања биткоина ка адресама и праћења статуса трансакција. Новчаник се користи за генерисање приватног кључа и њему одговарајућих јавних кључева (поступком датим у тачки 2.5.2) као и њима одговарајућим Биткоин адресама (поступком датим у тачки 2.7). Адреса везана за новчаник не може да постане неактивна, односно биткоини су трајно везани за неку адресу (и за приватни кључ). Неопходно је да новчаник генерише приватни кључ са високим степеном ентропије (на привидно случајни начин), јер је од великог значаја да се тешко дође у посед приватног кључа. Свако ко зна приватни кључ може да контролише биткоине новчаника коме је тај кључ придружен. Због тога је битно да се праве копије новчаника које имају експортоване приватне кључеве везане за њега, за случај, на пример, пада система рачунара на ком је новчаник инсталиран.

Новчаник који садржи целокупну копију ланца блокова назива се „потпуни чвор" (енг. full node), док су са друге стране новчаници који чувају само део ланца „лаки чворови" (енг. light node) (тачка 5.8). Новчаници који су типа потпуног чвора независно од других новчаника могу да провере исправност нове трансакције.

5.1 Протоколи унутар система Биткоин

Улога новчаника је и у пропагирању креиране трансакције ка мрежи како би остали чворови учествовали у провери исправности трансакције. Протокол по ком се трансакције пропагирају назива се „протокол оговарања" (енг. gossip protocol). Један новчаник повезује се са 8 насумично изабраних других чворова у мрежи и њима пропагира трансакцију на проверу исправности. Са друге стране новчаник прихвата на проверу трансакције највише 100 других новчаника, а потом их смешта у базу непотврђених трансакција уколико су исправне. Након појављивања новог блока у мрежи, новчаници имају улогу додатног проверавања трансакција унутар тог блока. Разлог додатне провере је што се на тај начин искључује могућност да је сâм рудар покушао злонамерно да измени неку од трансакција у блоку.

Протокол по ком се пропагирају тек формирани блокови кроз мрежу веома великом брзином назива се „Влакно" (енг. Fast Internet Bitcoin Relay Engine, FIBRE)[1]. Потреба за имплементацијом овог протокола постоји из разлога што је од кључног значаја да нови блок што пре стигне до свих чворова широм света. Највећу корист примене протокола имају рудари, јер не троше узалудно своје ресурсе и формирају блок са трансакцијама које су већ убачене. Због брзе пропагације блокова привремена гранања (тачка 4.3.1) се ређе дешавају, пошто је рудар тачно информисан о блоку који је тренутна глава ланца на који се рудар надовезује са блоком који је формирао. Додатна предност је и у томе што чворови имају праву информацију о дужини ланца и због тога имају ажурирану вредност тежине формирања блока (тежина се ажурира на просечно 2016 нових блокова, тачка 7.1.1).

Постоји још велики број других протокола унутар система (поред поменута два) као што су: *straum*, *straumv2*, *getblocktemplate* итд.

[1] `https://bitcoinfibre.org/`

5.2 Хладно складиште

Хладно складиште (енг. cold storage) је међутим медијум који није директно повезан са интернетом, а на ком се чува приватни кључ (и евентуално њему одговарајући јавни кључ) везан за једну од адреса неког новчаника. Приватни и јавни кључ се генеришу на рачунару који није повезан са мрежом, али има инсталиран Биткоин новчаник. Заједничка мана оваквог чувања кључева је што такви медијуми могу временом да имају физичка оштећења која могу да утичу на сâм запис кључа, што за последицу има трајни губитак биткоина који су за тај кључ везани. Примери оваквих складишта кључева дат је у наставку.

5.2.1 Екстерно складиште

Приватни кључ може се записати унутар фајла, а касније тај фајл сачувати на неком од екстерних складишта као што су флеш-меморија или диск. Вредност приватног кључа касније се користи само за потписивање трансакција користећи рачунар на ком је инсталиран новчаник који има везу са интернетом. Тренутак потписивања трансакције индиректно повезује приватни кључ са интернетом, што је и главна мана коришћења овог типа складиштења кључа. Из тог разлога се фајл, у ком је сачуван приватни кључ, додатно шифрује.

5.2.2 Папирни медијум

Приватни и јавни кључ су одштампани на папир. Сама штампа треба да се обави рачунаром и штампачем који нису повезани на интернет. Рачунар има инсталирану дистрибуцију оперативног система који је оригинално купљен само за ову примену. Добра пракса је рестартовати рачунар након завршетка штампе. Модеран приступ при оваквом начину чувања кључева је да, поред одштампаног приватног и јавног кључа на папиру, постоје одштампана два различита QR кôда - један за приватни, а други за јавни кључ. Употреба QR кôда олакшава касније потписивање трансакција једноставним скенирањем QR кôда, али и пружа додатни ниво безбедности.

5.2.3 Физички новчићи

Имају сличну улогу као папирни медијум. Приватни кључ се чува на медијуму који има изглед новчића. QR кôдом је представљена адреса, а приватни кључ је утиснут на полеђини металног новчића. Пожељно је овакав тип медијума наручивати од познатих произвођача, као што су Денаријум (Denarium), Касаксиус (Casascius) итд. Међутим, ни тада ризик није отклоњен, с обзиром на то да је приватни кључ генерисало неко треће лице. Из тог разглога овакав тип медијума је пожељно користити за трансакције које захтевају више потписа (тачка 3.3), тачније 2 од 2 потписа. Један потпис се генерише приватним кључем са новчића, док се за други потпис користи приватни кључ који власник новчића већ поседује. Тако се отклања могућност неовлашћеног трошења биткоина од стране компаније која производи новчић.

5.3 Новчаник и хладно складиште

Пример коришћења новчаника и приватног кључа који се чува на неком физичком медијуму (хладном складишту) састоји се од система два уређаја, једног

повезаног на интернет и другог ван њега. Улога уређаја који је повезан на мрежу је да само иницира (али не и да потпише) и касније прати статус трансакције. То је остварив кроз новчаник који се зове „новчаник за посматрање мреже" (енг. watch only wallet). Назив потиче од чињенице да таква врста новчаника не учествује у потписивању нових трансакција, услед недостатка информације о приватном кључу. Улога уређаја који је ван мреже је да само дигитално потпише трансакције користећи приватан кључ чуван на неком од претходно описаних медијума хладног складишта. Предност оваког система је у томе што биткоини нису везани за рачунар са ког се шаљу. У случају упада на рачунар на ком је новчаник инсталиран (од стране злонамерне особе) не постоји могућност крађе биткоина. Слабост оваквог система је тренутак потписивања; трансакција се преноси на рачунар ван мреже, повезује се са медијумом хладног складишта, потписује се и враћа на рачунар повезан са мрежом, чиме се ствара индиректна веза приватног кључа са интернетом.

5.4 Физички новчаници

Уређаји, без директне повезаности са интернетом, посебно дизајнирани за чување приватног кључа уз пружање могућности потписивања трансакције помоћу њега, називају се „физички новчаници" (енг. hardware wallets). Повезани су са рачунаром који има инсталиран новчаник којим се креирају трансакције, а улога физичких новчаника је да додатно потврде трансакцију и тако је потпишу. Потврђивање трансакције се најчешће изводи тако што физички новчаници имају мали екран где власник има увид у основне податке о трансакцији коју потписује. Додатно физички новчаници најчешће имају два тастера како би потврдили или отказали тренутну трансакцију. Узимајући у обзир да је то физички уређај и лако може доћи до његовог губитка, крађе или оштећења, препоручује се да се приватни кључ (који уређај чува) ипак ископира и на неки други медијум.

5.5 Паметни новчаници

Пример једног приватног кључа записаног у хексадекадном формату је:
753D7B5468FB3CF849915C02109652BF31DFC9FA8C8DC29E0614C39566CEC818
Рогобатан изглед оваквог кључа добра је подлога за грешку приликом, на пример, записивања кључа на папир са намером да се сачува. Уколико дође до грешке приликом записивања, немогуће је вратити биткоине који су за тај приватни кључ везани. Улога новчаника под називом „паметни новчаник" (енг. brain wallets) је да приликом генерисања кључева, псеудо-случајни генератор буде иницијализован речима (енг. random seed) које је власник новчаника одабрао. У случају губитка приватног кључа, он може бити поново генерисан користећи лозинку састављену од речи које је власник новчаника претходно изабрао. Проблем са овим типом новчаника је што се корисници служе предвидивим речима и шифрама и тако постају лака мета злонамерних корисника. С обзиром на то да су данас напади који разбијају корисничке лозинке поприлично развијени (речнички напад [10]), не препоручује се употреба паметних новчаника.

5.6 Мрежни новчаници

Мрежни новчаници ослањају се на екстерног провајдера који генерише и чува вредност приватног кључа. Провајдер пружа интерфејс власнику приватног кључа којим може да пребацује новац на жељену адресу, прати статус трансакције итд. Премда најкомфорнији (корисник само прави налог на сајту), мрежни новчаници су најнесигурније решење које постоји за чување приватног кључа, јер се корисник ослања на провајдера који може бити и злонамеран. Чак није неопходно да провајдер буде злонамеран, већ је довољно да буде немаран у чувању приватних кључева и на тај начин угрози биткоине корисника. Проблем се може решити помоћу „хибридних мрежних новчаника" (енг. hybrid e-wallets). Разлика у односу на мрежне је у томе што корисницима пружају могућност чувања приватног кључа на свом рачунару, а провајдер касније захтева вредност приватног кључа корисника како би потписао трансакцију. Захтевање се изводи помоћу програма који комуницира између сајта и рачунара власника на ком се чува приватни кључ. Иако се приватни кључ никада не шаље провајдеру, он може злонамерно да измени програм и прочита оригиналну вредност приватног кључа.

5.7 Детерминистички новчаници

Детерминистички новчаници су они који могу да генеришу више парова кључева (приватних и јавних) на основу једног приватног кључа. Предност оваквог система лежи у просторно мањем и лакшем прављењу копије новчаника, јер је довољно познавати само један кључ у циљу приступа осталим који су генерисани помоћу детерминистичког новчаника.

5.7.1 Детерминистички новчаници типа 1

Детерминистички новчаник типа 1 адресе генерише на основу основне лозинке (енг. master password). Довољно је узети неку произвољну реч за шифру и на њу додавати бројач, који се инкрементира са генерисањем сваке нове адресе, као суфикс. На пример, за шифру „КомпликованаШифра" могуће је направити κ таквих које би гласиле „КомпликованаШифра*1*", „КомпликованаШифра*2*", ..., „КомпликованаШифраκ". Приватни и јавни кључ се коначно генеришу на следећи начин:

$$d = H(mp||l)$$
$$T = dG,$$

где вредност приватног кључа d одговара резултату хеширања криптографском хеш функцијом H. Функција H хешира вредност добијену надовезивањем мастер-шифре mp и бројача l ($0 \leq l \leq k$). Јавни кључ T добија се (на начин објашњен у тачки 2.5.3) множењем приватног кључа d и генератора групе G. Мана детирминистичких новчаника типа 1 иста је као и мана паметних новчаника (тачка 5.5); користе шифру коју је корисник осмислио, а она у данашње време све лакше може да се разбије.

5.7.2 Детерминистички новчаници типа 2

Оно што одликује детерминистичке новчанике типа 2 је могућност генерисања нових јавних кључева, а даље и Биткоин адреса, без познавања приватног кључа.

Практична примена ових новчаника је када више корисника контролише различите адресе, али без права да се даље користе биткоини који су за те адресе везани. Таква организација погодна је у ситуацији када имамо, на пример, шефа који у поседу има све приватне кључеве новчаника, а радници примају уплате и надгледају трансакције које се обављају ка њиховим адресама за које су задужени. Радник има могућност да направи нову Биткоин адресу у оквиру новчаника, али не и (без претходног посредства шефа) да троши биткоине са адресе. Математички приказ генерисања јавног кључа дат је у наставку:

$$d = mpk + H(mp||l)$$
$$T = dG,$$

где је mpk основни приватни кључ сефа, а mp је основна лозинка радника, а l је вредност бројача. Заменом приватног кључа d у израз за T добија се,

$$T = mpkG + H(mp||l) * G$$
$$= T_{mpk} + H(mp||l) * G.$$

Приватни кључ d дефинисан је на основу приватног кључа сефа mpk (који новчаник генерише као и било које друге приватне кључеве унутар мреже) и резултата криптографске хеш функције H примењене на конкатенацију $mp||l$ (тачка 5.7.1). Касније, јавни кључ би се у регуларном случају рачунао као $T = dG(\mathrm{mod}\ n)$, али мењањем приватног кључа d изразом којим је добијен, закључујемо да нам је за генерисање јавног кључа довољно да знамо мастер-јавни кључ T_{mpk} и мастер-шифру mp. Генератор G и ред елиптичке криве *secp256k1* већ су познати мрежи.

5.7.3 Хијерархијски детирминистички новчаници

Са детерминистичким новчаницима типа 2, један новчаник може да генерише произвољан број јавних кључева, самим тим и Биткоин адреса, али и да има увид у све трансакције које се обављају унутар њега, чиме је угрожена анонимност новчаника. То је превазиђено организацијом адреса новчаника у виду стабла, где се у корену налази мастер-приватни кључ (означен као mpk из претходне тачке) добијен као и регуларни приватни кључ елиптичке криве, а он за потомке има чворове који могу да буду приватни чвор или јавни чвор, у зависности од тога који кључ чувају. Јавни чворови у стаблу имају податак о јавном кључу и као потомке могу да имају само чворове типа јавни чвор. Потомака може бити највише 2^{31}. Приватни чворови у стаблу имају податак о приватном кључу и они као потомке имају приватни чвор или јавни чвор. Приватни чворови могу имати до 2^{31} различитих потомака који су типа приватни чвор. Поред тога могу имати и до 2^{31} различитих потомака типа јавни чвор. Примена овакве организације адреса илустрована је примером у наставку.

Пример 5.7.1. У оквиру корпорације постоје две компаније А и Б. На челу корпорације је особа која поседује мастер-приватни кључ (корен стабла). Особа на челу компаније А власник је приватног кључа, односно представља корен подстабла чворова у његовој хијерархији. Особа може надгледати трансакције које се догађају према јавним адресама унутар подстабла чији је корен приватни чвор у његовом поседу. Како се ради о приватном чвору, особа може потписати нове трансакције које шаљу биткоине. Биткоини који су доступни особи за слање су само они који су део претходних трансакција упућених ка адресама у подстаблу чији је корен приватни чвор. Уколико особа одлучи да неовлашћено потроши биткоине, она може да наштети хијерархији само за толико биткоина колико их има у подстаблу над

којим има делимично власништво. Особа на челу компаније Б је власник јавног кључа, односно у поседу је податка који се налази у јавном чвору, корену подстабла хијерархије. Особа само има увид у трансакције које се обављају унутар њеног подстабла, али за разлику од власника компаније А, она не може да троши биткоине. Оба власника немају увид у трансакције ван својих подстабала. Хијерархијски испод поменута два власника, може постојати иста структура подстабала која укључује шефове и на крају раднике. Једино власник корпорације, власник мастер-приватног кључа, има приступ свим биткоинима у хијерархији.

5.8 Упрошћена верификација плаћања

Као што је већ речено, новчаници потпуног чвора подразумевају да рачунар, на ком су инсталирани, садржи целокупну копију тренутног ланца блокова. Проблем недостатка меморије (објашњен у оквиру тачке 4.6) због чувања ланца отежава инсталирање тог типа новчаника, на пример на мобилни телефон. Сатоши Накамото представио је у свом раду [11] новчанике са упрошћеном верификацијом плаћања (енг. SPV wallet), познате и као „лаки чворови". Новчаник SPV учествује у проверавању исправности трансакције коју креира чувајући само заглавља блокова ланца. Простор који захтева чување свих заглавља ланца износи приближно 52MB[2] (јул, 2020. године), што је прихватљив простор за потребе просечног мобилног телефона. Новчаници потпуног чвора проверавају исправност трансакције тако што испитују порекло биткоина све до базне трансакције или до почетног блока, док новчаник SPV може само да провери да ли је нека трансакција укључена у блок и самим тим у ланац. Новчаник SPV базира се на чињеници да чворови рудари не би трошили своје време (и електричну енергију) на укључивање трансакције која није исправна. Начин на који новчаник типа лаки чвор проверава да ли је трансакција укључена у ланац јесте ослањање на новчаник типа потпуни чвор. Новчаник SPV захтева од насумичног новчаника потпуног чвора, са којим је повезан преко „протокола оговарања" (објашњеним у уводном делу главе), следеће информације:

- Мерклеов корен блока (тачка 4.3.2) унутар ког је трансакција укључена;
- Мерклеов доказ, односно само оне хеш вредности унутар Мерклеовог стабла помоћу којих лаки чвор може да реконструише грану унутар које се налази тражена трансакција.

Илустроваћемо примером на који начин SPV новчаник може да утврди унутар ког блока се налази трансакција Tx1 ослањајући се на новчаник потпуног чвора. Пример прати конструкцију Мерклеовог стабла дату унутар слике 4.2.

Пример 5.8.1. Новчаник SPV успоставља комуникацију са новчаником потпуног чвора и пита унутар ког блока се налази трансакција Tx1. Новчаник потпуног чвора налази блок и новчанику SPV шаље Мерклеов корен блока и хеш вредности Hash(Tx2) и Hash_34 који представљају Мерклеов доказ[3]. Новчаник на основу примљених информација реконструише Hash_12 користећи Hash(Tx1) и Hash(Tx2), а потом и Мерклеов корен Hash_1234, користећи израчунат Hash_12 и добијени Hash_34. Разлог зашто SPV реконструише делове стабла јесте да сâм дође до Мерклеовог корена који на крају може да упореди са вредношћу Мерклеовог корена који

[2]Ова вредност је добијена множењем тренутне висине блока са бројем бајтова које заузима једно заглавље (висина блока је 638744, док је величина заглавља блока у бајтовима 80).

[3]Новчаник потпуног чвора тврди да се трансакција налази унутар блока са Мерклеовим кореном који је послао и шаље релевантне хеш вредности унутар стабла као Мерклеов доказ.

је добио од новчаника потпуног чвора. Тиме доказује да се трансакција налази унутар блока чији се Мерклеов корен Hash_1234 налази у заглављу. На основу позиције нађеног блока, унутар своје верзије ланца заглавља блокова, новчаник SPV може да провери колико блокова је уланчано после нађеног блока и тиме утврди да ли је блок прихваћен од стране мреже као исправан. Просечан број блокова који се очекује да буде изнад нађеног блока, после ког се сматра да је блок прихваћен, је 6.

Новчаник лаког чвора може да буде преварен тако што добије погрешне информације од новчаника потпуног чвора, али се то превазилази тиме што SPV новчаник захтева информацију о Мерклеовом корену и Мерклеовом доказу од више чворова са којима је повезан, не само од једног. Због свог ослањања на друге чворове новчаници лаког чвора функционишу на исправан начин само у случају да мрежу није преузео злонамеран чвор (или група чворова) нападом 51%.

5.9 Персонализоване адресе

Чворови унутар мреже Биткоин су анонимни, али тиме се губи могућност брендирања и маркетинга (уколико би, на пример, једна фирма желела да има фиксну адресу, која није само псеудо-случајно генерисан низ карактера, већ садржи и неку кључну реч). Такве јавне адресе је могуће поседовати и називају се **персноналиозване адресе** (енг. Vanity addresses). Најпростији (и најдужи) начин доласка у посед оваквих адреса је да се приликом иницијализације адресе новчаника провери адреса и, уколико не одговара жељама корисника, генерише нова и тако изнова док не задовољи његове потребе о изгледу јавне адресе. Овакво решење је временски непрактично. Супротно томе, алат који пружа могућност проналаска адреса, тако да јавној адреси одговара део унетог низа карактера или унетог шаблона датог регуларним изразом, назива се „Vanitygen". На примеру 5.9.1[4] приказано је једно покретање овог алата.

Пример 5.9.1. $ ./vanitygen 1Love
Difficulty: 4476342
[48165 K/s] [$total$ 2080000] [$Prob$ 37.2%] [50% in 21.2s]

Где су:

- *1Love*; аргумент који представља жељени део адресе,
- *Difficulty*; тежина, односно просечан број адреса који мора да се претражује пре проналаска жељене. Повезаност дужине карактера жељене адресе и тежине њеног проналска је таква да што је дужи низ карактера жељене адресе, теже ју је пронаћи,
- *K/s*; број провера кључева у секунди,
- *total*; број проверених кључева,
- *Prob*; вероватноћа успешног проналажења кључа,
- *50% in 21.2s*; процењено време проналаска кључа са датом вероватноћом.

[4]Пример преузет са сајта у фебрару 2020. године са ког се може преузети и сâм алат - `https://github.com/samr7/vanitygen`. Могуће је и користити и сајт `http://bitcoinvanitygen.com/` који за циљ има генерисање перснонализованих адреса. Генерисање адресе преко интернета је несигурније од оне коју пружа алат корисника „samr7".

Резултат који даје алат је приватни кључ и њему одговарајућа Биткоин адреса која садржи жељени низ карактера. Јасно је да неко са рачунаром бољих перформанси може да дође брже до жељених адреса и да на тај начин нуди услугу другим корисницима. Такви сервиси би имали безбедоносни ризик да ли ће особа која ради на генерисању таквих кључева, задржати копију кључа за себе.
Персонализоване адресе су често мете напада „подметнутом адресом" (енг. tampering attack). У овом нападу злонамеран корисник може да генерише адресу која садржи идентичан низ карактера адресе неког познатог бренда и на тај начин може случајно бити део трансакције и доћи у посед послатих биткоина који њему нису намењени.

6 Формирање блока

Концепт формирања блока објашњен је у оквиру тачке 4.2. Ово поглавље покрива различите технике формирања блокова.

6.1 Тежина формирања блока

Замисао Сатоши Накамота била је стабилност мреже у динамици производње нових блокова као и награђивања чворова који у њој учествују. Реализована је кроз повећање тежине проблема повезаног са доказом о раду (тачка 4.1). Како се повећава укупна брзина израчунавања хеш вредности целокупне мреже, неопходно је контролисати динамику проналаска нових биткоина из формираних блокова. Велика присутност биткоина на мрежи смањиће његову вредност. Већ је речено да доказ о раду који систем Биткоин захтева може да се интерпетира као минималан број нула на почетку хеш вредности формираног блока. Тежина $nBits$ је вредност која се користи за рачуницу потребног броја нула хеш вредности новог блока. Тежина формирања блока ажурира се на (просечно) 2016 нових блокова (тачка 7.1.2). Потребан број нула D израчунава се као:

$$D = \log_2(nBits)/4 + 8,$$

где је $nBits$ тежина формирања блока, константа 4 представља дужину записа једног хексадецималног карактера, а константа 8 представља број нула потекао од минималне тежине формирања блока на почетку рада система коју је поставио Сатоши.

Пример 6.1.1. Тежина израчунавања формирања блока у време писања овог рада (мај, 2020. године) износи $nBits = 16{,}104{,}807{,}485{,}529$[1]. Минималан број нула D добија се као:

$$\begin{aligned} D &= \log_2(16{,}104{,}807{,}485{,}529)/4 + 8 \\ &= 18.96813916215 \approx 19. \end{aligned}$$

Ово одговара броју нула са којим тренутна глава ланца (чија је висина блока 630449) 000000000000000000001b3288f2c32137c504fa324264e80ce870f1e8b61ed69[2] почиње.

6.2 Хардвер за формирање блокова

Чворови рудари улажу у своје машине за израчунавање хеш вредности не би ли победили у трци коју им мрежа намеће и обично учествују у формирању докле год им се то исплати. IT компаније нудиле су своје производе и радиле на иновативним технологијама не би ли им у тој трци помогли да изађу као победници. Из тог разлога, на питање да ли је данас формирање блокова профитабилно за особу са просечним рачунаром, одговор је - није.

[1] Податак преузет са сајта у мају 2020. године `https://btc.com/stats/diff`.

[2] Целокупна структура блока може се видети `https://www.blockchain.com/btc/block/000000000000000000001b3288f2c32137c504fa324264e80ce870f1e8b61ed69`.

6.2.1 Формирање блокова уз помоћ CPU

Тежина проблема доказа у раду је на почетку била мала. Минимална тежина формирања блока захтевала је да хеш вредност блока почне са свега 8 нула, што је много мање у односу на данашњих 19. Проблем ове тежине могао се решити и помоћу кућног рачунара где би сва рачунања (погађања) обављао процесор. Данашње формирање блокова помоћу CPU-а не сматра се профитабилним, јер процесори немају добру подршку за брзо израчунавање хеш вредности и неефикасни су у погледу потрошње електричне енергије у ову сврху. Развој процесора је током времена био усмерен на паралелизацију послова што га је избацило из трке у формирању блокова. Тренуто најбржи процесор у израчунавању хеш вредности је AMD Ryzen 9 3900X са 12 језгара и брзином од 12.1 kH/s (kilo hash per second). Јединица задужена за математичка израчунавања унутар процесора је аритметичко-логичка јединица, коју графичке картице садрже у пуно већем броју од процесора. Последица тога је улагање чворова рудара управо у GPU (Graphics processing unit).

6.2.2 Формирање блока уз помоћ GPU

Улога графичке картице у рачунару је специјализован рад на видеорендеровању. За разлику од процесора, графичке картице имају много већи број језгара при чему су инструкције које језгро обавља простије у односу на оне које извршава процесор. Управо из разлога што су графичке картице више усмерене ка једносмернијем раду, учинило их је упешним у брзом формирању блокова. AMD и NVIDIA, као најпопуларнији произвођачи, временом су усмерили свој развој графичких картица не би ли оптимизовали брзину формирања блокова не трошећи пуно електричне енергије. Тренутно, картица са највећом брзином израчунавања хеш вредности је модел NVIDIA : Tesla V100-SXM2-16GB са брзином од 93.3 MH/s (mega hash per second). Рудари су нагомилавали графичке картице у формирању нових блокова паралелизујући њихов рад, чиме настају тзв. „биткоин фарме".

Термин **биткоин фарма** односи се на просторију, спрат зграде или читаву зграду чији је простор намењен за формирање блокова. Унутар тог простора налази се сва потребна опрема у сврху формирања блокова која се састоји од: техничке опреме која обавља израчунавања (нпр. десетине или стотине графичких картица) и система хлађења те опреме, јер је ризик од прегревања неке од компоненти велики. Овакве фарме изискују велику количину електричне енергије и рудари их најчешће настањују у земљама где су ти трошкови мањи (Исланд, Џорџија итд). Понекад уз електрану која фарму снабдева електричном енергијом.

6.2.3 Формирање блокова уз помоћ FPGA хардвера

Следећи корак у побољшању брзине и енергетске фикасности формирања блокова постигнут је FPGA хардвером 2011. године који је добио на популарности (након наглог пада вредности биткоина) 2018. године. FPG (Field-Programmable Gate Array) представља хардвер који може да се репрограмира тако да извршава специфичан посао. За разлику од графичких картица, троши много мање електричне енергије за исти број израчунатих хеш вредности. Разлог зашто формирање блокова уз помоћ FPGA није и данас популарно јесте у тежини конфигурације уређаја, односно рудар мора да зна да испрограмира хардвер FPGA у сврху формирања блокова, што није лако изводљиво. Просечна брзина израчунавања хеш вредности користећи FPGA хардвер је око 1GH/s (giga hash per second).

6.2.4 Формирање блокова уз помоћ ASIC хардвера

Свака од претходно описаних техника користила је хардвер који је могао да се искористи и у друге сврхе, поред формирања блокова. Тиме се не постиже максимална ефикасност и тако повећава потрошња електричне енергије коју хардвер захтева, јер је претходно дизајниран да буде до одређеног нивоа флексибилан. Хардвер који је специјално намењен за једну делатност назива се „ASIC" (Application-Specific Integrated Circuit). У случају биткоин ASIC чипова, хардвер је дизајниран тако да је предодређен само за формирање блокова у систему Биткоин[3]. Предност коришћења ове технике формирања блокова је у постојању биткоин чипова унутар којих се налази копија функције SHA256 (која се користи за рачунање хеш вредности блока). Тиме се добија на великој брзини израчунавања криптографске хеш вредности блока и повећавању шансе за победу у трци формирања новог блока. Тренутна просечна брзина израчунавања хеш вредности помоћу ASIC хардвера је 100 TH/s (tera hash per second), која га поставља на прво место алата за формирање блокова, али му је и цена набавке већа у поређењу са претходно споменутим варијантама хардверима.

[3]Постоје и други ASIC чипови испрограмирани да учествују у проширивању ланца алткоина.

7 Bitcoin Core

Термин „Биткоин" подразумева мрежу чворова, криптовалуту, али и имплементацију протокола, заједно са новчаником који га користи, који је развио Сатоши Накамото. Програм познат и као „Сатошијев клијент" (енг. Satoshi client), касније преименован у „Bitcoin Core", имплементација је протокола обављања трансакција унутар мреже. Bitcoin Core је написан у програмском језику `C++` који је објектно-оријентисан програмски језик, те је и цела структура кôда написана модуларно[1], односно без претеране међусобне зависности. Модуларност је олакшала гранање кôда на алтернативне криптовалуте (алткоини, споменути у уводу) настале управо на основу појединих модула Bitcoin Core. Програм је садржао и имплементиран модул за формирање блокова, али је он 2016. године избачен из употребе, јер већ у то време формирање блокова на кућном рачунару није имало пуно смисла. Како је кôд Bitcoin Core јаван, пројекат је доступан свима онима који желе да учествују у његовом развијању. То подразумева истраживање, тестирање програма, писање тестова, решавање проблема присутних у кôду[2], али и развијање нових функционалности. Свака измена која подразумева мењање тренуног кôда пропраћена је од стране затворене групе програмера на челу са Владимиром ван дер Ланом (Wladimir J. van der Laan). Његова званична улога, као главног програмера у развијању програма Bitcoin Core, јесте прегледање и на крају убацивање новонаписаних функционалности у Bitcoin Core као, и доношење одлуке када ће бити пуштена у рад нова верзија протокола. Пре њега су се тиме бавили Гевин Андерсен, Марти Малми (Martti Malmi), Ласло Ханјец (Laszlo Hanyecz)[3], а први од њих Сатоши Накамото. Новчаник унутар Bitcoin Core је тип новчаника пуног чвора (глава 5).

7.1 Анализа кôда Bitcoin Core

У овој тачки биће посвећена пажња анализи различитих делова кôда програма.

7.1.1 Bitcoin Core мреже

Новчаник се може повезати на три различите мреже: главна мрежа (енг. main network), тест мрежа (енг. test net), мрежа регресионог теста (енг. regression test). **Главна мрежа** је намењена обављању трансакција између регуларних чворова и унутар главне мреже, биткоин има своју реалну тржишну вредност. **Тест мрежа** је алтернатива главној мрежи и служи за тестирање измена направљених у кôду протокола или новчаника. Биткоини који су део трансакција унутар тест мреже немају вредност ван ње. Чворови унутар тест мреже одржавају своју верзију ланца

[1] Кôд није био модуларан у првој верзији новчаника, већ је већински део имплементације био садржан унутар фајла *main.cpp*. Прва верзија кôда налази се на локацији `https://github.com/benjyz/bitcoinArchive`

[2] Целокупан списак нерешених проблема налази се на локацији `https://github.com/bitcoin/bitcoin/contribute`. Лабела *good first issue* назнака је програмерима који немају претходног искуства са решавањем проблема унутар кôда новчаника.

[3] Ласло Хајнец је први који је реализовао трансакцију размене биткоина за робу. Трансакција се састојала од размене његових 10 000BTC за два парчета пице. Вредност његових биткоина 2010. године када је и реализована транскација, износила је око 40$. Говорећи у терминима данашње вредности биткоина (јун 2020. године), он је две пице платио преко 90 000 000$.

блокова. У тест мрежи се детектују, а потом и отклањају потенцијални проблеми пре него што измене постану део главне мреже. Како се кроз тест мрежу обавља много мање трансакција у односу на званични ланац, дужина ланца у тест мрежи је вишеструко мања величина ланца тест мреже износи $\approx$ 40GB (тачка 7.1.4), док величина главног ланца износи преко 260GB. Додатно, сваки усвојени предлог о унапређењу, BIP (дефиниција BIP налази се у тачки 4.4), који је прихваћен од стране рудара, најпре се проверава у тест мрежи. **Мрежа регресионог теста** има сличну улогу као тест мрежа, једина разлика је у томе што је време потребно за формирање нових блокова унутар ове мреже много мање у односу на главну и тест мрежу. Управо из тог разлога тестирање измене, која зависи од броја формираних блокова, изводи се у мрежи регресионог теста.

Главна мрежа, тест мрежа и мрежа регресионог теста, дефинисане су редом унутар фајла *chainparams.cpp* као класе *CMainParams*, *CTestNetParams*, *CRegTestParams*. Неки од заједничких параметара за ове три класе су:

- *nPowTargetTimeSpan*; параметар који означава време за које ће мрежа поново ажурирати тежину формирања блокова (тачка 6.1). У оквиру све три мреже параметар има вредност: $14 * 24 * 60 * 60$ - где чиниоци редом означавају: 14 дана, 24 сата, 60 минута, 60 секунди. Ова рачуница одговара чињеници да тежина формирања блока треба да буде ажурирана на сваке две недеље,

- *nPowTargetSpacing*; параметар који означава колико често ће бити формиран нови блок. У оквиру све три мреже параметар има вредност: $10 * 60$ - где чиниоци редом означавају: 10 минута, 60 секунди. Параметар постоји у оквиру мреже регресионог теста, али се не користи,

- *nMinerConfirmationWindow*; параметар који означава вредност на колико блокова ће се ажурирати тежина формирања блока. Вредност параметра *nMinerConfirmationWindow* унутар класе CRegTestParams (мрежа регресионог теста) је nMinerConfirmationWindow = 144, док је у оквиру друге две класе вредност добијена коришћењем претходна два параметра:

$$nMinerConfirmationWindow = \frac{nPowTargetTimeSpan}{nPowTargetSpacing} = 2016.$$

- *defaultAssumeValid*; параметар који означава број блокова уназад, пре појављивања новог блока, који се проверавају у тренутку испитивања исправности новог блока. Вредност параметра унутар све три класе је $nPruneAfterHeight = 623950$,

- *SegwitHeight*; параметар који означава тренутак меког гранања и укључивање одвојеног сведока (тачка 4.6.2) у мрежу. У оквиру главне и тест класе, вредност овог параметра одговара висини блока унутар ког је он активиран и има вредност $SegwitHeight = 481824$. У класи регресиони тест, вредност параметра је 0. Постоје и слични параметри као што су *BIP16Exception*, *BIP34Height* итд,

- *nMinimumChainWork*; параметар који означава минималан укупан број хешева блокова[4] ланца чију копију клијент преузима при првом покретању када се синхронизује са остатком мреже. Параметар има улогу да заштити клијента да не буде преварен од стране нападача (као једног или више чворова са којим се синхронизује клијент) који нуде лажни ланац блокова као главни.

[4]Термин *chainwork* означава укупан број израчунатих хешева у ланцу.

Вредност овог параметра ажурира се при сваком објављивању нове верзије протокола, јер се дужина ланца, а и самим тим број израчунатих хеш вредности, у међувремену повећала. Главна и тест мрежа имају различите вредности за овај параметар, јер имају различите дужине ланаца. Унутар класе мреже регресионог теста вредност параметра је 0,

- *nSubsidayHalvingInterval*; као што је објашњено у оквиру тачке 4.2, награда за формиран блок се повремено полови. Вредност параметра *nSubsidayHalvingInterval* одговара броју нових блокова које је потребно пронаћи после којих ће се награда преполовити. У оквиру класе главне мреже и тест мреже, вредност параметра је $nSubsidayHalvingInterval = 210000$, док је у класи мреже регресионог теста $nSubsidayHalvingInterval = 150$.

7.1.2 Почетни блок

Почетни блок (енг. genesis block) је први блок креиран у ланцу и једини који нема свог родитеља већ само потомке. Креирање почетног блока је директно убачено у кôд и сви наредно формирани блокови ће се уланчавати даље иза њега. Унутар фајла *chainparams.cpp* налази се функција *CreateGenesisBlock* чија је улога креирање објекта класе *CBlock* декларисане унутар фајла *block.h*. Класа садржи податке о заглављу блока (структура заглавља је објашњена у оквиру тачке 4.3). Алгоритам 7.1 је кôд који креира почетни блок.

Алгоритам 7.1: CreateGenesisBlock

```
1
2  static CBlock CreateGenesisBlock(const char* pszTimestamp, const CScript&
       genesisOutputScript, uint32_t nTime, uint32_t nNonce, uint32_t nBits,
       int32_t nVersion, const CAmount& genesisReward)
3  {
4      CMutableTransaction txNew;
5      txNew.nVersion = 1;
6      txNew.vin.resize(1);
7      txNew.vout.resize(1);
8      txNew.vin[0].scriptSig = CScript() << 486604799 << CScriptNum(4) <<
           std::vector<unsigned char>((const unsigned char*)pszTimestamp, (const
           unsigned char*)pszTimestamp + strlen(pszTimestamp));
9      txNew.vout[0].nValue = genesisReward;
10     txNew.vout[0].scriptPubKey = genesisOutputScript;
11
12     CBlock genesis;
13     genesis.nTime    = nTime;
14     genesis.nBits    = nBits;
15     genesis.nNonce   = nNonce;
16     genesis.nVersion = nVersion;
17     genesis.vtx.push_back(MakeTransactionRef(std::move(txNew)));
18     genesis.hashPrevBlock.SetNull();
19     genesis.hashMerkleRoot = BlockMerkleRoot(genesis);
20     return genesis;
21 }
```

Параметри функције су:

- pszTimestamp; вредност овог параметра у случају почетног блока је: *"The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"*. Ово је текст

са насловне стране дневних британских новина „The Times" објављен 3. јануара 2009. године. Разлог овакве структуре <scriptSig> у случају почетног блока је доказ да блок није направљен пре тог датума и тиме се искључује могућност да је Сатоши креирао много дужи, тајни ланац пре него што је почетни блок ланца заиста представљен. Новине су добро покриће за потпис, јер Сатоши није могао да зна текст насловне стране пре него што је објављен,

- genesisOutputScript; излазна скрипта која садржи адресу,
- nTime; временски печат изражен кроз *UNIX* време[5]. Вредност параметра у случају почетног блока је 1231006505 што означава датум 3. јануар 2009. године, 18.15 часова (GMT),
- nNonce; псеудо-случајан број за формирање блока. У случају почетног блока та вредност је фиксна и износи 486604799 (0x1D00FFFF),
- nBits; параметар који одговара тежини формирања блока (тачка 4.1),
- nVersion; верзија Биткоина. У случају почетног блока вредност параметра је 1,
- genesisReward; награда за формирање блока. Награда за формирање почетног блока је 50*COIN. Вредност COIN је статичка константа дефинисана унутар фајла *Amount.h*[6] и дефинисана је као $COIN = 100\ 000\ 000$. Јединица мере је сатоши.

Прва половина тела функције (линије 4 - 10) базира се на креирању и попуњавању поља објекта класе *CMutableTransaction* смештеног унутар фајла *transactions.h*. Како је већ објашњено, базна трансакција може да има улогу у гласању за BIP остављајући поруку у оквиру дела <scriptSig>. У овом случају Сатоши је искористио поље <sig> да остави јединствену поруку користећи pszTimestamp, nNonce и вредности 4 као *extraNonce*[7]. Разлог баш овакве конструкције поруке, није познат. Излаз TxOut базне трансакције садржи вредност награде за формирање блока, *genesisReward* и scriptPubKey. Карактеристично за базну трансакцију унутар почетног блока је и то што награду формирања блока (50 биткоина) није могуће даље трошити. Трансакција је искључена из ланца од стране свих новчаника условом у кôду који прескаче све трансакције блока у случају да хеш вредност блока одговара хеш вредности почетног блока.
Друга половина тела функције (линије 13 - 19) додељује вредности променљивим које одговарају заглављу блока. Заглавље се попуњава подацима:

- nTime; тренутак формирања почетног блока израженог у UNIX времену,
- nBits; тежина формирања почетног блока,
- nNonce; псеудо-случајан број,
- nVersion; верзија протокола,

[5] *UNIX* време означава број секунди од почетка *UNIX* епохе 1. јануара 1970. године до произвољног тренутка.

[6] Унутар овог фајла налази се и податак о укупном броју новчића унутар мреже кроз статичку константу $MAX\_MONEY = 21\ 000\ 000 * COIN$ што одговара 21 000 000 биткоина.

[7] Улога *extraNonce*-а је да уведе додатни ниво случајности у рачунању хеш вредности блока. Његово коришћење је ван протокола, односно не постоји такво поље унутар заглавља данашњих блокова.

- txNew; криптографска хеш вредност трансакције,
- вредност родитеља почетног блока се поставља на *NULL*,
- Мерклеов корен (тачка 4.3.2).

7.1.3 Скрипт

У оквиру тачке 3.1 било је речи о језику скрипт помоћу којег се дефинишу скрипта <scriptSig> и скрипта <scriptPubKey> унутар трансакције. У наставку ове тачке биће представљено на који начин је тај језик и реализован.

CScript

Операције унутар језика скрипт дефинисане су као набројиви типови под називом *enum opcodetype*, унутар фајла *script.h*. Постоји 112 јединствених операција од којих је 15 искључено из употребе јер могу да изазову поједине проблеме у кôду. Међу њима су: OP_CAT, OP_SUBSTR, OP_LEFT итд. Класа CScript наслеђује класу *CScriptBase* (која је у основи класа *vector*), чува и обрађује податке потребне за обраду скрипте. Подаци унутар скрипте могу бити: децимални број, оператор и вектор карактера (који представља комбинацију оператора и операнада). Подаци којима се попуњава објекат CScript су скрипте <scriptSig> и <scriptPubKey>. Попуњавање се врши редефинисаним оператором за унос података „≪” у зависности од типа улазног податка. Алгоритам 7.2 је кôд који имплементира оператор „≪” када је тип аргумента вектор карактера.

Алгоритам 7.2: operator≪(const std::vector<unsigned char>& b)

```
1  CScript& operator$<<(const std::vector<unsigned char>& b)
2  {
3     if (b.size() < OP_PUSHDATA1)
4     {
5        insert(end(), (unsigned char)b.size());
6     }
7     else if (b.size() <= 0xff)
8     {
9        insert(end(), OP_PUSHDATA1);
10       insert(end(), (unsigned char)b.size());
11    }
12    else if (b.size() <= 0xffff)
13    {
14       insert(end(), OP_PUSHDATA2);
15       uint8_t _data[2];
16       WriteLE16(_data, b.size());
17       insert(end(), _data, _data + sizeof(_data));
18    }
19    else
20    {
21       insert(end(), OP_PUSHDATA4);
22       uint8_t _data[4];
23       WriteLE32(_data, b.size());
24       insert(end(), _data, _data + sizeof(_data));
25    }
26    insert(end(), b.begin(), b.end());
27    return *this;
```

```
28 }
```

Уколико је аргумент функције „≪” типа вектор, уписивању података из вектора претходи упис његове дужине као индикатор за касније читање. Уколико број бајтова за запис дужине превазилази вредност једног од оператора OP_PUSHDATA{1, 2, 4}, вредност тог оператора се користи као ознака у запису величине вектора. На пример, у случају да вредност за запис дужине вектора превазилази вредност оператора OP_PUSHDATA2, за запис дужине користе се 2 бајта. Ако та вредност превазилази вредност оператора OP_PUSHDATA4, за запис се користе 4 бајта. Потреба за оператором OP_PUSHDATA4 још увек не постоји, јер би његово коришћење означавало читање података величине 4GB.

У случају када је тип аргумента оператор, врши се тривијална провера да ли је оператор валидан, односно да ли се налази у скупу дефинисаних оператора. Супротно, он се само чува, што показује кôд дат у алгоритму 7.3.

Алгоритам 7.3: operator«(opcodetype opcode)

```
1  CScript& operator<<(opcodetype opcode)
2  {
3     if (opcode < 0 || opcode > 0xff)
4        throw std::runtime_error("CScript::operator<<(): invalid opcode");
5     insert(end(), (unsigned char)opcode);
6     return *this;
7  }
```

Где је 0xff вредност оператора OP_INVALIDOPCODE.
Алгоритам 7.4 је кôд који имплементира оператор „≪“ када је аргумент децимални број:

Алгоритам 7.4: operator«(opcodetype opcode)

```
1  CScript& operator<<(int64_t b) { return push_int64(b); }
2
3  CScript& push_int64(int64_t n)
4  {
5     if (n == -1 || (n >= 1 && n <= 16))
6     {
7        push_back(n + (OP_1 -1));
8     }
9     else if (n == 0)
10    {
11       push_back(OP_0);
12    }
13    else
14    {
15       *this << CScriptNum::serialize(n);
16    }
17    return *this;
18 }
```

где је децимална вредност оператора OP_1 = 81. Уколико је вредност аргумента n = -1, његова вредност се записује као $n + (\text{OP\_1} - 1)$. Крајњи резултат овог израза даје 79, што је такође децимална вредност оператора OP_1NEGATE. Уколико је вредност у опсегу од 1 до 16, резултат израза ће бити у опсегу од 81 до 96 што одговара редом децималним вредностима оператора OP_{1, ..., 16}. Када је $n = 0$,

у запис улази оператор OP_0. У супротном, вредност се серијализује помоћу статичке функције *serialize*. Улога ове функције јесте да сачува бројеве чије вредности превазилазе вредности оператора OP_16. Реализована је уписивањем битовске репрезентације броја n.

Евалуација скрипте

Функција која парсира податке скрипт објекта је *EvalScript*. Улазни аргументи функције су: празан вектор вектора којим се реализује стек (у ознаци *stack*) и над којим ће се обављати скрипт инструкције, *const* референца објекта типа CSript, *unsigned int* као индикатор специфичности скрипте (на пример да ли се ради о трансакцији P2SH, тачка 3.4), *const* адреса објекта за проверу потписа, *BaseSignatureChecker*, аргумент набројивог типа који је индикатор да ли се користи SegWit (тачка 4.6.2), показивач на објекат типа *ScriptError* који се ажурира кроз помоћну функцију *set_ error* у случају грешке приликом парсирања.
Структура функције је таква да се у петљи итерира кроз податке објекта *CScript* чиме се они парсирају у одређеним гранама кôда у зависности од оператора у тренутној итерацији. Скраћена верзија кôда[8], где се парсирају само оператори коришћени у табели 3.3, уз макро и помоћну функцију која олакшава рад са стеком, приказана је у оквиру алгоритма 7.5.

Алгоритам 7.5: EvalScript (скраћена верзија)

```
1  //...
2  #define stacktop(i) (stack.at(stack.size()+(i)))
3  static inline void popstack(std::vector<valtype>& stack)
4  {
5     if (stack.empty())
6        throw std::runtime_error("popstack(): stack empty");
7     stack.pop_back();
8
9  }
10 //...
11 bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const
       CScript& script, unsigned int flags, const BaseSignatureChecker& checker,
       SigVersion sigversion, ScriptError* serror)
12 {
13    //...
14    opcodetype opcode;
15    valtype vchPushValue;
16    CScript::const_iterator pc = script.begin();
17    CScript::const_iterator pend = script.end();
18    CScript::const_iterator pbegincodehash = script.begin();
19    //...
20    while (pc < pend)
21    {
22       //...
23       if (!script.GetOp(pc, opcode, vchPushValue))
24          return set_error(serror, SCRIPT_ERR_BAD_OPCODE);
25       //...
26       stack.push_back(vchPushValue);
27       //...
28       switch (opcode)
```

[8]Функција у целости има преко 700 линија кôда.

```
29      {
30         //...
31         case OP_DUP:
32         {
33         if (stack.size() < 1)
34         return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
35         valtype vch = stacktop(-1);
36         stack.push_back(vch);
37         }
38         break;
39      }
40         //...
41         case OP_RIPEMD160:
42         case OP_SHA1:
43         case OP_SHA256:
44         case OP_HASH160:
45         case OP_HASH256:
46         {
47            if (stack.size() < 1)
48               return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
49            valtype& vch = stacktop(-1);
50            valtype vchHash((opcode == OP_RIPEMD160 || opcode == OP_SHA1 ||
                  opcode == OP_HASH160) ? 20 : 32);
51            //...
52            else if (opcode == OP_HASH160)
53               CHash160().Write(vch.data(),
                     vch.size()).Finalize(vchHash.data());
54            //...
55            popstack(stack);
56            stack.push_back(vchHash);
57         }
58         break;
59         //...
60         case OP_EQUAL:
61         case OP_EQUALVERIFY:
62         {
63            if (stack.size() < 2)
64               return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
65            valtype& vch1 = stacktop(-2);
66            valtype& vch2 = stacktop(-1);
67            bool fEqual = (vch1 == vch2);
68
69            popstack(stack);
70            popstack(stack);
71            stack.push_back(fEqual ? vchTrue : vchFalse);
72            if (opcode == OP_EQUALVERIFY)
73            {
74               if (fEqual)
75                  popstack(stack);
76               else
77                  return set_error(serror, SCRIPT_ERR_EQUALVERIFY);
78            }
79         }
80         break;
```

```
81          //...
82          case OP_CHECKSIG:
83          case OP_CHECKSIGVERIFY:
84          {
85             if (stack.size() < 2)
86                return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
87
88             valtype& vchSig  = stacktop(-2);
89             valtype& vchPubKey = stacktop(-1);
90
91             bool fSuccess = true;
92             if (!EvalChecksig(vchSig, vchPubKey, pbegincodehash, pend, flags,
                   checker, sigversion, serror, fSuccess)) return false;
93             popstack(stack);
94             popstack(stack);
95             stack.push_back(fSuccess ? vchTrue : vchFalse);
96             if (opcode == OP_CHECKSIGVERIFY)
97             {
98                if (fSuccess)
99                   popstack(stack);
100               else
101                  return set_error(serror, SCRIPT_ERR_CHECKSIGVERIFY);
102            }
103         }
104         break;
105         //...
106   }
107}
```

Функција *GetOpCode* враћа *тачно* уколико оператор постоји на тренутној позицији итератора *pc* којим се обилази скрипта. Вредност оператора садржана је унутар промењиве *opcode*, а вредност операнада, уколико их оператор захтева, је унутар *vchPushValue*. Супротно, уколико функција врати *нетачно*, поставља се тип грешке *SCRIPT_ERR_BAD_OPCODE* као вредност објекта *serror*, прекида се извршавање и скрипта се сматра неисправном. Операнди се стављају на врх стека, а *switch* гранање извршава се у зависности од оператора. Гранања су следећа (основна функционалност оператора је објашњена у оквиру тачке 3.2):

- *case OP_DUP*; уколико је величина стека већа од 1, последња вредност са стека се копира и ставља на врх стека. У противном, *serror* се поставља на вредност *SCRIPT_ERR_INVALID_STACK_OPERATION* и скрипта се сматра неисправном,

- *case OP_HASH160*; у тренутку извршења овог оператора на врху стека требало би да се нађе јавни кључ; његова копија се узима са стека и привремено смешта унутар промењиве *vch*. Резултат примене оператора је хеш вредност јавног кључа уз помоћ криптографских хеш функција SHA256 и RIPEMD160. Оператор се извршава конструкцијом привременог објекта типа *CHash160*, уписом вредности јавног кључа и на крају позивом функције *Finalize* над привременим објектом. Функција *Finalize* је помоћна функција која позива функцију *Transform* која имплементира функције SHA256 и RIPEMD160. Имплементација криптографских хеш функција одговара алгоритмима објашњеним у тачкама 2.3.1 и 2.3.2. Јавни кључ се скида са стека, а на врх стека се гура

његова хеш вредност. Услов за исправно стање стека, самим тим и скрипте, идентичан је као у случају гране OP_DOP,

- *case OP_EQUALVERIFY*; узимају се копије последње две вредности са врха стека и пореде се. Резултат поређења смешта се у променљиву *fEqual*. Уколико вредност унутар променљиве *fEqual* није *тачно*, скрипта се сматра неисправном и прекида се извршавање *while* петље. У супротном, извршавање се наставља. Иста грана је искоришћена и за оператор *OP_EQUAL*, где се променљиве пореде, али за ралику од *OP_EQUALVERIFY*, резултат поређења се смешта на крај вектора (врх стека),

- *case OP_CHECKSIGVERIFY*; са врха стека се узима копија потписа и привремено смешта у променљиву *vchSig*, а за њим и јавни кључ и смешта унутар променљиве *vchPubKey*. Променљиве *vchSig* и *vchPubKey* су прва два аргумента помоћне функције *EvalCheckSig*. Аргументи *flags*, *checker*, *sigversion*, *serror*, прослеђују се са вредностима које су послате функцији *EvalScript*. Аргументи *pbegincodehash* и *pend* остављају могућност да се провера исправности потписа не врши над целом скриптом, већ само над њеним делом. Почетак скрипте, у том случају, дефинише се преко оператора OP_CODESEPARATOR, који модификује вредност променљиве *pbegincodehash* и поставља њену вредност на тренутну вредност итератора *pc*. Тренутно не постоји трансакција, дефинисана протоколом, која користи овај оператор.

  Функција *EvalChecksig* прво проверава исправност конструкције скрипте трансакције у односу на променљиве *flags* и *segversion* и креира привремени објекат класе *CScript* под називом *scriptCode* који позива конструктор са вредношћу итератора *pbegincodehash* и *pend*. На крају, позива функцију *CheckSig* над објектом *checker* класе *BaseSignatureChecker* и прослеђује јој аргументе *vchSig*, *vchPubKey*, *scriptCode*, *sigversion*. Скраћена верзија кôда функције *CheckSig* приказана је у оквиру алгоритма 7.6.

Алгоритам 7.6: CheckSig (скраћена верзија)

```
1
2  template <class T>
3  bool GenericTransactionSignatureChecker<T>::CheckSig(const
       std::vector<unsigned char>& vchSigIn, const std::vector<unsigned
       char>& vchPubKey, const CScript& scriptCode, SigVersion sigversion)
       const
4  {
5      CPubKey pubkey(vchPubKey);
6      if (!pubkey.IsValid())
7          return false;
8
9      std::vector<unsigned char> vchSig(vchSigIn);
10     if (vchSig.empty())
11         return false;
12     int nHashType = vchSig.back();
13     vchSig.pop_back();
14
15     uint256 sighash = SignatureHash(scriptCode, *txTo, nIn, nHashType,
           amount, sigversion, this->txdata);
16
17     if (!pubkey.Verify(sighash, vchSig);)
18         return false;
```

```
19
20      return true;
21  }
```

Функција *CheckSig* прво конструише привремени објекат *pubkey* класе *CPubKey* са прослеђеним податком о јавном кључу (променљива *vchPubKey*) и проверава његову исправност функцијом *IsValid*. Функција *IsValid* проверава дужину јавног кључа. Уколико је дужина 0, следи да је потпис неисправан или је скрипта лоше конструисана. Протокол налаже да се вредност типа хеша (тачка 3.1) налази на крају скрипте <scriptSig>, чиме се иницијализује променљива *nHashType*. Потом се израчунава хеш вредност потписа трансакције уз помоћ функције *SignatureHash* користећи податке о трансакцији чији се потпис верификује. Резултат се прослеђује функцији *Verify* објекта *pubkey* заједно са прослеђеном вредношћу потписа. Функција *Verify*, проверава исправност потписа поступком објашњеним у тачки 2.5.3. Функција враћа вредност 1 у случају исправног потписа, а 0 у супротном.

Новчаник реагује на поруке унутар мреже послате од стране других чворова функцијом *ProcessMessage* која се налази унутар фајла *net_processing.cpp*. Једна од тих порука је и нови блок, који се новчанику даје на проверу, пошто га рудар објави. Новчаник коме је порука таквог типа стигла, покушава нови блок да надовеже на (њему познат) тренутно најдужи ланац функцијом *ConnectBlock* имплементираном унутар фајла *validation.cpp*. Улога функције *ConnectBlock* је да итерира кроз све трансакције блока и провери исправност скрипти унутар њих. Провера исправности скрипти врши се индиректно позивајући функцију *EvalScript*. Функција *ConnectBlock* такође имплементира игнорисање базне трансакције у случају почетног блока, као што је већ речено унутар тачке 7.1.2.

7.1.4 Експерименти са новчаником

Након преузимања кôда BitcoinCore-а[9] и његовог компајлирања, програм се може користити уз помоћ графичког интерфејса или помоћу командне линије. Постоје три програма која се могу покренути, а то су:

- bitcoinid; програм који при покретању синхронизује ланац блокова на рачунару са осталим пуним чворовима у мрежи. Bitcoind може да се покрене са аргументом <−−daemon> што ће омогућити да програм синхронизује ланац блокова у позадини, док је за то време могуће позивати његове функционалности,

- bitcoin-cli; програм који пружа коришћење свих услуга новчаника преко терминала. Услуге су типа: слање и примање биткоина, информације о блоку на основу његове хеш вредности итд[10]. Пре позива било које функције програма bitcoin-cli, неопходно је претходно покретање програма bitcoinid,

- bitcoinid-qt; програм који интегрише програме bitcoind и bitcoin-cli уз додатак графичког интерфејса. У позадини програма bitcoinid-qt, програм bitcoinid синхронизује ланац у односу на рачунар са ког се покреће у позадини, док се кроз графички интерфејс позивају функције програма bitcoin-cli.

[9] BitcoinCore се може преузети са локације `https://github.com/bitcoin/bitcoin`

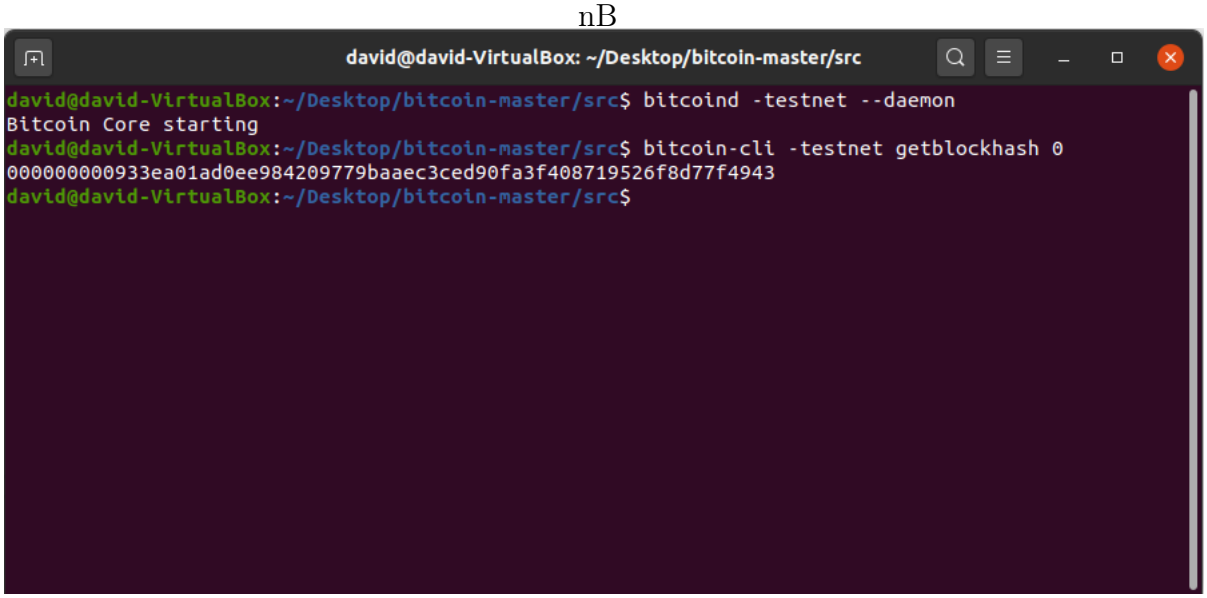
[10] Целокупан списак команди може се погледати на сајту `https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_calls_list` .

Сва три наведена програма могу да се покрену повезујући се са све три различите мреже (тачка 7.1.1), а то су: главна, тест и мрежа регресионог теста. Додатни аргумент командне линије за покретање било ког од наведених програма из листе на неку од тест мрежа су:<-testnet>, за тест мрежу, <-regtest> за мрежу регресионог теста. Уколико се не наведе ниједан аргумент, покретање програма подразумева да се новчаник повезује на главну мрежу. Која год мрежа да се изабере (од наведених) за покретање програма, најпре bitcoinid, а потом bitcoin-cli, аргумент који одговара одабиру мреже мора бити исти за оба програма.

Модификација клијента

У наставку је дат пример модификације програма bitcoin-cli додавањем нове функционалност, а потом њено позивање. Са слике 7.1 може се видети покре-

nB



Слика 7.1: GetBlockHash

тање програма bitcoind, његово качење на тест мрежу (<-testnet>), као и аргумент који назначава да ће програм своју синхронизацију ланца радити у позадини (<−−daemon>). Након покретања програма bitcoind, покреће се програм bitcoin-cli са истим аргументом (<-testnet>) као назнаком да се будући позиви функција извршавају над тест мрежом. Функција која се позива је *getblockhash* и као аргумент прихвата висину блока као ознаку позиције у ланцу са ког се блок узима. Аргумент 0 одговара почетном блоку у ланцу и његова хеш вредност се види са слике као резултат функције.

Функција коју ћемо додати рачуна укупну вредност биткоина у оквиру једног блока који ћемо реферисати на основу његове висине (позиције) у ланцу. Како функција користи податке из ланца блокова (да бисмо приступила блоку и прегледали његове трансакције), функцију додајемо унутар фајла *blockchain.cpp*. Дефиниција функције дата је у наставку.

Алгоритам 7.7: getblockvalue

```
1
2 static UniValue getblockvalue(const JSONRPCRequest& request)
3 {
4     int blockHeight = request.params[0].get_int();
5     if (blockHeight < 0 || blockHeight > ::ChainActive().Height())
```

```
6       {
7           throw JSONRPCError(RPC_INVALID_PARAMETER, "Block height out of range");
8       }
9
10      CBlockIndex* pBlockIndex = ::ChainActive()[blockHeight];
11      if(pBlockIndex == nullptr)
12      {
13          throw JSONRPCError(RPC_INVALID_PARAMETER, "Block index is NULL");
14
15      }
16      const CBlock block = GetBlockChecked(pBlockIndex);
17      UniValue resultObject(UniValue::VOBJ);
18      double blockValue = 0;
19      int numberOfTransactions = 0;
20
21      for(const auto &transaction : block.vtx)
22      {
23          for (unsigned int i = 0; i < transaction->vout.size(); ++i)
24          {
25              blockValue+= transaction->vout[i].nValue;
26          }
27          ++numberOfTransactions;
28      }
29
30      resultObject.pushKV("blockHeight",      (int)blockHeight);
31      resultObject.pushKV("numberOfTransactions", (int)numberOfTransactions);
32      resultObject.pushKV("blockValue",       ValueFromAmount(blockValue));
33
34      return resultObject;
35  }
```

Функција *getblockvalue*[11] дефинисана је тако да јој је повратна вредност *JSON* објекат, док је аргумент висине дефинисан унутар објекта *request*. Након провере да ли задата висина као аргумент испуњава услове (да је већа од 0 и да не превазилази тренутну висину ланца), приступа се адреси блока помоћу показивача *pBlockindex*. Показивач *pBlockIndex* се касније користи као аргумент функције *GetBlockChecked* који, након провере да ли је показивач исправно постављен, чита блок из ланца блокова (који је уписан на диск). Потом се итерира кроз све трансакције унутар блока, где се у оквиру једне трансакције обилазе сви излази из ње. Збир вредности биткоина на излазима свих трансакција чува се унутар промењиве *blockValue*. Промењива *numberOfTransactions* одговара укупном броју трансакција садржаном у блоку. На крају *JSON* објекат формира се тако што се користи функција *pushKV* (push key-value), где је кључ (key) идентификатор промењиве, а вредност (value) резултат израчунавања. Да би ова функција била видљива као команда која се позива кроз клијент bitcoin-cli, морамо проширити већ постојећи списак команди који се налази унутар истог фајла. Једну команду дефинише:

- категорија; у нашем случају „blockchain”,
- име; име команде која се додаје,
- функција; меморијска адреса функције која ће се позвати при позиву унете команде,

[11] Функција је креирана по угледу на функцију *getblockhash*

- аргументи; уколико команда захтева аргументе, они се означавају именом као вектор стрингова.

Пример укључивања команде и за коју функцију је везана, приказана је у кôду алгоритма7.8.

Алгоритам 7.8: Везивање функције за команду (скраћена верзија)

```
1
2  void RegisterBlockchainRPCCommands(CRPCTable &t)
3  {
4  // clang-format off
5  static const CRPCCommand commands[] =
6  { //  category            name                      actor (function)
       argNames
7    //  ---------------------------------------------------------------------
       -----
8     { "blockchain",       "getblockvalue",    &getblockvalue,    {"height"} },
9    //...
```

Како наша нова команда захтева аргумент који одговара висини, потребно је проширити и скуп аргумената команди дефинисаних у фајлу *client.cpp*. Списак аргумената који је везан за команду дефинишу:

- име команде,
- индекс аргумента у позиву,
- име аргумента који одговара истом имену и унутар вектору команди *commands*[] из фајла *blockchain.cpp*.

Пример проширивања приказан је у кôду алгоритма 7.9.

Алгоритам 7.9: Везивање аргумената за команду (скраћена верзија)

```
1
2  static const CRPCConvertParam vRPCConvertParams[] =
3  {
4     { "getblockvalue", 0, "height" },
5  //...
```

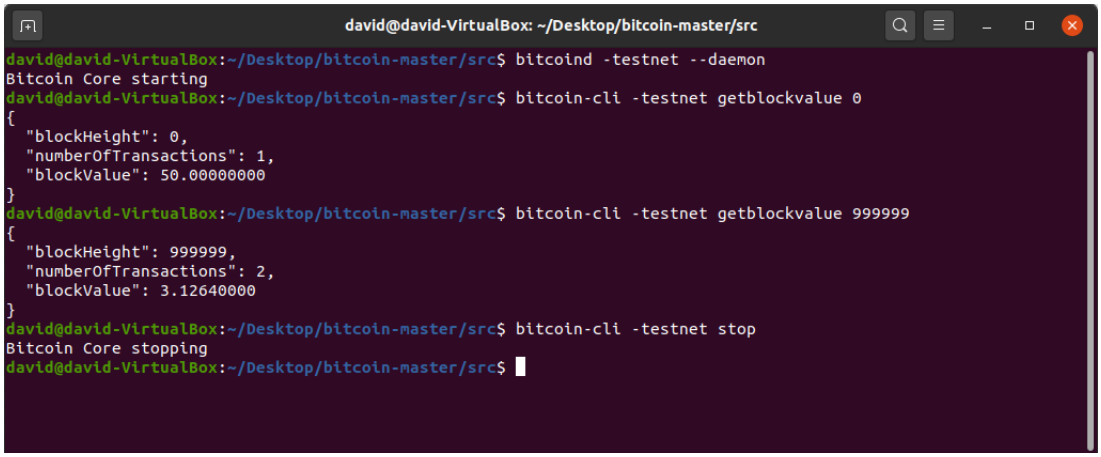
Пример покретања програма bitcoind на тест мрежи, а затим и команде *getblockvalue* са аргументима 0 и 99999999 дат је на слици 7.2. Први резултат[12] који даје целокупну вредност блока одговара вредности почетног блока. Почетни блок има само једну трансакцију, а вредност излаза му је фиксан, 50BTC и одговара награди формирања почетног блока (тачка 7.1.2).

Употреба новчаника

За потребе приказа реалне употребе новчаника, користићемо BitcoinCore клијент, преузет са сајта[13], инсталиран на Windows оперативном систему и повезан

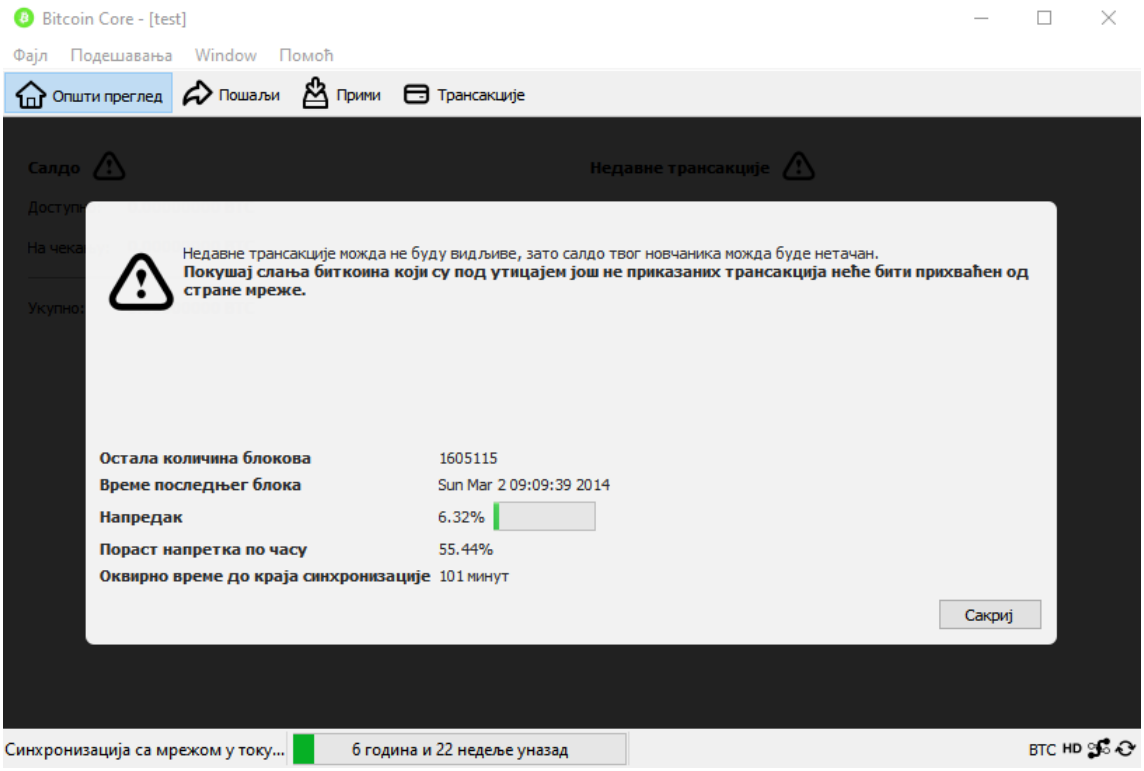
[12] Резултати се могу проверити помоћу сајта: `https://blockstream.info/testnet/`

[13] `https://bitcoin.org/en/download`



Слика 7.2: getblockvalue

на тест мрежу (са изабраним српским језиком за коришћење клијента). При првом покретању, корисник одређује локацију где ће се наћи целокупан ланац блокова уз упозорење колико је простора неопходно за чување ланца. Меморија која је потребна за тест мрежу је 40GB. Програм се синхронизује са остатком мреже при сваком покретању, односно ажурира копију ланца коју чува на диску. У случају првог покретања синхронизација траје до неколико сати на тест мрежи, док на главној мрежи синхронизација може да потраје данима због прикупљања информација о свим блоковима у мрежи. Дужина трајања синхронизације зависи од брзине корисничке интернет конекције, као и од самих перформанси рачунара корисника. Приказ првог покретања програма и његове синхронизације са мрежом дат је на слици 7.3. Bitcoin Core прво преузима заглавља блокова тренутно најдужег ланца,



Слика 7.3: Синхронизација блокова при првом покретању програма

а потом и целе блокове најдужег ланца. Разлог преузимања података у овом редоследу је што програм тиме игнорише одбачене гране привременог гранања (тачка

4.3.1). Овакав редослед преузимања је представљен у верзији протокола 0.10.0. BitcoinCore синхронизацију обавља у односу на чворове са којима се аутоматски упарује (тачка 5.1). Листа чворова се може видети „Window" → „Колеге". Прво покретање нема за улогу само преузимање ланца блокова, већ програм за сваки преузети блок проверава исправност криптографске хеш вредности блока и да ли та вредност задовољава доказ о раду (тачка 4.1), као и низ других провера објашњених у глави 3. У случају потребе за реинсталацијом програма ланац не мора да се преузима изнова у потпуности, јер локација[14] где се чува ланац није везана за локацију где се налази инсталација програма.

Након што је синхронизација завршена, приступићемо креирању новог новчаника („Фајл"→„Направи новчаник"). Кориснику је представљено обавезно поље за назив новчаника као и опција за шифровање фајла *wallet.dat* у ком ће се чувати приватни кључ. Програм аутоматски генерише приватни кључ при креирању новог новчаника. Уколико одабере опцију шифровања новчаника, корисник дефинише лозинку која се користи као улаз криптографске хеш функције SHA-512[15]. Резултујућа хеш вредност се касније користи као један од улаза функције EVP_BytesToKey библиотеке OpenSSL[16]. Резултат ових корака је кључ који се користи за шифровање основног кључа (енг. master key) који је претходно генерисан на псеудо-случајан начин. Улога основног кључа је да шифрује приватни кључ новчаника. Препорука је да лозинка садржи више од 8 речи. У овом експерименту назив новчаника је „ТестНовчаник", а лозинка је „тест".

У секцији „Прими" могуће је креирати нову Биткоин адресу. За различите трансакције у оквиру истог новчаника, препоручује се креирање различитих Биткоин адреса. Из тог разлога постоји неколико начина да се кориснику олакша разазнавање адреса етикетирајући их, везујући поруку за њих или уписујући износ биткоина као назнаку. Свака од наведених опција није обавезна и не утиче на резултат генерисања јавног кључа, а потом и адресе. Приказ генерисања Биткоин адресе дат је на слици 7.4. Након притиска на дугме „Направи нову адресу за примање" кориснику се приказује прозор са информацијама о јавној Биткоин адреси (Address), њен приказ у формату линка (URI) и QR кôда. Приказ резултата овог корака дат је на слици 7.4.

Овако генерисану адресу искористићемо за примање биткоина на тест мрежи. Постоји велики број различитих сајтова који донирају биткоине и захтеваћемо донацију биткоина са једног од њих[17] у износу од 0.00022BTC (22000 сатошија). Број чворова који је прихватио блок у ком се налази обављена трансакција може бити испраћен у секцији „Трансакције", али тек у тренутку када тај блок буде убачен у ланац, биткоини постају доступни. Иако су биткоини одмах доступни за даље трошење, препорука је да се сачека нових 6 блокова после блока у ком се налази трансакција како би се онемогућио Финијев напад (тачка 4.5.1). Из тог разлога трансакција има статус „непотврђена", док број чворова не пређе 6. Након тога статус прелази у „потврђена".

Идентификатор трансакције TXID (тачка 3.1) може бити употребљен на сајту[18] за претрагу трансакције у ланцу блокова. Приказ експерименталне трансакције са сајта дат је на слици 7.5. На слици се види и то да је донатор биткоина унутар трансакције са нама сместио и додатни излаз ка некој другој Биткоин адреси. Крипто-

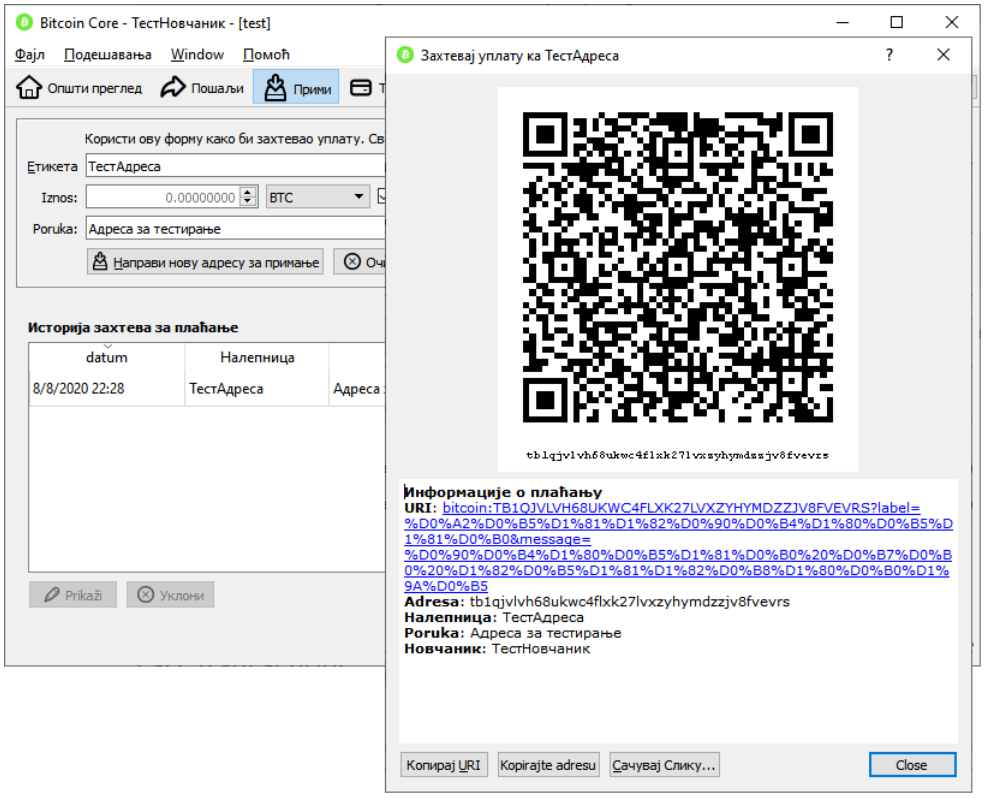
[14] Ланац блокова се чува на локацији %APPDATA%/Bitcoin, уколико се користи Windows оперативни систем.

[15] `https://en.bitcoinwiki.org/wiki/SHA-512`

[16] `https://www.openssl.org/`

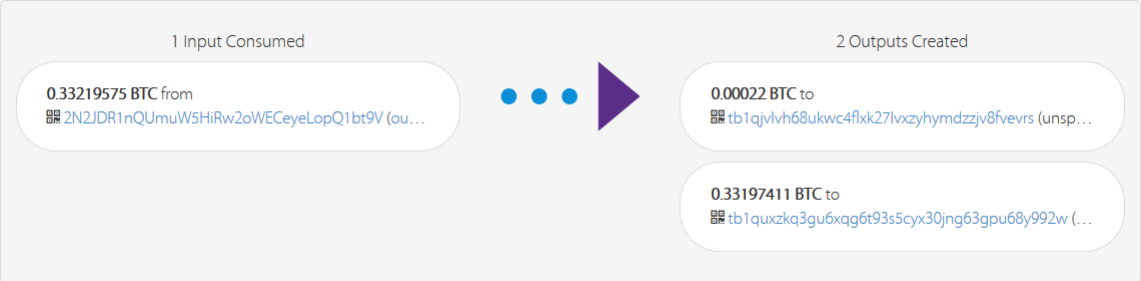
[17] `https://bitcoinfaucet.uo1.net/`

[18] `https://live.blockcypher.com/`



Слика 7.4: Креирање Биткоин адресе

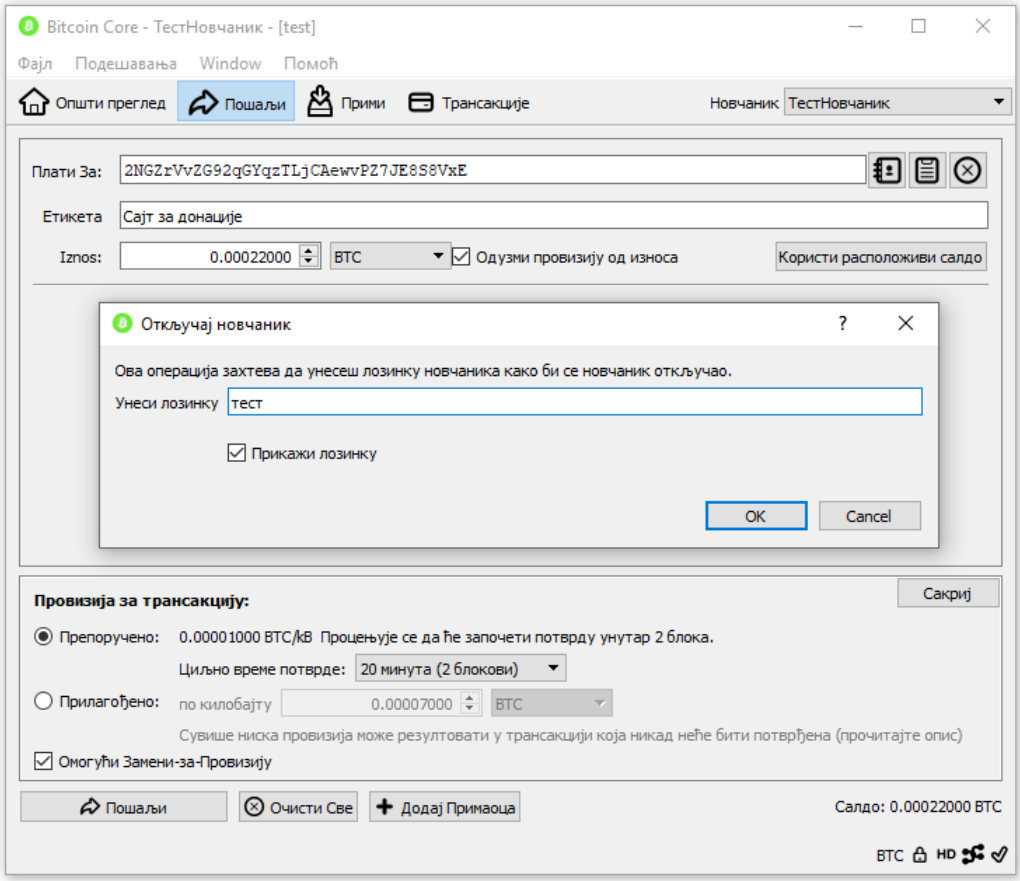
графска хеш вредност блока у оквиру ког је смештена експериментална трансакција је 000000000000001ec8d0f14416dfd4f4109b0f383e39095596c4f2d00d394e67, док је висина блока 1 806 065. У блоку се поред експерименталне трансакције налази и још других 95 трансакција.



Слика 7.5: Преглед обављене трансакције

Приступићемо сада слању биткоина са наше адресе назад ка сајту који нам је биткоине и донирао. Унутар секције „Пошаљи” кориснику је презентовано поље које попуњава Биткоин адресом на коју шаље биткоине. Корисник може етикетирати адресу унутар поља „Етикета” и на тај начин разликовати адресе које је претходне користио за уплату биткоина, у даљем коришћењу. У оквиру истог прозора налази се поље које се попуњава вредношћу биткоина који се шаљу. У нашем случају то је 0.00022BTC. На дну прозора налази се секција „Провизија за трансакцију” у оквиру које се контролише провизија на трансакцију (тачка 4.2.1). Програм кориснику нуди могућност процене препоручене величине провизије уколико одабере опцију „Препоручено”. Уколико се одабере опција „Прилагођено” корисник контролише износ провизије уз напомену да време, за које ће трансакција бити убачена у блок, може да зависи од величине одабране провизије. Уколико корисник иза-

бере премалу провизију, трансакција може остати заглављена у бази неповрђених трансакција. Из тог разлога програм нуди опцију „Омогући замени-за-Провизију” [9] која кориснику нуди могућност замене трансакције истом, али са задатом већом провизијом. Након притиска на дугме „Пошаљи”, корисник потврђује своје власништво над новчаником уколико је шифрован. Приказ описаног корака дат је на слици 7.6.



Слика 7.6: Слање биткоина

Програм не нуди директан приказ приватног кључа из сигурносних разлога, што отежава власнику да запише свој приватни кључ (на пример) на папир. До приватног кључа долази се коришћењем програма bitcoin-cli (тачка 7.1.4) чије се функционалности позивају кроз конзолу („Window” → „Конзола”). Конзолни приказ програма прво обавештава корисника да буде пажљив, јер постоје команде које могу да угрозе безбедност новчаника (и биткоина везаних за њега). Као што је већ речено приватни кључ је сачуван у фајлу *wallet.dat* који је шифрован претходно дефинисаном лозинком (у нашем случају „тест”). Из тог разлога, неопходно је привремено откључати новчаник командом *walletpassphrase*, која за аргументе има:

- passphrase; лозинка којом је шифрован новчаник,
- timeout; време мерено у секундама за које ће оригинална вредност приватног кључа бити учитана у меморију.

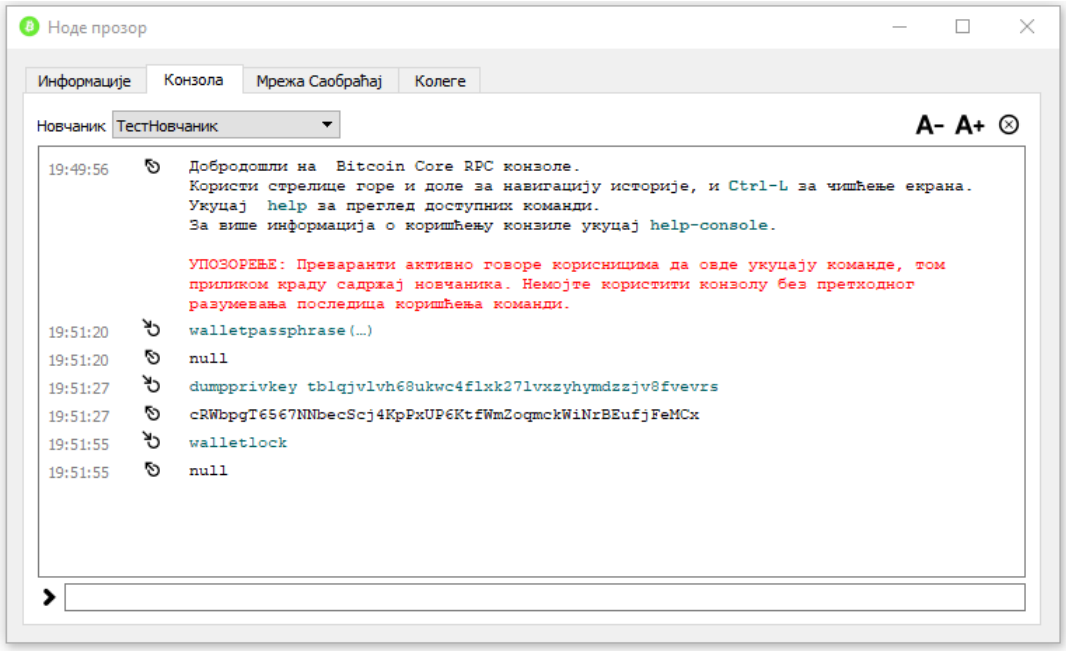
Пример покретања команде је:

>walletpassphrase тест 1000

Команда је успешно позвана уколико конзола испише *null*. Командом *dumpprivkey* добијамо вредност приватног кључа у WIF формату (Wallet Import Format)[19]. Тај формат је погодан за касније увожење приватног кључа (и новчаника за који је везан), уколико дође до брисања или реинсталације BitcoinCore програма. Команда *dumpprivkey* као аргумент прима вредност Биткоин адресе новчаника (једну од адреса уколико их новчаник има више). Изглед позива команде у нашем случају дат је у наставку:

>dumpprivkey tb1qjvlvh68ukwc4flxk27lvxzyhymdzzjv8fvevrs

Након исписивања приватног кључа у конзоли, новчаник се може поново закључати командом *walletlock*. Пример покретања описаних команди, као и резултујући кључ у WIF формату приказани су на слици 7.7.



Слика 7.7: Извожење приватног кључа

Оригинална вредност приватног кључа може се добити тако што се вредност кључа у WIF формату декодира користећи Base58 (тачка 2.6). Приказ Base58 декодираног кључа у WIF формату дат је у наставку (резултат декодирања је хексадекадна вредност):

cRWbpgT6567NNbecScj4KpPxUP6KtfWmZoqmckWiNrBEufjFeMCx

$$\downarrow \textit{ Base58 декодирање } \downarrow$$

ef753d7b5468fb3cf849915c02109652bf31dfc9fa8c8dc29e0614c39566cec81801e8a450ed

Након тога приступамо уклањању 4 бајта са краја записа (која се користе као контролна сума) и једног бајта са почетка записа. Уколико вредност кључа у WIF формату на тест мрежи почиње карактером 'c' или 9 (као у нашем случају), то је назнака да је јавни кључ компримован и потребно је уклонити још један бајт са краја записа. Приказ овог корака дат је у наставку.

~~ef~~753d7b5468fb3cf849915c02109652bf31dfc9fa8c8dc29e0614c39566cec81~~801e8a450ed~~

[19] https://en.bitcoin.it/wiki/Wallet_import_format

Оригинална вредност приватног кључа у хексадекадном формату нашег новчаника „ТестНовчаник” је:

753D7B5468FB3CF849915C02109652BF31DFC9FA8C8DC29E0614C39566CEC818.

8 Закључак

У раду су представљене технологије и криптографска решења на којима се заснива систем Биткоин. Преглед кôда у оквиру главе 7 даје могућност за боље разумевање имплементације протокола и самим тим будући рад може да буде посвећен побољшању функционисања система. Један од проблема на који будуће истраживање може да се фокусира је скалабилност. Скалабилност је главни проблем са којим се технологија ланаца блокова суочава и управо је то један од разлога зашто Биткоин, као један од пројеката те технологије, није део наше свакодневнице.

Поред недостатака скалабилности Биткоина, систем и даље опстаје са просечно обрађених 300 000 трансакција дневно. Да ли ће се тако наставити у будућности, уколико број корисника, (као и број трансакција) буде још више растао, а услови функционисања не буду круцијално промењени, није извесно.

Литература

[1] Dobbertin H., Bosselaers A., Preneel B. Ripemd-160: A strengthened version of ripemd. in: Gollmann d. (eds) fast software encryption. fse 1996. lecture notes in computer science, vol 1039. springer, berlin, heidelberg. https://doi.org/10.1007/3-540-60865-6_44, 1996.

[2] BIPs. `https://github.com/bitcoin/bips`.

[3] Kristian Bjoernsen. Koblitz curves and its practical uses in bitcoin security, 2015.

[4] Chaum D. Blind signatures for untraceable payments. in: Chaum d., rivest r.l., sherman a.t. (eds) advances in cryptology. springer, boston, ma. https://doi.org/10.1007/978-1-4757-0602-4_18, 1983".

[5] Joseph Poon, Thaddeus Dryja. The bitcoin lightning network:scalable off-chain instant payments. page 59, 01 2016.

[6] Eyal I., Sirer E.G. Majority is not enough: Bitcoin mining is vulnerable. in: Christin n., safavi-naini r. (eds) financial cryptography and data security. fc 2014. lecture notes in computer science, vol 8437. springer, berlin, heidelberg. https://doi.org/10.1007/978-3-662-45472-5_28. 2014.

[7] Pedro Franco. *Understanding Bitcoin: Cryptography, Engineering and Economics*. Wiley Finance Series, 2015.

[8] Ethan Heilman. One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner (poster abstract). volume 8438, pages 161–162, 03 2014.

[9] Opt in Full Replace-by Fee Signaling. `https://github.com/bitcoin/bips/blob/master/bip-0125.mediawiki`.

[10] Jayeeta Majumder. Dictionary attack on md5 hash. 05. 2020.

[11] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[12] Cristopher Paar, Jan Perlzl. Understanding cryptography, 2010.

[13] Certicom Research. Sec 2: Recommended elliptic curve domain parameters.

[14] Bitcoin wikipedia. `https://en.bitcoin.it/wiki/BIP_0016`, 04 2010.

[15] Bitcoin wikipedia. `https://en.bitcoinwiki.org/wiki/Bitcoin_Improvement_Proposals`, 04 2010.

[16] Eric Lombrozo, Johnson Lau, Pieter Wuille. `https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki`, 12 2015.

[17] Qiheng Zhou, Huawei Huang, and Zibin Zheng. Solutions to scalability of blockchain: A survey, 01 2020.