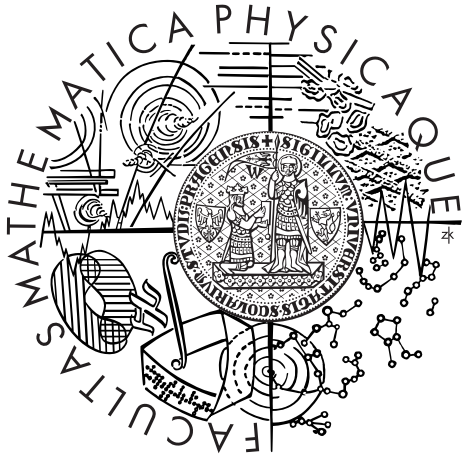
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Lucie Deptová

Virtuální měna Bitcoin

Katedra algebry

Vedoucí bakalářské práce:	Mgr. Milan Boháček
Studijní program:	Matematika
Studijní obor:	Matematické metody informační bezpečnosti

Praha 2014

Děkuji vedoucímu své bakalářské práce za jeho trpělivost, ochotu a věnovaný čas.

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 19. května 2014 Lucie Deptová

Název práce: Virtuální měna Bitcoin

Autor: Lucie Deptová

Katedra: Katedra algebry

Vedoucí bakalářské práce: Mgr. Milan Boháček, Katedra algebry

Abstrakt: Bitcoin platební systém a zároveň virtuální měnu spustil poprvé Satoshi Nakamoto v roce 2009. Stejně jako ostatní digitální měny musí Bitcoin zaručit určitou míru bezpečnosti. Jeho povinností je například zabránit tomu, aby bylo jednou „mincí“ placeno dvakrát, dále je potřeba zajistit anonymitu plateb. Čím se však od většiny ostatních virtuálních měn liší, je absence jakékoliv důvěryhodné autority, která by výše uvedené vlastnosti obstarávala. Obsahem této práce je popis struktury a vlastností základních stavebních jednotek tohoto platební systému a zároveň vysvětlení mechanismů, kterými se Bitcoin vypořádává s důsledky své decentralizovanosti. Práce poskytuje čtenáři komplexní a dostatečně detailní informace, které jsou jinak dostupné pouze v oficiálním zdrojovém kódu bitcoin klienta nebo jsou roztroušené ve velkém množství jednotlivých článků.

Klíčová slova: Bitcoin, elektronická měna

Title: Bitcoin digital currency

Author: Lucie Deptová

Department: Department of Algebra

Supervisor: Mgr. Milan Boháček, Department of Algebra

Abstract: In 2009 Satoshi Nakamoto started the electronic payment system and virtual cash Bitcoin for the first time. Bitcoin has to guarantee some level of security as well as other digital currencies. It's necessary to prevent double-spending or it's required to provide anonymity of payments. The diference between Bitcoin and most of the other virtual currencies is absence of any trusted party which would provide demands mentioned above. In this paper we describe the structure and properties of basic elements of this payment system. In the mean time we explain methods how to deal with the fact that Bitcoin is completely decentralized. The paper gives complex and detailed information which you can find only in official source code of bitcoin client or in many separated and particular articles.

Keywords: Bitcoin, electronic cash

Obsah

Úvod

Svět informačních technologií se vyvíjí rychlým tempem a snaží se zajišťovat stále více služeb, na které jsme v každodenní realitě zvyklí. Už nemusíme chodit na poštu s dopisy, stačí poslat e-mail. Výjimkou není ani open source platební systém a současně virtuální měna Bitcoin, kterou se tato práce zabývá. Místo papírových bankovek a kovových mincí si představte bitcoiny uložené ve virtuální peněžence. Bitcoin se nejenže snaží nahradit klasickou měnu a poskytnout i komfort plateb, které můžete provádět po internetu z počítače či telefonu. Oproti dříve navrženým virtuálním měnám si navíc klade za cíl decentralizovanost. To znamená, že do chodu měny nemůže nikdo zasahovat ani jej řídit. Další ambicí Bitcoinu je anonymita plateb, jako je tomu u běžných bankovek. To, co u klasických měn zajišťuje banka, musí síť bitcoin klientů ošetřovat sama. Pro zajištění všech bezpečnostních a autoregulačních požadavků jsou použity kryptografické mechanismy jako je digitální podpis s využitím eliptických křivek, kryptografické hešovací funkce atd.

Platební systém Bitcoin byl spuštěn v roce 2009 a zprvu se do něj zapojoval jen úzký okruh nadšenců. Postupně se však stal komoditou, o kterou mají zájem nejen obchodníci na burzách, ale i vlády jednotlivých zemí a v neposlední řadě i širší veřejnost. S konceptem tohoto platebního systému přišel Satoshi Nakamoto, anonym, jehož identitu se nikdy nepodařilo zjistit. Nejprve se podílel i na samotné implementaci, ale poté se zcela odmlčel.

Pokud se někdo snaží proniknout do tajů bitcoinového světa, může si zcela jistě přečíst spoustu popularizačních či povrchních článků, které mu dají jistou intuici, jaký je princip této měny. Jestliže však někdo stojí o preciznější znalosti a podrobnosti, naráží na množství jednotlivých neoficiálních článků, podávajících kusé informace o partikulárních prvcích bitcoin systému, do kterých je zprvu těžké proniknout. Jediný oficiální zdroj informací je zdrojový kód bitcoin klienta (pracovali jsme s kódem z [1] ve verzi dostupné 9.9.2013). Účelem práce je zpracovat tento zdrojový kód a doplňkové články do celistvého a dostatečně detailního popisu Bitcoin platebního systému a jeho protokolu. Cílovým čtenářem je každý, kdo se chce dozvědět, jak fungují základní mechanismy Bitcoinu a jaké mají vlastnosti.

Kapitola 1 představuje malý slovníček pojmů, které se v textu vyskytují, a zavádí značení, jehož se budeme nadále držet. Kapitola 2 se zabývá těmi matematickými a kryptografickými pojmy a jejich vlastnostmi, jež jsou pro Bitcoin důležité a vyskytují se dalším v textu. Kapitola 3 popisuje, k čemu slouží a jak vzniká bitcoin adresa, tedy jakési uložiště vámi vlastněných bitcoinů. Podíváme se i na to, jaká je pravděpodobnost, že by si dva lidé vygenerovali totožnou adresu. Kapitola 4 se zabývá vnitřní strukturou transakcí i skriptovacím systémem, který zajišťuje to, že s vašimi bitcoiny nemůže nakládat nikdo jiný než vy. Poslední kapitola 5 pak popisuje, co je to blok transakcí a jak se udržuje bezpečnost bitcoin sítě.

1. Slovníček a značení

V následující kapitole nejprve uvedeme slovníček pojmů, které se v práci objevují. Za ním zavedeme značení, jehož se budeme v textu držet.

1.1 Slovníček

- `peer-to-peer` síť je taková síť, kde klienti vzájemně komunikují přímo (nikoliv přes server) a každý je zároveň příjemce i poskytovatel dat.
- (Bitcoin) klient je každý, kdo se účastní komunikace a výměny dat v bitcoin síti.
- (Bitcoin) síť je peer-to-peer síť všech aktivních klientů.
- (Bitcoin) peněženka je software, který vám umožňuje vlastnit bitcoiny, vytvářet a přijímat transakce. V peněžence se ukládají soukromé klíče vašich adres, proto je potřeba ji šifrovat a chránit před krádeží.
- Testnet je alternativní block chain (viz sekce 5.2.4 v Kapitole 5) sloužící k experimentům a testování, kde se nepracuje s opravdovými bitcoiny.
- `BIP`, neboli Bitcoin Improvement Proposal, je dokument rozšiřující Bitcoin protokol. Jednotlivé jeho verze jsou číslovány, např. `BIP 005`, a o jeho přijetí rozhoduje to, zda ho začne používat většina bitcoin klientů.
- Výpočetní síla klienta (resp. části sítě) je počet hešů funkce SHA-256, které je klient (resp.část sítě) schopen provést za sekundu.

1.2 Značení

- Zápis $(\text{číslo})_{16}$ bude značit číslo zapsané v hexadecimální soustavě. Tedy např. $(\text{D61967F6})_{16}$=$D61967F6$.
- Symbol $\wedge$ značí konjunkci.
- Symbol $||$ značí konkatenaci dvou řetězců.
- Zápis $\text{číslo}_1 \ll \text{číslo}_2$ značí, že $\frac{\text{číslo}_1}{\text{číslo}_2} < 2^{-9}$.
- Pojmem big endian budeme označovat takovou reprezentaci čísla, kde nejvýznamnější cifra je na začátku čísla. Naopak pojem little endian označuje reprezentaci, kdy nejvýznamnější cifra je na konci čísla. Např. číslo $2 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$ lze reprezentovat v big endian jako 235, v little endian jako 532.
- Zkratka BTC znamená bitcoin.
- Buď $a, b \in \mathbb{Z}$. Symbolem $\text{NSD}(a, b)$ budeme značit největší společný dělitel a a b.

2. Teoretická kapitola

V této kapitole zadefinujeme matematické a kryptologické pojmy, které se v dalších kapitolách používají, a uvedeme jejich základní vlastnosti. Zdrojem pro sekce 2.1 a 2.3 byla [14] a [15], pro sekci 2.2 je to [19] a [12]. Pro sekci 2.4 jsme čerpali z [11] a pro sekci 2.5 z [4] a z [13].

2.1 Hešovací funkce

Definice 1. *Mějme X spočetnou množinu a Y konečnou množinu. Pak funkci $H : X \to Y$ nazveme (kryptografickou) hešovací funkcí, splňuje-li následující podmínky:*

1. *$\forall x \in X$ je snadné spočítat $H(x)$.*
2. *$\forall y \in Y$ je těžké najít $x \in X : H(x) = y$ (vlastnost preimage resistance).*
3. *$\forall x \in X$ je těžké najít $x' \in X, x' \neq x : H(x') = H(x)$ (vlastnost second preimage resistance).*
4. *Je těžké najít $x, x' \in X : H(x) = H(x')$ (vlastnost collision resistance).*

Funkci H nazveme jednosměrnou kompresní funkcí, jestliže množina X je konečná.

Účelem této práce není studium hešovacích funkcí, proto se zde nebudeme zabývat tím, co přesně znamená fakt, že je něco „těžké najít“. Hešovací funkce obecně slouží k tomu převést různě velká data na blok předem určené velikosti, který představuje otisk původních dat. U kryptografických hešovacích funkcí pak po takovém otisku požadujeme, aby se z něj nedalo vyčíst nic o původních datech (funkce by se měla chovat nahodile) a zároveň aby pro dva různé vstupy nevznikaly ty samé otisky. Jednoduše řečeno by taková funkce měla být jednoduchá pro výpočet a náročná pro nalezení inverzu.

2.1.1 Merkle-Damgårdovo schéma

Merkle-Damgårdovo schéma je jedna z metod, jak zkonstruovat kryptografickou hešovací funkci. Principem této metody je převedení problému výpočtu kryptografické hešovací funkce na problém výpočtu jednosměrné kompresní funkce. Označme písmenem f nějakou jednosměrnou kompresní funkci, která na vstupu očekává blok délky k, a vstupní zprávu, již chceme zhešovat. Funkci f budeme opakovaně aplikovat vždy na kombinaci vstupní zprávy M a výsledku předchozího kroku. Na konci celého procesu získáme heš H vstupní zprávy M.

Mějme tedy vstupní zprávu M rozdělenou na bloky $M = (m_1, m_2, ..., m_n)$, kde $|m_1| = |m_2| = ... = |m_{n-1}| = k$. Hodnota k je pevná a určená standardem konkrétní hešovací funkce. Z posledního bloku m_n, který obecně nemusí mít délku k, se vytváří tzv. padding. Padding slouží k tomu, aby doplnil poslední blok zprávy na požadovanou délku a zároveň neporušil vlastnosti, které má kryptografická

hešovací funkce mít. V Merkle-Damgårdově konstrukci se používá tzv. *length padding*, který na konec zprávy přidá blok $m'_n = (m_n||1||00...0||l(M))$, kde $l(M)$ je délka zprávy M a počet nul v řetězci bezprostředně před $l(M)$ je takový, aby $|m'_n| = k$. Může se však stát, že $|m'_n| > k$ kvůli velikosti $l(M)$. V takovém případě položíme $m'_n = (m_n||1||0||...||0)$, $|m'_n| = k$ a dále musí být přidán blok $m_{n+1} = (00...0||l(M))$, kde počet nul je takový, aby $|m_{n+1}| = k$. Pokud $|m_n| = k$, pak $m'_n := m_n$ a přidáme blok $m_{n+1} = (1||00...0||l(M))$, počet nul je opět takový, aby $|m_{n+1}| = k$.

Přidáním $l(M)$ na konec posledního bloku se zabrání některým útokům na hešovací funkce.

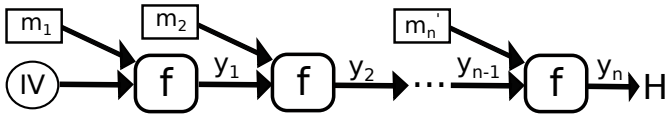
Výpočet heše H zprávy M, který označíme $H(M)$, pak probíhá následujícím způsobem:

$$\begin{aligned} y_1 &= f(IV, m_1) \\ y_i &= f(y_{i-1}, m_i)\ \forall i \in \{2, .., n-1\} \\ y_n &= f(y_{n-1}, m'_n) =: H(M) \end{aligned}$$

Pokud byl přidán blok m_{n+1}, pak:

$$\begin{aligned} y_1 &= f(IV, m_1) \\ y_i &= f(y_{i-1}, m_i)\ \forall i \in \{2, .., n-1\} \\ y_n &= f(y_{n-1}, m'_n) \\ y_{n+1} &= f(y_n, m_{n+1}) =: H(M) \end{aligned}$$

kde IV je inicializační vektor, jenž je daný specifikací té které hešovací funkce. Schéma hešovaní je uvedeno na obrázku 2.1.



Obrázek 2.1: Merkle-Damgårdovo schéma

V bitcoin protokolu se používají dvě hešovací funkce, a to SHA-256 a také RIPEMD-160.

Definice 2. SHA-256 : $\{0,1\}^* \to \{0,1\}^{256}$ *je kryptografická hešovací funkce vytvořená pomocí Merkle-Damgårdova schématu a popsaná ve standardu FIPS 180-2 [17].*

Definice 3. RIPEMD-160 : $\{0,1\}^* \to \{0,1\}^{160}$ *je kryptografická hešovací funkce vytvořená pomocí Merkle-Damgårdova schématu a publikovaná v roce 1996 na konferenci FSE (Fast Software Encryption) autory H.Dobbertin, A.Bosselaers, B.Preneel [9].*

2.2 Algebra

V dalším textu předpokládáme, že čtenář zná definici (abelovské) grupy, viz str.76 v [19]. Grupu $\mathcal{G} = (G, \cdot, ^{-1}, 1)$ budeme značit písmenem G a říkáme, že G je v multiplikativním zápisu. Pokud jsou na G definovány operace $+,-,0$ splňující podmínky uvedené v definici grupy, říkáme, že G je v aditivním zápisu.

Definice 4. *Buď G grupa $(G, \cdot, ^{-1}, 1)$. Řekneme, že G je cyklická, jestliže $\exists\, \alpha \in G$ $\forall\, a \in G : a = \alpha^k$ pro nějaké $k \in \mathbb{Z}$. Prvku α říkáme generátor G. Dále definujeme řád prvku $a \in G$ jako nejmenší $n \in \mathbb{N}$ takové, že $a^n = 1$.*

2.2.1 Eliptické křivky

Teorie eliptických křivek je dosti rozsáhlá, proto se v jejím popisu omezíme na následující fakta a vlastnosti (převzaty z [12]), které nebudeme dokazovat.

Buď $p > 3$ prvočíslo, eliptická křivka E nad tělesem $\mathbb{F}_p$ je křivka definovaná rovností

$$y^2 = x^3 + ax + b \tag{2.1}$$

kde $a, b \in \mathbb{F}_p$ a $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. Uvažme množinu $E(\mathbb{F}_p)$ jakožto množinu bodů (x, y), $x, y \in \mathbb{F}_p$ splňujících rovnici 2.1 spolu s bodem 0 nazývajícím se bod v nekonečnu. Na množině $E(\mathbb{F}_p)$ zavedeme operaci binární $+$, unární $-$ a konstantu 0, čímž dostaneme grupu bodů na eliptické křivce. Operace definujeme následujícím způsobem pro $P = (x_1, y_1), Q = (x_2, y_2) \in E(\mathbb{F}_p)$:

1. $P + 0 = 0 + P = P$
2. $(x_1, y_1) + (x_1, -y_1) = 0$
3. Nechť $P \neq \pm Q$, pak $P + Q = (x_3, y_3)$, kde

$$\begin{aligned} x_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \\ y_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right) \cdot (x_1 - x_3) - y_1 \end{aligned}$$

4. Nechť $P \neq -P$, pak $2P = (x_3, y_3)$, kde

$$\begin{aligned} x_3 &= \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \\ y_3 &= \left(\frac{3x_1^2 + a}{2y_1}\right) \cdot (x_1 - x_3) - y_1 \end{aligned}$$

2.3 Digitální podpis

Digitální podpis je soubor dat, který v určitém smyslu zastupuje klasický podpis rukou a splňuje následující kritéria. Prvním je autentizace, to znamená, že musí být prokazatelné, od koho podpis pochází. Dále požadujeme nepopiratelnost. Jelikož tvůrce podpisu je jediný, kdo takový podpis může vydat, nemůže zapřít podpis, který mu patří. A nakonec od podpisu očekáváme zajištění integrity. To znamená, že si chceme být jisti tím, že zpráva, po tom, co byla podepsána, nebyla nijak poškozena ani úmyslně změněna.

Digitální podpis (ostatně jako každá aplikace asymetrické kryptografie) pracuje se dvěma klíči – soukromým a veřejným. Jedině ten, kdo vlastní soukromý klíč, je schopen podepisovat a musí si proto klíč patřičně chránit. Kdokoliv, kdo by získal jeho soukromý klíč, by se totiž za něj mohl vydávat a podepisovat. Veřejný klíč je naopak znám komukoliv a slouží všem okolo, aby si mohli ověřit platnost podpisu.

My zde představíme digitální podpis *ElGamal*, se kterým přišel Taher Elgamal v roce 1985 a publikoval jej v [10]. Zobecněný algoritmus ElGamal s grupou G (v multiplikativním zápisu) řádu n a generátorem α má následující podobu (popis je převzat z [14]):

Mějme dvě strany A, B a zprávu M. A chce podepsat M a poslat podepsanou zprávu straně B, ta podpis ověří. Dále mějme H kryptografickou hešovací funkci.

1. *(Algoritmus generování klíče)* Pro vygenerování klíče provede A následující:
   (a) Zvolí náhodně uniformně celočíselné a takové, že $1 \leq a \leq n-1$ a spočítá $y = \alpha^a$.
   (b) Veřejný klíč A je dvojice (α, y) spolu s popisem násobení v grupě G, soukromý klíč je a.
2. *(Algoritmus podepisování)* A podepíše zprávu M takto:
   (a) Zvolí náhodně uniformně celočíselné tajné k takové, že $1 \leq k \leq n-1$ a $\mathrm{NSD}(k, n) = 1$.
   (b) Spočítá $r = \alpha^k$.
   (c) Spočítá $H(M), H(r)$.
   (d) Spočítá $s = k^{-1}(H(M) - aH(r)) \pmod{n}$.
   (e) Podpis strany A zprávy M tvoří dvojice (r, s).
3. *(Algoritmus ověřování)* B ověří podpis (r, s) zprávy M pomocí veřejného klíče (α, y) strany A následovně:
   (a) Spočítá $H(M), H(r)$.
   (b) Spočítá $v_1 = y^{H(r)} r^s$ a $v_2 = \alpha^{H(M)}$.
   (c) Podpis je ověřený právě tehdy, když $v_1 = v_2$.

Pokud napíšeme, že se nějaký prvek volí náhodně uniformně, pak to znamená, že jeho volba proběhne náhodně a že každý prvek bude vybrán se stejnou pravděpodobností.

Pro zachování bezpečnosti podpisového schématu je nutné, aby se prvek k z kroku 2a použil k podpisu právě jedné zprávy. Stačí totiž uvážit podpisy dvou zpráv M_1, M_2, které vznikly s použitím stejného k. Tímto budeme mít dvě rovnosti z kroku 2d se stejným k. Neznámými v této soustavě dvou lineárních rovnic jsou k a a, které snadno spočítáme, čímž zjistíme soukromý klíč.

Lemma 1. *Algoritmus ověřování ElGamal popsaný v bodě 3 je správný.*

Důkaz. Buď (α, y) veřejný klíč, a k němu příslušný soukromý klíč a (r, s) podpis nějaké zprávy M tímto soukromým klíčem. Jelikož (α, y) je veřejný klíč a (r, s) je podpis, musí platit:

$$\begin{aligned} y &= \alpha^a && (2.2)\\ r &= \alpha^k && (2.3)\\ s &\equiv k^{-1}(H(M) - aH(r)) \pmod{n} && (2.4) \end{aligned}$$

kde k splňuje vlastnosti z 2a. Díky tomu platí:

$$\exists k^{-1} : 1 \leq k^{-1} \leq n-1 \ \wedge \ k * k^{-1} \equiv k^{-1} * k \equiv 1 \pmod{n} \qquad (2.5)$$

Pro ověření (r, s) strana B spočítá v_1, kam může dosadit z rovností 2.2, 2.3, 2.4 a následně využít 2.5:

$$\begin{aligned} v_1 &= y^{H(r)} r^s = \\ &= \alpha^{aH(r)} \alpha^{k*k^{-1}(H(M)-aH(r))} \pmod{n} = \\ &= \alpha^{aH(r)+H(M)-aH(r)} = \\ &= \alpha^{H(M)} = \\ &= v_2 \end{aligned}$$

Tedy (r, s) je podpis zprávy M pomocí klíče a právě tehdy, když $v_1 = v_2$.

❑

Vzhledem k následné implementaci a praktickému využití by bylo dobré, aby měl podpis fixní délku. V našem schématu je podpis $(r, s) \in G \times \mathbb{Z}_n$.

Americký institut pro standardizaci roku 1993 vydal standard FIPS–186, ve kterém na Elgamalův podpis aplikoval následující změny:

- Za grupu G se zvolí $\mathbb{Z}_p^*$, kde p je velké prvočíslo.
- Řád n generátoru α je prvočíslo z intervalu $\langle 2^{159}, 2^{160} \rangle$.
- Krok 2b z popisu *ElGamal* se modifikuje na $r = \alpha^k \pmod{n}$. Tím se zajistí, že $(r, s) \in \mathbb{Z}_n \times \mathbb{Z}_n$. Proto už v následujících krocích algoritmu není potřeba pracovat s $H(r)$, stačí použít samotné r. To však znemožní použít r při ověření podpisu, které se tím změní následujícím způsobem.
- Ověření podpisu:
  1. Spočítá $H(M)$ a $w = s^{-1} \pmod{n}$.
  2. Spočítá $u_1 = w \cdot H(M) \pmod{q}$ a $u_2 = r \cdot w \pmod{q}$.

3. Spočítá $v = (\alpha^{u_1} y^{u_2} \pmod p) \pmod q$.
4. Podpis je ověřený právě tehdy, když $v = r$.

Takto upravený ElGamalův digitální podpis se nazývá *DSA* (Digital Signature Algorithm).

O pár let později byla provedena další změna, konkrétně za grupu G se zvolila grupa bodů na eliptické křivce. Takovýto digitální podpis je znám pod názvem *ECDSA* (Eliptic Curve Digital Signature Algorithm). Ten v kroku 2b za r zvolí první souřadnici součinu $k \cdot \alpha$.

Bitcoin protokol používá pro digitální podpis *ECDSA*, jako grupu G volí grupu bodů na eliptické křivce `secp256k1` (viz [8]) nad konečným tělesem $\mathbb{F}_p$, kde

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

Parametry této křivky jsou:

$$\begin{aligned} a &= 0 \\ b &= 7 \end{aligned}$$

Generátor α grupy G je bod na křivce řádu
$n = 115792089237316195423570985008687907852837564279074904382605163141518161494337$
a souřadnicemi (x, y), kde
$x = 55066263022277343669578718895168534326250603453777594175500187360389116729240$
$y = 32670510020758816978083085130507043184471273380659243275938904335757337482424$

V poslední době se objevují názory (viz blog Daniela Bernsteina [6]), že grupy bodů na eliptických křivkách tvaru 2.1 nejsou vhodnou volbou z důvodu jejich náchylnosti podlehnout útokům postranními kanály. Nedávno byl prezentován útok (viz [5]) na Bitcoin využívající této slabiny, který vedl k získání *ECDSA* soukromého klíče uvnitř peněženky pomocí změření doby výpočtu u zhruba dvou set podpisů.

2.4 Pravděpodobnost

V celém textu je několikrát počítána pravděpodobnost jevů. Vždy však pracujeme s konečnými množinami, proto si vystačíme s následující definicí pravděpodobnosti (převzata z [11]).

Definice 5. *Buď X konečná množina a $\mathcal{A}$ množina všech podmnožin X. Pravděpodobnost definujeme jako funkci $P : \mathcal{A} \to \langle 0, 1\rangle$ splňující pro $\forall A, A_1, ..., A_k \in \mathcal{A}$ pro $k \in \mathbb{N}$ následující vlastnosti:*

- $P(A) \geq 0$
- $P(X) = 1$, $P(\emptyset) = 0$
- $P(\bigcup_{i=1}^{k} A_i) = \sum_{i=1}^{k} P(A_i)$, *jsou-li A_i po dvou disjunktní.*

Každé $A \in \mathcal{A}$ nazýváme jev a $A' = X \setminus A$ nazýváme doplňkový jev k jevu A.

2.4.1 Narozeninový paradox

Představme si, že máme n kuliček, které umísťujeme do m přihrádek.

Definice 6. *Řekneme, že došlo ke kolizi, pokud existují dvě kuličky, které leží v jedné přihrádce.*

Zajímá nás pravděpodobnost, že během umísťování kuliček dojde ke kolizi. Tuto pravděpodobnost označíme V a budeme ji počítat pomocí doplňkového jevu. Označme P_j pravděpodobnost, že po umístění j kuliček nedošlo ke kolizi, $j = 1, ..., n$.

Určitě $P_1 = 1$. Dále $P_{j+1} = P_j \cdot (1 - \frac{j}{m})$, protože, aby nedošlo přidáním $j+1$ kuličky ke kolizi, nesmí k ní dojít během přidání předchozích j kuliček, a navíc musí být $j+1$ kulička umístěna do přihrádky, kde žádná z j předchozích neleží. Takto dostáváme vztah

$$P_n = \prod_{j=1}^{n-1} (1 - \tfrac{j}{m}) \tag{2.6}$$

Použijeme faktu, že $e^{-x} \approx (1 - x)$ pro x malá. Předpokládejme tedy, že $n \ll m$, pak máme $1 - \frac{j}{m} \approx e^{-\frac{j}{m}}$, který dosadíme do 2.6:

$$P_n \approx \prod_{j=1}^{n-1} e^{-\frac{j}{m}} = e^{-\sum_{j=1}^{n-1} \frac{j}{m}} = e^{-\frac{n(n-1)}{2m}} \tag{2.7}$$

Poslední rovnost dostáváme sečtením prvních $n-1$ členů aritmetické posloupnosti 1,2,3,... . Vztah 2.7 můžeme zjednodušit, pokud $n^2 \ll 2m$. Pak je totiž zlomek $\frac{n(n-1)}{2m}$ malý a můžeme opět využít aproximace pro e:

$$P_n \approx e^{-\frac{n(n-1)}{2m}} \approx 1 - \frac{n(n-1)}{2m} \tag{2.8}$$

Nás zajímá pravděpodobnost V. Pro ni dostáváme za předpokladu $n^2 \ll 2m$ (který v sobě zahrnuje i první předpoklad $n \ll m$) následující aproximaci:

$$V = 1 - P_n \approx \frac{n(n-1)}{2m} \approx \frac{n^2}{2m} =: \mathcal{B}(n, m) \tag{2.9}$$

Tento problém se nazývá narozeninový paradox, protože se často formuluje jako hledání pravděpodobnosti, že mezi n lidmi existují dva, kteří mají narozeniny ve stejný den (tedy $m = 365$).

2.5 Stromy

Poslední část kapitoly tvoří základní definice a terminologie týkající se grafů, speciálně pak stromů.

Definice 7. *Graf G je uspořádaná dvojice množin (V, E), kde $V \neq \emptyset$ a E je množina dvouprvkových podmnožin množiny V. Prvkům množiny V říkáme vrcholy, prvkům množiny E říkáme hrany.*

Orientovaný graf je takový, pro který je $E \subseteq V \times V$. Prvky E jsou pak orientované hrany.

Cesta v grafu G je posloupnost po dvou různých vrcholů $v_1, v_2, ..., v_k$ takových, že $\forall i = 1, ..., k-1 : \{v_i, v_{i+1}\} \in E$.

Tzv. smyčky, neboli hrany, které začínají a končí v tom samém vrcholu, v celé práci neuvažujeme.

Máme-li nějaký orientovaný graf G a jeho hranu $(v_1, v_2) = e \in V \times V$, můžeme říct, že hrana e vede z vrcholu v_1 do vrcholu v_2. Vrchol v_1 nazýváme rodičem v_2, vrchol v_2 nazýváme potomkem v_1. Teď zadefinujeme binární strom, se kterým se ve zbylé práci často setkáme.

Definice 8. *Strom je takový graf $G = (V, E)$, že mezi každými dvěma vrcholy existuje právě jedna cesta.*

Binární strom je orientovaný graf s jedním význačným vrcholem, kterému říkáme kořen, ve kterém platí, že kromě kořene má každý vrchol nejvýše dva potomky.

List stromu je vrchol, který nemá žádného potomka.

Hloubka stromu je počet vrcholů, které leží na nejdelší cestě z kořene do listu.

Hloubka vrcholu v je počet vrcholů, které leží na nejkratší cestě z v do listu, do tohoto počtu se list nezapočítává. Hloubka listu je 0.

Sourozenec vrcholu v v binárním stromě je vrchol, který má společného rodiče s v.

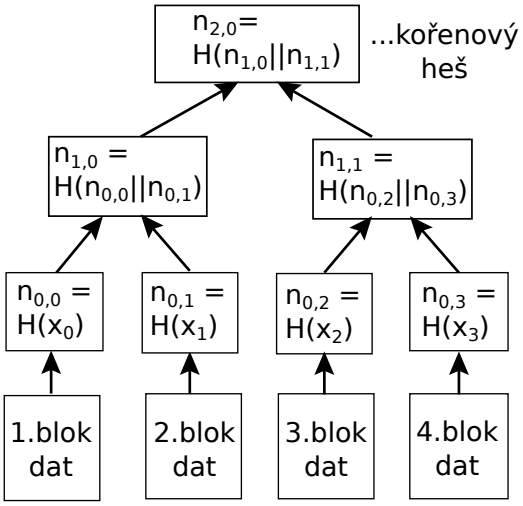
2.5.1 Hešovací stromy

Hešovací strom, neboli merkle strom (pojmenovaný podle tvůrce konceptu, Ralpha Merkleho), je datová struktura, která se používá ve chvíli, kdy je potřeba zajistit integritu dat a její ověření. Příkladem použití je přenos dat prostřednictvím peer-to-peer sítě v bitcoin protokolu.

Hešovací strom je strom, kde listy jsou tvořeny zhešovanými daty. My zde budeme pracovat se stromem binárním, který se používá právě k uchovávání bitcoin transakcí v hlavičce bloku (viz kapitoly 4, 5). Buď $H : \{0,1\}^m \to \{0,1\}^l$ pro nějaké $m, l \in \mathbb{N}$ kryptografická hešovací funkce. Označme $n_{i,j}$ vrchol stromu v i-té hloubce a $(j+1)$ v pořadí mezi všemi vrcholy dané hloubky číslováno zleva. Merkle strom má následující podobu:

1. Buď $x_0, x_1, x_2, ..., x_k$ bloky dat, či množiny souborů.
2. Vezměme $y_0, y_1, y_2, ..., y_k$, kde $y_i = H(x_i)$, $\forall i \in \{0, 1, 2, ..., k\}$.
3. Listy stromu definujeme jako $n_{0,j} = y_j \ \forall j \in \{0, 1, 2, ..., k\}$.
4. Ostatní vrcholy stromu pak tvoříme takto: rodič listů $n_{0,j}$, $n_{0,j+1}$ bude mít tvar $H(n_{0,j}||n_{0,j+1})$. Obecně pak rodič dvou vrcholů $n_{i,j}$, $n_{i,j+1}$ je tvaru $H(n_{i,j}||n_{i,j+1})$. Strukturu stromu ilustruje obrázek 2.2.

Význačným prvkem celého stromu je jeho kořen, který nazýváme *kořenový heš* (angl. root hash). V praxi jde o jedinou část stromu, kterou je potřeba získat od nějaké důvěryhodné autority. Zbytek stromu už může být stáhnut z libovolného zdroje. Integrita dat, z nichž je strom postaven, je pak prokázána tím, že má strom právě takový kořen, který nám poskytla důvěryhodná strana.



Obrázek 2.2: Hešovací strom

Vlastnosti

V hešovacím stromě platí, že každý rodič je nástroj k ověření správnosti dat ze svých potomků. Máme-li např. vrchol $n_{1,0}$, jenž je rodičem listů $n_{0,0}$ a $n_{0,1}$, pak pokud platí $H(n_{0,0}||n_{0,1}) = n_{1,0}$, jsou data v obou listech nezměněná a nepoškozená. Induktivně dojdeme k tomu, že pomocí jednoho vrcholu můžeme ověřit celistvost a nezměněnost celého podstromu tohoto vrcholu (z dat, jejichž správnost chceme ověřit, postavíme daný podstrom a pak porovnáme jeho kořen s vrcholem, který jsme měli na začátku).

Z výše uvedeného plyne také vlastnost, že integrita jednotlivých větví stromu může být ověřena i ve chvíli, kdy ještě není k dispozici strom celý (ovšem pořád je třeba mít kořenový heš z důvěryhodného zdroje). Pokud chceme například ověřit integritu části dat x_i pro nějaké $i \in \{0, 1, 2, ..., k\}$, pak potřebujeme kromě příslušného heše $y_i = H(x_i)$ znát posloupnost vrcholů $s_0, s_1, ..., s_{r-2}$ (r je hloubka stromu), kde s_0 je sourozenec y_i, s_1 je sourozenec vrcholu, který obsahuje $H(y_i||s_0)$ atd. Posloupnost $s_0, s_1, ..., s_{r-2}$ tedy představuje sourozence takových vrcholů, kterými projdeme cestou z y_i do kořene stromu. Ostatní vrcholy stromu nás v tu chvíli nemusí zajímat, stačí pouze pomocí $s_0, s_1, ..., s_{r-1}$ sestavit část našeho hešovacího stromu, zjistit tak kořenový heš a porovnat ho s tím, který jsme obdrželi od důvěryhodné autority. Pokud jsou kořenové heše stejné, prokázali jsme integritu části dat x_i.

3. Adresy

V této kapitole jsme čerpali ze stránek Address, Base58Check_encoding, Technical_background_of_Bitcoin_addresses, Testnet na [3].

(Bitcoin) adresa je číslo kódované v Base58 kódování (viz 3.4), jehož typická podoba je 25–34 velkých a malých písmen a číslic (kromě znaků `o`,`O`,`i`,`I`), která slouží k uchovávání a manipulaci s bitcoiny[1]. Pokud chceme někomu poslat určitou částku bitcoinů, pošleme ji na jednu z jeho adres. Naopak když sami vlastníme nějaké bitcoiny, tak to znamená, že máme přístup (tj. jak uvidíme později soukromý klíč) k nějaké adrese. Přičemž každý může mít libovolný počet adres, dokonce pro zajištění anonymity je doporučeno používat pro každou transakci novou adresu. Znaky `o`,`O`,`i`,`I` jsou z výběru vyřazeny a to proto, že v některých fontech vypadají stejně a mohly by tak být vytvořeny dvě vizuálně stejné adresy.

3.1 Adresa a klíče

Jak uvidíme dále, každá adresa představuje veřejný klíč z dvojice veřejného a soukromého klíče *ECDSA* (viz sekce 2.3 v Kapitole 2), na který je postupně aplikována řada úprav, díky nimž z adresy nelze odpovídající veřejný klíč zjistit. Pro každou adresu je vygenerován nový pár veřejného a soukromého klíče *ECDSA*. Tento pár klíčů je s adresou úzce svázán tak, že adresa i soukromý klíč jsou uloženy v peněžence vlastníka. Dále pokud chcete přistoupit k bitcoinům na určité adrese (máte v plánu je např. utratit), potřebujete soukromý klíč k této adrese. Jestliže svou peněženku ztratíte a nemáte příslušný soukromý klíč, pak se k bitcoinům na dané adrese nikdy nedostanete ani vám nebudou nijak nahrazeny.

3.2 Tvorba adresy

Bitcoin adresa je zcela zdarma a je možné ji vytvořit např. prostřednictvím bitcoin klienta, účtu ve směnárně či online peněženky (viz Kapitola 1). Není k tomu potřeba žádné registrace. Prostě si adresu vytvoříme a začneme ji používat (tj. objeví se v nějaké transakci viz Kapitola 4).

Mějme dvojici nově vygenerovaného soukromého K_{pri} a veřejného K_{pub} klíče *ECDSA*. Adresu A z K_{pub} vytvoříme následovně:

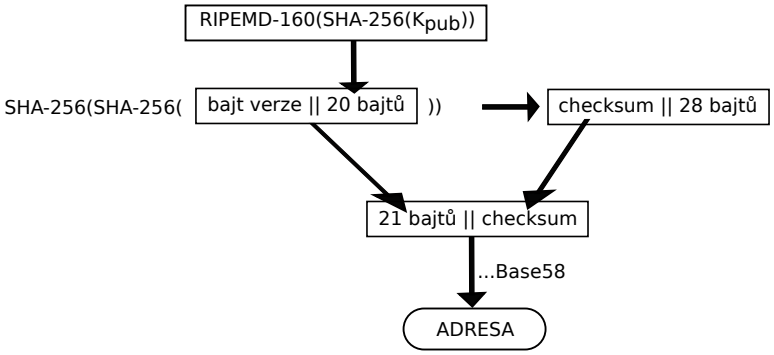
$$x = v_b || \text{RIPEMD-160}(\text{SHA-256}(K_{pub})) \tag{3.1}$$

$$A = \text{Base58Check}(x) \tag{3.2}$$

kde v_b je bajt označující verzi transakce. Dvojité zhešování veřejného klíče zajišťuje jednotnou velikost adres a také to, že ze samotné adresy nelze zjistit k ní příslušný veřejný klíč. Funkce Base58Check obstarává dvě věci. Jednak vkládá do adresy ochranný prvek tzv. *checksum*, který slouží k ověření toho, že je adresa úplná (tj. že nejde o delší adresu, ze které se některé znaky ztratily) a nezměněná (tedy že nedošlo např. k překlepu). Checksum jsou první čtyři bajty

[1] Ve skutečnosti existují dva typy adres. Jedny jsou běžné adresy s uvedenou délkou 25–34 znaků. Druhé jsou tzv. testnet adresy, které jsou používány v alternativním block chain. Takovéto adresy mají délku 26–34 znaků a začínají zpravidla písmenem m nebo n.

z SHA-256(SHA-256(x)), které se připojí na konec řetězce x. Druhou úpravu, kterou Base58Check provede, je zformátování řetězce x||checksum pomocí `Base58` kódování (viz 3.4). Celý proces ilustruje obrázek 3.1. Jednotlivé úpravy a další komentáře jsou uvedeny v následujícím příkladu.



Obrázek 3.1: Tvorba bitcoin adresy

Příklad tvorby adresy (formát a data příkladu jsou přejaty ze stránky Technical_background_of_Bitcoin_addresses na [3]):

1. Mějme soukromý klíč K_{pri}:

   $(18E14A7B6A307F426A94F8114701E7C8E774E7F9A47E2C2035DB29A206321725)_{16}$

2. Vezměme jemu odpovídající 65 bajtů dlouhý veřejný klíč K_{pub}:

   $(0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B23522CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6)_{16}$

3. Zhešujeme veřejný klíč hešovací funkcí SHA-256:

   $(600FFE422B4E00731A59557A5CCA46CC183944191006324A447BDB2D98D4B408)_{16}$

4. Na heš z předchozího kroku aplikujeme hešovací funkci RIPEMD-160:

   $(010966776006953D5567439E5E39F86A0D273BEE)_{16}$

5. Před heš z předchozího kroku přidáme bajt označující verzi (současná implementace má verzi $(00)_{16}$):

   $(00010966776006953D5567439E5E39F86A0D273BEE)_{16}$

   Následuje použití *Base58Check* kódování:

6. Řetězec v podobě z kroku 5 zhešujeme funkcí SHA-256:

   $(445C7A8007A93D8733188288BB320A8FE2DEBD2AE1B47F0F50BC10BAE845C094)_{16}$

7. Výsledný heš z předchozího kroku znovu zhešujeme funkcí SHA-256:
   $(\text{D61967F63C7DD183914A4AE452C9F6AD5D462CE3D277798075B107615C1A8A30})_{16}$
8. První čtyři bajty řetězce z kroku 7 představují kontrolní součet (checksum):
   $(\text{D61967F6})_{16}$
9. Vezměme výsledek z předchozího kroku a přidejme jej na konec řetězce z kroku 5. Tímto dostáváme 25 bajtový řetězec, označme ho A':
   $(00010966776006953D5567439E5E39F86A0D273BEED61967F6)_{16}$
10. Pro získání A zbývá už jen zakódovat A' pomocí *Base58* kódování. Při něm budeme na A' nahlížet jako na jedno velké celé číslo a budeme jej opakovaně dělit číslem 58. Zbytky po dělení budeme kódovat tak, jak je uvedeno v 3.4. Je ovšem třeba se zvlášť vypořádat s nulovými bajty (tedy bajty složenými ze samých nul) na začátku A'. Takové bajty se totiž v samotném *Base58* kódování neprojeví, jelikož jejich odstraněním nám zůstává to stejné velké číslo. Proto přidáme na začátek zakódovaného řetězce A' tolik znaků '1', kolik bylo v A' vedoucích nulových bajtů (protože nula odpovídá v *Base58* znaku '1'). [2]
11. Jako výsledek *Base58* kódování řetězce A' dostáváme bitcoin adresu A délky 33 znaků:
    16UwLL9Risc3QfPqBUvKofHmBQ7wMtjvM

Všechny úpravy počínaje krokem 5 mají za úkol (když pominu význam kontrolní sumy) převést dvojnásobně zhešovaný veřejný klíč do jednotného formátu (např. všechny adresy jedné verze budou začínat stejným znakem). V takovéto zformátované podobě se s adresou pracuje veřejně, tj. ve chvíli, kdy se přesouvají bitcoiny z jedné adresy na druhou (např. na stránkách [2] můžeme v jednotlivých transakcích vidět přesuny bitcoinů mezi adresami). Interně však ve skriptech transakcí (viz kapitola 4) pracujeme pouze s dvojnásobně zhešovaným veřejným klíčem. Proto si dovolíme v následujících kapitolách nazývat adresou nejen A z kroku 11, ale i dvakrát zhešovaný veřejný klíč jako například ve výsledku kroku 4.

3.3 Kolize adres

Tato sekce předkládá náš vlastní důkaz toho, že pravděpodobnost kolize dvou adres je velmi malá.

Kolize, tedy případ, kdy si dva klienti vygenerují stejnou adresu, vede k tomu, že oba mohou manipulovat s penězi, jež na této adrese jsou. Jak může ke kolizi dojít? Podíváme-li se na algoritmus tvorby adresy, dojdeme k závěru, že jsou následující možnosti:

1. Oba uživatelé budou mít stejnou dvojici soukromého a veřejného klíče k_{pri} a k_{pub}.

[2] Tento postup zaručí jednoduchou identifikaci adres, protože všechny adresy s bajtem označujícím verzi rovným $(00)_H$ tak začínají znakem '1'.

2. Dojde ke kolizi jedné z hešovacích funkcí. Tedy buď nastane kolize hešovací funkce SHA-256, která dva navzájem různé veřejné klíče k_{pub1},k_{pub2} zhešuje na stejný řetězec.

   Nebo dojde ke kolizi v hešovací funkci RIPEMD-160, která dva různé heše SHA-256(k_{pub1}), SHA-256(k_{pub2}), kde $k_{pub1} \neq k_{pub2}$, zhešuje na stejné řetězce, které opět vedou na tutéž výslednou bitcoin adresu.

Uvažujme n existujících adres a spočtěme pravděpodobnost, že dvě z těchto adres jsou stejné. Mějme řetězce A, A'-dva veřejné klíče *ECDSA*, a dále řetězce B, B', C, C' tak, že:

$$A \longrightarrow B = \text{SHA-256}(A) \longrightarrow C = \text{RIPEMD-160}(B)$$

$$A' \longrightarrow B' = \text{SHA-256}(A') \longrightarrow C' = \text{RIPEMD-160}(B')$$

kde $a := |A| = |A'| = 65 \cdot 8 = 520$ bitů, $b := |B| = |B'| = 256$ bitů, $c := |C| = |C'| = 160$ bitů.

Označme jevy $A_1 = \{A = A'\}$, $B_1 = \{B = B'\}$, $C_1 = \{C = C'\}$ a doplňkové jevy $A_2 = \{A \neq A'\}$, $B_2 = \{B \neq B'\}$, $C_2 = \{C \neq C'\}$. Dále použijeme vzorec $\mathcal{B}(n, m) = \frac{n^2}{2m}$ ze sekce 2.4 Kapitoly 2, který vyjadřuje aproximaci pravděpodobnosti, že mezi n řetězci bitů fixní délky existují dva stejné, kde m je počet všech řetězců dané délky, za předpokladu, že $n^2 \ll 2m$. Pro nás n představuje počet existujících adres a nejmenší m, které budeme do vztahu $\mathcal{B}$ dosazovat, je $m = 2^{160}$, tedy $2 \cdot m = 2 \cdot 2^{160} \doteq 2,92 \cdot 10^{48}$. Odhadem je na světě 6 miliard lidí. Kdyby si každý denně vytvořil 1000 adres po dobu 100 let, pak by $n^2 \doteq 4,67 \cdot 10^{34}$. Vidíme tak, že předpoklad pro použití $\mathcal{B}$ je splněn pro $m = 160$, tím spíše pak pro $m = 256$ a $m = 520$. Platí, že

$$\begin{aligned} P(A_1) &\approx \mathcal{B}(n, 2^a) \\ P(A_1 \cup A_2) &= 1 \\ P(A_1) \neq 0 &\neq P(A_2) \end{aligned}$$

analogicky pro jevy B_1, B_2.

Tímto jsou ověřeny předpoklady věty o úplné pravděpodobnosti (viz [11]), ze které dostáváme rovnost:

$$P(B_1) = P(B_1|A_1) \cdot P(A_1) + P(B_1|A_2) \cdot P(A_2)$$

kde

$$\begin{aligned} P(B_1|A_1) &= 1 \\ P(B_1|A_2) &\approx \mathcal{B}(n, 2^b) \\ P(A_2) &= 1 - P(A_1) \end{aligned}$$

Tedy

$$P(B_1) \approx \mathcal{B}(n, 2^a) + \mathcal{B}(n, 2^b) \cdot (1 - \mathcal{B}(n, 2^a))$$

Analogicky z věty o úplné pravděpodobnosti počítáme

$$P(C_1) = P(C_1|B_1) \cdot P(B_1) + P(C_1|B_2) \cdot P(B_2)$$

kde

$$\begin{aligned} P(C_1|B_1) &= 1 \\ P(C_1|B_2) &\approx \mathcal{B}(n, 2^c) \\ P(B_2) &= 1 - P(B_1) \end{aligned}$$

Dostáváme tak

$$\begin{aligned} P(C_1) \approx\ & \mathcal{B}(n, 2^a) + \mathcal{B}(n, 2^b) \cdot (1 - \mathcal{B}(n, 2^a)) \\ & + \mathcal{B}(n, 2^c) \cdot (1 - \mathcal{B}(n, 2^a) - \mathcal{B}(n, 2^b) \cdot (1 - \mathcal{B}(n, 2^a))) \end{aligned} \tag{3.3}$$

$$\begin{aligned} P(C_1) \approx\ & \mathcal{B}(n, 2^a) + \mathcal{B}(n, 2^b) + \mathcal{B}(n, 2^c) \\ & - \mathcal{B}(n, 2^b) \cdot \mathcal{B}(n, 2^a) - \mathcal{B}(n, 2^c) \cdot \mathcal{B}(n, 2^a) - \mathcal{B}(n, 2^c) \cdot \mathcal{B}(n, 2^b) \\ & + \mathcal{B}(n, 2^a) \cdot \mathcal{B}(n, 2^b) \cdot \mathcal{B}(n, 2^c) \end{aligned} \tag{3.4}$$

Spočteme nejprve

$$\mathcal{B}(n, 2^a) = \frac{n^2}{2^{521}} \tag{3.5}$$

$$\mathcal{B}(n, 2^b) = \frac{n^2}{2^{257}} \tag{3.6}$$

$$\mathcal{B}(n, 2^c) = \frac{n^2}{2^{161}} \tag{3.7}$$

$$\mathcal{B}(n, 2^a) + \mathcal{B}(n, 2^b) + \mathcal{B}(n, 2^c) = \frac{n^2 \cdot (1 + 2^{264} + 2^{161})}{2^{521}} \tag{3.8}$$

Jak vidíme z (3.5), (3.6), (3.7), jsou pravděpodobnosti $\mathcal{B}(n, 2^a), \mathcal{B}(n, 2^b), \mathcal{B}(n, 2^c)$ velmi malá čísla. Proto budou součiny těchto výrazů v (3.4) ještě menší čísla a můžeme je tedy ve výpočtech zanedbat. Pravděpodobnost, že v n existujících adresách najdeme dvě stejné tak můžeme aproximovat

$$P(C_1) \approx \frac{n^2 \cdot (1 + 2^{264} + 2^{161})}{2^{521}}$$

3.4 BaseX kódování

BaseX kódování se používá v případě, kdy potřebujeme binární řetězec dlouhý n bajtů reprezentovat jako posloupnost tisknutelných znaků z dolní poloviny ASCII tabulky [7]. Symbol X v názvu značí počet znaků, do kterých je vstupní binární řetězec kódován. Při výběru těchto znaků se jednotlivé implementace liší, avšak vždy je snahou vybrat takovou množinu tisknutelných znaků, která bude součástí co největšího počtu kódování. Příkladem může být množina znaků velikosti 64, kde prvních 52 znaků jsou malá a velká písmena anglické abecedy, dalších deset znaků jsou číslice 0–9 a poslední dva znaky jsou `'+'` a `'/'`.

3.4.1 Postup kódování

Mějme binární řetězec B dlouhý n bajtů, který chceme zakódovat pomocí *BaseX* kódování. Kódování probíhá tak, že budeme na řetězec B jakožto na dlouhé celé číslo opakovaně aplikovat dělení se zbytkem číslem X, zbytek po dělení pak pomocí *BaseX* tabulky převedeme na příslušný znak.

3.4.2 Base58 kódování

Base58 kódování je *BaseX* kódování pro $X = 58$. Množina používaných znaků v *Base58* sestává z 10 číslic 0–9 a následuje 48 velkých a malých písmen anglické abecedy (vyřazena jsou písmena `o,O,i,I`). Tabulka *Base58* má tedy podobu:
`"123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz"`
a interpretujeme ji tak, že i-tý prvek tabulky odpovídá zbytku i po dělení 58 pro $i = 0, ..., 57$ (tj. nultý znak tabulky odpovídá zbytku nula po dělení 58 atd.).

4. Transakce

Zdrojem k této kapitole jsou stránky Transaction, OP_CHECKSIG, Contracts, Script, Protocol_specification na [3], dále [1] a [2].

Transakce je podepsaný úsek dat, kterým jsou převáděny bitcoiny z jedné adresy na druhou. Takováto data jsou přenášena po bitcoin síti a shromažďována v blocích (viz Kapitola 5). Transakce vždy obsahuje určité množství vstupů a výstupů.

Veškerá manipulace s bitcoiny je realizována prostřednictvím transakcí. Proto jsou bitcoiny a transakce úzce spjaty. O tom svědčí i fakt, že se nové bitcoiny vytváří („těží“) pomocí tzv. *coinbase* transakce. Vlastnit určitou částku bitcoinů pak znamená moci nakládat s výstupem nějaké transakce, který má příslušnou hodnotu. A nakládáním s výstupem je opět míněno vytvoření nějaké nové transakce, která se bude na onen výstup odkazovat.

Vstup transakce obsahuje, odkud, tedy z jaké již existující transakce, chceme bitcoiny vzít, a data o tom, že máme právo s těmito bitcoiny manipulovat. To znamená, že vstup nové transakce se odkazuje na výstup nějaké již proběhlé transakce (říkejme jí předchozí), přičemž na každý výstup transakce se může odkazovat nejvýše jeden vstup. Vstup tak obsahuje heš předchozí transakce, index výstupu předchozí transakce, na který se tím odkazuje, velikost skriptu, samotný skript a číslo sekvence. Výstup transakce je pak složen z čísla udávajícího hodnotu v bitcoinech, jakou představuje, následuje velikost skriptu a skript samotný. Všechny pojmy jsou vysvětleny v sekcích 4.1 a 4.3. Prostřednictvím skriptů je zajištěno, aby s bitcoiny z výstupu jedné transakce mohl nakládat jen ten, komu jsou určeny. Proto skript výstupu předchozí transakce obsahuje požadavky, které jsou splnitelné pouze pro toho, komu mají bitcoiny tohoto výstupu patřit. Každý, kdo se na tyto bitcoiny odkazuje vstupem své transakce (tj. chce s nimi manipulovat), se pak prostřednictvím skriptu v tomto vstupu prokazuje a snaží se splnit požadavky výstupu předchozí transakce. Pokud je splní, jsou bitcoiny jeho. Pokud ne, jeho nová transakce není platná. Jak uvidíme dále, tento mechanismus ověření adresáta je složitější.

Pokud chceme uskutečnit novou transakci, tak její vstupy budou obsahovat odkazy na výstupy takových transakcí, jejichž bitcoiny chceme přesunout. Nově vzniklá transakce tedy bude představovat takovou hodnotu v bitcoinech, kterou je součet hodnot výstupů, na které se nová transakce odkazuje svými vstupy. Novou transakcí pak mohu tyto bitcoiny poslat všechny na jednu adresu, nebo můžu jednotlivé částky poslat zvlášť, v tom případě pro každou částku vytvořím zvláštní výstup.

Definice 9. *Hodnota vstupu nějaké transakce je hodnota výstupu, na který se vstup odkazuje.*

Samozřejmě pro každou transakci musí platit, že součet hodnot všech jejích vstupů je větší, nebo roven součtu hodnot všech jejích výstupů. Pokud nastane nerovnost, pak jsou bitcoiny, které nejsou součástí žádného výstupu transakce, považovány za *transakční poplatek* a připadnou tomu, kdo připojí blok, v němž je ona transakce, k block chainu (viz kapitola 5). Takovéto neutracené bitcoiny majitel nikdy nedostane zpátky, nejsou totiž součástí výstupu, na který by se dalo odkázat.

Síla tohoto návrhu spočívá mimo jiné v tom, že se na jedné transakci prostřednictvím jejích jednotlivých vstupů může podílet více lidí. Lze tak kupříkladu uskutečnit nákup ve více lidech.

Příklad: Teta mi chce poslat x BTC, za které si plánuji koupit knihu v hodnotě y BTC a zbytek si nechat. Teta tedy uskuteční transakci Tx_1, která převede x BTC na jednu z mých adres. Já s těmito x BTC můžu nakládat opět prostřednictvím transakce. Tedy vytvořím transakci Tx_2, která se svým jediným vstupem bude odkazovat na výstup Tx_1 v hodnotě x BTC. Prvním výstupem mojí Tx_2 bude y BTC na adresu knihkupectví a druhým výstupem bude $(x - y)$ BTC na jednu z mých adres. Kdybych $(x - y)$ BTC neposlala sama sobě, staly by se transakčním poplatkem, a připadly by úspěšnému těžaři bloku obsahujícího Tx_2.

4.1 Reprezentace transakce

Transakce je řetězec bajtů v podobě, kterou udává tabulka 4.1 (příklad takovéto běžné transakce najdete v Příloze A):

DATA	POPIS
version	Číslo označující verzi software, který transakci vytvořil
vin_sz	Počet vstupů transakce
in	Seznam vstupů
vout_sz	Počet výstupů transakce
out	Seznam výstupů
lock_time	Časový zámek

Tabulka 4.1: Struktura transakce (*Převzato z Transaction na [3]*).

Časový zámek nám říká, do jaké doby je transakce uzamčena. Zámkem tak určujeme období, po jaké má transakce čekat na vyřízení a může být nahrazena nějakou svou jinou verzí. Význam jiných verzí transakce se ozřejmí v sekci 4.4. Když tato doba uplyne, transakce je považována za finální.

4.1.1 Výstup transakce

Výstup určuje, která adresa má být příjemce dané částky bitcoinů a jakým způsobem se daná adresa musí prokázat, aby jí byly bitcoiny zpřístupněny. Struktura výstupu je uvedena v tabulce 4.2.

DATA	POPIS
value	Hodnota výstupu
script length	Velikost následujícího skriptu
`scriptPubKey`	Skript

Tabulka 4.2: Struktura výstupu transakce *(Převzato z Transaction na [3]).*

Hodnota výstupu je udávána v jednotce Satoshi, kde 1 BTC=10^8 Satoshi. Hodnota je uložena v osmi bajtech, které tak určují, na kolik částí lze jeden bitcoin rozdělit.

4.1.2 Vstup transakce

Vstup transakce představuje odkaz na výstup předchozí transakce a data, jimiž k bitcoinům předchozí transakce získáme přístup. Struktura vstupu je uvedena v tabulce 4.3.

DATA	POPIS
hash	Heš předchozí transakce
index	Index výstupu předchozí transakce
script length	Velikost následujícího skriptu
`scriptSig`	Skript
nSequence	Číslo sekvence

Tabulka 4.3: Struktura vstupu transakce *(Převzato z Transaction na [3]).*

Heš předchozí transakce je SHA-256(SHA-256(předchozí transakce)). Index výstupu předchozí transakce je číslo takového výstupu, na který se daný vstup odkazuje. Číslo sekvence udává, o kolikátou verzi dané transakce jde. V současné chvíli se však nSequence nepoužívá, přesněji řečeno všechny transakce ji musí mít nastaveny na maximální možnou hodnotu, která reprezentuje to, že nelze vytvořit další verzi dané transakce. Důsledky případného budoucího používání nSequence uvidíme v sekci 4.4. `scriptSig` je sada dat a příkazů pracujících se zásobníkem. Podrobnosti uvidíme v 4.3.

Definice 10. *Řekneme, že vstup transakce Tx je platný, jestliže po vyhodnocení* `scriptPubKey` *a* `scriptSig` *zůstane na zásobníku* `true`*. Přičemž* `scriptSig` *je skript ze vstupu Tx a* `scriptPubKey` *je skript výstupu transakce, na který se vstup Tx odkazuje.*

Význam definice bude zřejmý po přečtení sekce 4.3.

Definice 11. *Řekneme, že transakce Tx je platná, pokud je každý její vstup platný. Transakce Tx je neplatná, pokud existuje nějaký její vstup, který není platný.*

4.2 Coinbase transakce

Zvláštním druhem transakce jsou tzv. coinbase transakce. Jde o takové transakce, které slouží k emisi nových bitcoinů a k odměňování těžařů, kteří tak získávají nově vytěžené bitcoiny a transakční poplatky obsažené v daném bloku (viz Kapitola 5). V coinbase transakci vždy musí platit, že součet hodnot vstupu = odměna za blok plus všechny transakční poplatky v daném bloku. Coinbase transakce má v podstatě stejnou podobu, jako standardní transakce a platí:

- Má vždy právě jeden vstup.
- Heš předchozí transakce tohoto vstupu je vždy nastaven na samé nuly, index výstupu předchozí transakce je nastaven na číslo $(\text{FFFFFFFF})_{16}$. Tyto hodnoty reprezentují to, že coinbase transakce je první v bloku a neodkazuje se na žádnou jinou transakci.
- Obsah skriptu `scriptSig` je jiný – viz 4.3.2.

4.3 Skript

Bitcoin používá vlastní skriptovací systém pro tvorbu a ověřování platnosti transakcí. Skriptovací systém je sada příkazů (ty budeme nadále značit velkými písmeny) pracujících se zásobníkem. Jak už jsme uvedli výše, transakce se krom jiného skládá ze vstupů a výstupů. Vstup nám říká, z jaké jiné transakce bitcoiny beru, výstup říká, kam (na jakou adresu) je přemisťuji.

Má-li někdo ve vlastnictví nějaké bitcoiny, znamená to, že má přístup k výstupu nějaké transakce Tx_1 (přesněji má soukromý klíč k adrese, která je ve výstupu uvedena). To, že má takový přístup, musí nějak prokázat, obvykle se dotyčný prokazuje *ECDSA* podpisem (viz sekce 2.3 v Kapitole 2), čímž demonstruje, že má požadovaný soukromý klíč, a to ve chvíli, kdy vytváří vlastní transakci Tx_2, ve které chce své bitcoiny přemístit, utratit.

Zpravidla celý proces proběhne tak, že tvůrce Tx_1 umístí do výstupu Tx_1 skript obsahující příkaz na kontrolu podpisu a tvůrce Tx_2 umístí do vstupu Tx_2 skript obsahující podpis a veřejný klíč. Při ověřování platnosti Tx_2 se oba skripty vyhodnotí a transakce se prohlásí za platnou, pokud je podpis úspěšně ověřen. Má-li Tx_2 více vstupů, pak je platná, pokud pro každý její vstup proběhne úspěšné vyhodnocení příslušných skriptů. Tento mechanismus zajišťuje , aby s bitcoiny na výstupu Tx_1 mohl manipulovat jen ten, komu jsou určeny.

Skript je tedy posloupnost dat (ty budeme nadále značit v šípových závorkách `'<>'`) a příkazů, které říkají, co se má dít se zásobníkem a s daty, které jsou součástí skriptu.

V transakcích se obvykle setkáme se dvěma typy skriptů – tzv. `scriptPubKey` a `scriptSig`. První z nich, `scriptPubKey`, je posloupnost příkazů a dat, které jsou součástí výstupu transakce a určují, jakým způsobem lze k bitcoinům tohoto výstupu získat přístup. Jinak řečeno `scriptPubKey` předkládá, co se požaduje od vstupu, který se na něj odkáže. Pomocí skriptovacího systému si každý tvůrce transakce zvolí sám, co musí splnit ten, kdo chce bitcoiny utratit. Lze například vytvořit transakci, u níž může kdokoliv, nebo naopak nikdo, utratit převáděné bitcoiny. `scriptSig` je posloupnost dat a příkazů, která je součástí vstupu transakce a slouží k prokázání toho, že jsou splněny požadavky uvedené ve výstupu, na který se vstup se skriptem odkazuje.

Když chceme ověřit, zda daný vstup jedné transakce je oprávněn nakládat s bitcoiny na výstupu nějaké jiné transakce, tak vezmeme `scriptSig` vstupu a `scriptPubKey` příslušného výstupu a oba skripty v tomto pořadí vyhodnotíme. Pokud na vrcholu zásobníku po vyhodnocení zůstane `true`, pak vstup skutečně může s danými bitcoiny manipulovat (a vstup je tedy platný).

4.3.1 Technické detaily

Zásobník, který je obsluhován operačními příkazy skriptovacího systému, pracuje s bajtovými vektory. Ty jsou interpretovány v little endian a reprezentovány jako `integer` proměnné délky. Bajtové vektory můžeme vyhodnocovat i jako proměnnou typu `boolean`, v takovém případě jsou všechny vektory, které reprezentují nulu, považovány za hodnotu `false` a všechny ostatní vektory za hodnotu `true`.

Vybrané příkazy

Příkazů skriptovacího jazyka je velké množství, jejich úplný přehled můžete nalézt na stránce Script na [3]. My se zde podíváme na několik nejpodstatnějších, které uvedeme v tabulce 4.4. Vstupem příkazu jsou vždy hodnoty z vrcholu zásobníku, které jsou tak ze zásobníku sejmuty. Naopak výstup daného příkazu je vždy vložen na vrchol zásobníku.

PŘÍKAZ	HEX	VSTUP	VÝSTUP
	POPIS		
`OP_VERIFY`	$(69)_{16}$	`true` / `false`	nic / `false`
	Pokud na vrcholu zásobníku není `true`, pak označí celou transakci za neplatnou. `true` je z vrcholu zásobníku odstraněna, `false` nikoliv.		
`OP_DUP`	$(76)_{16}$	x_1	x_1,x_1
	Položku z vrcholu zásobníku zkopíruje a zapíše na zásobník.		
`OP_EQUAL`	$(87)_{16}$	x_1, x_2	`true` / `false`
	Pokud jsou vstupy stejné, tak vrátí `true`, jinak vrátí `false`.		
`OP_EQUALVERIFY`	$(88)_{16}$	x_1, x_2	`true` / `false`
	Nejprve proběhne `OP_EQUAL` a následně `OP_VERIFY`.		
`OP_ADD`	$(93)_{16}$	x_1, x_2	$x_1 + x_2$
	Sečte vstupní hodnoty.		
`OP_HASH160`	$(A9)_{16}$	x_1	RIPEMD-160(SHA-256(x_1))
	Vstup je nejprve zhešován funkcí SHA-256, následně je aplikována funkce RIPEMD-160.		
`OP_CODESEPARATOR`	$(AB)_{16}$	žádný	žádný
	Slouží k oddělení dat, která se mají zhešovat.		
`OP_CHECKSIG`	$(AC)_{16}$	`<sig>`,`<pubkey>`	`true` / `false`
	Slouží k ověření podpisu `<sig>` pomocí `<pubkey>`, viz 4.3.3.		

Tabulka 4.4: Příkazy skriptovacího systému *(Převzato z Script na [3]).*

Pokud se ve skriptu objeví samotná data, pak to znamená, že mají být vložena na vrchol zásobníku.

4.3.2 Příklady skriptů

Zde uvedeme několik příkladů skriptů. První z nich slouží k ilustraci skriptovacího systému, další dva jsou součástí bitcoin protokolu. V příkladech je vždy uvedeno, jaké změny se provedly, jak v daném momentě vypadá zásobník, kdy vrchol zásobníku je vpravo, a jak vypadá dosud nevykonaná část skriptu.

Ilustrační příklad

Následující příkaz slouží k ilustraci toho, jak funguje skriptovací systém. Mějme tři čísla `<a>`, `<b>` a `<c>`. Budeme chtít zjistit, zda `<a>` + `<b>` = `<c>`. Náš skript vypadá takto: `script = <c> <a> <b> OP_ADD OP_EQUAL`. Vyhodnocení skriptu :

1. Ze skriptu zatím nebyl proveden žádný příkaz.
   Zásobník: prázdný
   Skript: `<c> <a> <b> OP_ADD OP_EQUAL`

2. Číslo `<c>` bylo vloženo na zásobník.
   Zásobník: `<c>`
   Skript: `<a> <b> OP_ADD OP_EQUAL`

3. Číslo `<a>` bylo vloženo na zásobník.
   Zásobník: `<c> <a>`
   Skript: `<b> OP_ADD OP_EQUAL`

4. Číslo `<b>` bylo vloženo na zásobník.
   Zásobník: `<c> <a> <b>`
   Skript: `OP_ADD OP_EQUAL`

5. Čísla `<a> <b>` z vrcholu zásobníku byla sečtena.
   Zásobník: `<c> <a+b>`
   Skript: `OP_EQUAL`

6. Čísla z vrcholu zásobníku jsou sejmuta a porovnána. Výsledek porovnání je vložen na zásobník.
   Zásobník: `true` / `false`
   Skript: prázdný

Standardní transakce

Tento příklad je příklad skriptu a jeho vyhodnocení použitého při standardní transakci. Skripty mají následující podobu:
`scriptPubKey = OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`
`scriptSig = <sig> <pubkey>`

Přičemž `scriptSig` je skript obsažený ve vstupu aktuální transakce (označme ji Tx_2) a `scriptPubKey` je skript obsažený ve výstupu nějaké předchozí transakce (označme ji Tx_1), na kterou se vstup Tx_2 obsahující `scriptSig` odkazuje. Dále `<pubKeyHash>` představuje adresu toho, komu jsou bitcoiny posílány, `<sig>` a `<pubkey>` je po řadě podpis a veřejný klíč tvůrce Tx_2, tedy toho, kdo usiluje o přístup k bitcoinům z výstupu Tx_1. `<sig>` je *ECDSA* podpis SHA-256(SHA-256(Tx_2)) pomocí soukromého klíče příslušného veřejnému klíči `<pubkey>`.

Při vyhodnocení skriptů, a tedy ověření, zda autorovi Tx_2 jsou určeny bitcoiny na výstupu Tx_1, jsou vyhodnocovány skripty `scriptSig` a `scriptPubKey` za sebou jako jeden skript takto (v příkladu proběhne ověření kladně):

1. Ze skriptu zatím nebyl proveden žádný příkaz.
   Zásobník: prázdný
   Skript: `<sig> <pubkey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`

2. Data `<sig> <pubkey>` byla vložena na zásobník.
   Zásobník: `<sig> <pubkey>`
   Skript: `OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`

3. Vektor na vrcholu zásobníku byl zkopírován.
   Zásobník: `<sig> <pubkey> <pubkey>`
   Skript : `OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`

4. Vektor na vrcholu zásobníku byl zhešován funkcí RIPEMD-160 $\circ$ SHA-256.
   Zásobník: `<sig> <pubkey> <pubHashA>`
   Skript : `<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`

5. Data `<pubKeyHash>` byla vložena na vrchol zásobníku.
   Zásobník: `<sig> <pubkey> <pubHashA> <pubKeyHash>`
   Skript : `OP_EQUALVERIFY OP_CHECKSIG`

6. Dva vektory z vrcholu zásobníku byly použity jako vstupní hodnoty pro funkci `OP_EQUALVERIFY`. Ta nejprve provede příkaz `OP_EQUAL`, který vrátí na zásobník `true`, pokud jsou vstupy stejné (jako v našem příkladě), jinak vrátí na zásobník `false`. Následně je proveden příkaz `OP_VERIFY`, který prohlásí transakci za neplatnou, pokud je na vrcholu zásobníku `false`. Pokud je na vrcholu zásobníku vektor interpretován jako `true`, pak je tento vektor ze zásobníku sejmut. V této části kódu tedy zjišťujeme, zda RIPEMD-160(SHA-256(`<pubkey>`)) ze vstupu Tx_2 je roven `<pubKeyHash>` z výstupu Tx_1. Dochází tak ke kontrole, zda bitcoiny, které jsou autorem Tx_1 poslány na adresu `<pubKeyHash>`, skutečně budou zpřístupněny někomu s touto adresou. Postup zde samozřejmě nemůže skončit vzhledem k tomu, že adresa je zjednodušeně řečeno zhešovaný veřejný klíč, který by kdokoliv mohl prohlašovat za svůj. Pokud se však k bitcoinům snaží přistoupit někdo z jiné adresy, než na kterou jsou poslány, pak nemá smysl pokračovat a transakce Tx_2 je rovnou prohlášena za neplatnou.
   Zásobník: `<sig> <pubkey>`
   Skript : `OP_CHECKSIG`

7. Příkaz `OP_CHECKSIG` provede ověření podpisu `<sig>` pomocí veřejného klíče `<pubkey>`. V našem případě byl podpis úspěšně ověřen. Podrobnosti viz sekce 4.3.3.
   Zásobník: `true`
   Skript: prázdný

Coinbase transakce

Coinbase transakce (viz sekce 4.2) je výjimečná tím, že se neodkazuje na žádnou předchozí transakci. Proto `scriptSig` jejího vstupu slouží většinou jen k přenosu dat, která např. uvádějí, podle jakých `BIP` (viz Kapitola 1) byla daná transakce vytvořena. V současné době `scriptSig` obsahuje hloubku bloku (viz Kapitola 5) s coinbase transakcí ve větvi bitcoin stromu (podle `BIP 0034`), extranonce (viz sekce 5.2.2 v Kapitole 5) a nakonec „/P2SH/“ (podle `BIP 0016`).

4.3.3 OP_CHECKSIG

`OP_CHECKSIG` je příkaz, který slouží k ověření podpisu ze vstupu nějaké transakce. Zpravidla se používá jako součást skriptu `scriptPubKey`. Podpis, který je ověřován, je vždy *ECDSA* podpis (viz sekce 2.3 v Kapitole 2) dat tvaru SHA-256(SHA-256(Tx)), kde Tx je transakce, v jejímž vstupu podpis leží (je tedy logické, že když se Tx podepisuje, tak se podepisuje bez skriptů vstupů, do nichž budou podpisy uloženy).

Podpis transakce může mít několik variant (tzv. forem) určených hodnotou zvanou `nHashType`. Forma transakce slouží mimo jiné ke tvorbě některých typů kontraktů (viz sekce 4.4). Příkladem takového kontraktu je transakce, ve které jednotlivé vstupy pochází od různých lidí. Konkrétně si lze představit veřejnou sbírku na koupi nějaké věci, která se uskuteční jen pokud se vybere dostatečné množství peněz. Formy transakcí však mají význam jedině tehdy, jze-li používat a měnit hodnotu nSequence (viz sekce 4.1), což v tuto chvíli není možné.

`nHashType` může nabývat tří hodnot:

- `SIGHASH_ALL`: Jde o defaultní hodnotu. Značí, že podepisujeme celou transakci (samozřejmě kromě skriptů vstupů), nemůže tedy dojít k její změně bez našeho vědomí (pokud se transakce změní, náš podpis nebude platný, a tudíž nebude platná ani celá transakce).

- `SIGHASH_NONE`: V tomto případě nepodepisujeme výstupy transakce a dáváme tak najevo, že je nám jedno, jaké jsou. Zároveň nastavíme nSequence všech vstupů kromě našeho na nulu. Číslo nSequence je součástí každého vstupu a označuje, zda lze vytvořit novou verzi daného vstupu. Tím, že nastavíme nSequence ostatních vstupů na nulu, říkáme, že mohou být vytvořeny nové verze dané transakce, aniž by se zneplatnil podpis našeho vstupu.

- `SIGHASH_SINGLE`: Opět jsou nSequence všech vstupů kromě našeho nastaveny na nulu. Podepíší se všechny vstupy transakce a jeden jediný výstup (ten, co je na stejné pozici jako náš vstup). Znamená to, že bez našeho souhlasu nikdo nesmí změnit vstupy transakce a jeden náš výstup.

`nHashType` můžeme kombinovat s modifikátorem `SIGHASH_ANYONECANPAY`, kterým říkáme, že chceme podepsat pouze náš vstup a ostatní vstupy nás nezajímají.

`OP_CHECKSIG` jako vstupní parametry vyžaduje dvě hodnoty na vrcholu zásobníku, a to veřejný klíč `<pubkey>` a hned pod ním podpis `<sig>`. Zároveň potřebuje mít k dispozici aktuální transakci a index vstupu, jehož skript je vyhodnocován. Představme si, že máme transakci Tx a vyhodnocujeme její vstup s indexem `nIn`. Proto vezmeme skript `scriptPubKey` z výstupu takové transakce, na kterou se Tx svým vstupem s indexem `nIn` odkazuje, a dále `scriptSig` z našeho vstupu. Oba skripty v tomto pořadí spojíme do skriptu `script`, jehož součástí bude příkaz `OP_CHECKSIG` a data `<pubkey>` a `<sig>`. Vyhodnotíme `script` až k příkazu `OP_CHECKSIG`. Vyhodnocením části skriptu byl na zásobník umístěn `<sig>` a poté `<pubkey>`. Příkaz `OP_CHECKSIG` má za úkol uvést Tx do správné podoby Tx', zhešovat ji a následně vůči tomuto heši ověřit podpis `<sig>` pomocí klíče `<pubkey>`. Přesněji ověří, že `<sig>` je podpis dat ve tvaru Tx' pomocí soukromého klíče, který přísluší veřejnému klíči `<pubkey>`. Konkrétně provede toto:

1. Sejme ze zásobníku `<pubkey>` a `<sig>`.
2. Ze `script` vytvoří subskript `scriptCode`, který začíná posledním vyhodnoceným `OP_CODESEPARATOR`. Pokud žádný takový není, je subskript roven celému `script`.
3. Pokud je v subskriptu přítomen `<sig>`, pak je z něj vymazán (data, co byla podepsána, nemohla obsahovat podpis).
4. Do proměnné `nHashType` přiřadí poslední bajt podpisu, tento bajt je z podpisu smazán.
5. Z `scriptCode` jsou smazány všechny příkazy `OP_CODESEPARATOR`.
6. Vytvoří kopii transakce Tx do proměnné `txTmp`, v níž jsou skripty `scriptSig` u všech vstupů nastaveny na prázdné, kromě vstupu s indexem `nIn`. U tohoto vstupu totiž `scriptSig` nastaví na `scriptCode`.
7. Upraví `txTmp` podle hodnoty `nHashType`, jak bylo popsáno výše, pro defaultní hodnotu `SIGHASH_ALL` transakci nijak neupraví. Pokud `nHashType = SIGHASH_NONE`, pak smaže všechny výstupy a nastaví `nSequence` všech vstupů kromě toho s indexem `nIn` na nulu. Pokud `nHashType = SIGHASH_SINGLE`, pak smaže všechny výstupy kromě toho s indexem `nIn` a nastaví `nSequence` všech vstupů kromě toho s indexem `nIn` na nulu.
8. Pokud je nastaven modifikátor `SIGHASH_ANYONECANPAY`, pak ještě smaže všechny vstupy kromě toho s indexem `nIn`.
9. Provede serializaci a `txTmp` dvakrát zhešuje funkcí SHA-256.
10. Pomocí `<pubkey>` ověří, zda `<sig>` je skutečně podpisem dat ve tvaru SHA-256(SHA-256(`txTmp`)). Pokud ano, vrátí `true`, jinak vrátí `false`.

4.4 Kontrakty

Kontrakty jsou v tuto chvíli nepoužívanou součástí Bitcoinu. Transakce však v sobě mají navrženy mechanismy, které umožňují jejich budoucí použití. Konkrétně pomocí hodnot nSequence ve vstupu transakce, časového zámku na konci transakce, příznaků v proměnné `nHashType` a skriptovacího systému lze vytvářet složitější transakce, než jaké jsme viděli doposud.

Kontrakt je určitý typ dohody, kterou lze uzavřít prostřednictvím Bitcoinu s jinými lidmi. Prostřednictvím kontraktu lze vytvořit transakci, ve které jednotlivé vstupy pochází od různých lidí, kteří tak mohou uskutečnit společný nákup. Následující příklad ukazuje princip tvorby kontraktů.

4.4.1 Kontrakt na zálohu

Příklad byl převzat ze stránky Contracts na [3].

Představme si, že máme uživatele, který si chce na půl roku založit účet na nějaké webové stránce, kde je třeba zaplatit zálohu `x` BTC, která mu bude po uzavření účtu vrácena. Pokud je uživatel důvěřivý, tak pošle danou částku na

příslušný účet, ale nikdo mu nezaručí, že, až účet uzavře, provozovatelé webu mu zálohu pošlou zpět. Nedůvěřivý uživatel si proto vytvoří kontrakt. Vytvoření kontraktu nejenže zajistí, že peníze dostane zpátky, kontrakt dokonce znemožní druhé straně tyto peníze utratit. Až nadejde ten pravý čas, budou uživateli zaslány bez ohledu na to, zda daný web bude fungovat, či nikoliv. Kontrakt se vytvoří následujícím způsobem:

1. Uživatel si s provozovatelem vymění nově vygenerované veřejné klíče.

2. Uživatel vytvoří transakci Tx_1 s výstupem v hodnotě `x` BTC, přístup k této částce bude ve skriptu `scriptPubKey` podmíněn přítomností podpisu jak uživatele, tak provozovatele. Jedná se o podpisy, které přísluší veřejným klíčům z kroku 1. Uživatel prozatím transakci Tx_1 nezveřejní, jen pošle její heš provozovateli.

3. Provozovatel vytvoří transakci Tx_2, která se vstupem odkazuje na výstup Tx_1 (tedy utrácí bitcoiny z Tx_1) a posílá bitcoiny tohoto výstupu zpět uživateli na adresu určenou veřejným klíčem z kroku 1. Časový zámek v Tx_2 je nastaven na dobu o šest měsíců později a nSequence je nastavena na nulu. Kdybychom v tuto chvíli chtěli ověřit platnost Tx_2, tj. vyhodnotili bychom `scriptPubKey` z výstupu Tx_1 a následně `scriptSig` ze vstupu Tx_2, byla by Tx_2 označena jako neplatná, protože `scriptSig` obsahuje pouze podpis provozovatele, kdežto `scriptPubKey` vyžaduje i podpis uživatele.

4. Provozovatel pošle Tx_2 uživateli, ten si zkontroluje, že je transakce v takové podobě, jak bylo domluveno, a přidá k ní svůj podpis.

5. Následně uživatel uveřejní Tx_1 a Tx_2 v tomto pořadí.

Podstatné je to, že s bitcoiny z výstupu Tx_1 nemůže provozovatel ani uživatel nakládat bez souhlasu (podpisu) toho druhého. A zároveň po uplynutí lhůty půl roku se z transakce Tx_2 stane finální transakce a tím uživatel dostane své peníze zpět. Za tuto dobu se však mnoho věcí může změnit. Právě fakt, že je nSequence nastavena na nulu, umožňuje na různé změny elegantně reagovat. Kupříkladu se může stát, že uživatel bude chtít svůj účet uzavřít okamžitě ještě před ukončením lhůty půl roku. V takovém případě provozovatel vytvoří novou verzi transakce Tx_2 s časovým zámkem nastaveným na nulu a s nSequence nastavenou na maximální možnou hodnotu (transakce je tedy ihned finální), podepíše ji a pošle uživateli. Ten, pokud souhlasí, ji také podepíše, zveřejní ji po bitcoin síti a dostane své peníze zpátky.

Druhou změnou může být to, že uživatel bude chtít prodloužit dobu platnosti zálohy. Proto bude opět vytvořena nová verze Tx_2 s časovým zámkem nastaveným na dobu, kdy chce uživatel účet skončit, a s nSequence nastavenou o jedna vyšší. Podepíše provozovatel a následně uživatel, který ji zveřejní.

Transakci Tx_1 říkáme platba, transakci Tx_2 říkáme kontrakt. Postup při tvorbě kontraktů je takový, že se nejprve vytvoří kontrakt, který se podepíše, ale zůstane neveřejný. Následně se vytvoří platba, která se pošle po bitcoin síti poté, co je kontrakt podepsán všemi zúčastněnými stranami. Jako poslední se zveřejní kontrakt.

5. Bloky a těžba

V této kapitole jsme čerpali ze stránek Protocol_specification, Target, Difficulty, Block_hashing_algorithm, Double-spending, Protocol_rules, Block, Proof_of_work, Mining, Confirmation, Nonce, Transaction_fees na [3], dále pak z [1], [16] a [18].

Nejvýznamnější vlastností Bitcoinu je jeho decentralizovanost. To znamená, že neexistuje člověk, skupina lidí či instituce (jako tomu je u běžných měn v podobě banky), která by do chodu měny mohla jakkoliv zasahovat. Některé funkce, které plní banka, však musí Bitcoin také zajistit. Jeden ze základních požadavků je ochrana proti tzv. *double-spending* (viz sekce 5.3). To znamená, že se nesmí stát, aby nějaká částka bitcoinů byla utracena víckrát. Jak už víme, veškerá manipulace s bitcoiny je realizovaná prostřednictvím transakcí (viz Kapitola 4). Abychom mohli zabránit double-spending, potřebujeme mít přehled o všech transakcích, které se kdy uskutečnily.

Měnu je také nutné nějak emitovat, což u běžných platidel zajišťuje opět banka. Emise bitcoinu je spojena s potřebou zaznamenávat všechny proběhlé transakce. Transakce jsou totiž shromažďovány do tzv. *bloků* a bloky se připojují ke všemi uživateli sdílené historii zvané *block chain* (viz sekce 5.2.4). Každý blok a tedy každá transakce, která je součástí block chain, je považována za platnou. Aby se transakce dostala do block chain, musí být blok, jehož je součástí, k block chain připojen. Ten, kdo se o připojení snaží, se nazývá těžař, a musí nejprve ověřit, že transakce, které připojuje, jsou platné a neobsahují double-spending. Následně musí vyřešit výpočetně náročnou úlohu, tzv. těžařskou úlohu. Těžař, kterému se tato úloha podaří vyřešit, pak může zveřejnit blok po bitcoin síti a říkáme, že vytěžil blok. Každý klient si zkontroluje, že je vytěžený blok v pořádku (bude vysvětleno později) a pokud ano, pak jej tzv. akceptuje.

Proč by se však někdo namáhal řešit výpočetně náročnou úlohu? Motivací každého těžaře je odměna v podobě transakčních poplatků a nově vytvořených bitcoinů (v současné chvíli 25 BTC za jeden blok), kterou dostane ten, kdo vytěží blok, jenž je akceptován bitcoin sítí. Jak vidíme, tak block chain a jeho prodlužování plní jak funkci archivu všech uskutečněných platných transakcí, tak i funkci emise měny. Dokonce je vnitřním mechanismem tempo emise bitcoinů regulováno tak, aby se vytěžilo v průměru 6 bloků za hodinu, tj. $25 \cdot 6 = 150$ BTC za hodinu. Podrobnosti jsou uvedeny ve zbytku kapitoly.

5.1 Bloky

Blok je množina transakcí, které jsou reprezentovány tak, jak uvádí Kapitola 4.

Definice 12. *Vytěžený blok akceptovaný Bitcoin sítí nazýváme platný.*

První transakcí každého bloku je coinbase transakce (viz sekce 4.2 v Kapitole 4). Každý blok obsahuje odkaz na nějaký jiný, tzv. předchozí blok.

Definice 13. *Odkazuje-li se blok B_2 na blok B_1, pak říkáme, že B_2 je (bezprostřední) následník B_1.*

Bitcoin je decentralizovaný, proto není žádná instituce ani důvěryhodná autorita, která by vzala blok a prohlásila jej za platný. Blok sám o sobě musí obsahovat

informace, na základě kterých je akceptován celou uživatelskou sítí bitcoinu. To znamená, že každý uživatel musí mít možnost ověřit, že blok splňuje to, co má, aby se stal platným (podrobnosti viz sekce 5.2).

Blok je řetězec bajtů se strukturou uvedenou v tabulce 5.1. Podoba bloků je také ilustrována v Příloze A.

DATA	POPIS
version	Verze bloku závislá na software, kterým byl vytvořen
prev_block	Heš (SHA-256(SHA-256)) hlavičky předchozího bloku
merkle_root	Kořen hešovacího stromu
timestamp	Čas, kdy byly transakce seskupeny do bloku
bits	Reprezentace target, viz 5.2.3
nonce	Čtyřbajtová hodnota, jejíž iterováním se těžař snaží vyřešit výpočetně náročnou úlohu
txn_count	Počet transakcí obsažených v bloku
txns	Výčet transakcí tohoto bloku

Tabulka 5.1: Struktura bloku *(Převzato ze stránky Protocol_specification na [3])*

Prvním šesti položkám říkáme hlavička bloku.

Když se vytváří blok, vezmou se všechny v něm obsažené transakce a z nich se sestaví binární hešovací strom (viz Kapitola 2.5.1). Jako hešovací funkce je použita SHA-256(SHA-256()). Pokud je počet všech vrcholů stromu v nějaké hloubce lichý, pak je poslední vrchol zdvojen. Kořen stromu je součástí hlavičky bloku.

Podle protokolu každý blok musí obsahovat alespoň jednu transakci, a to coinbase.

5.2 Těžba

Těžení je proces, kterým se připojuje blok k block chain.

5.2.1 Odměna

Pokud se připojení podaří, pak je těžař daného bloku odměněn bitcoiny, které se tím nově vytvoří. Velikost odměny není konstantní. Na začátku činila 50 BTC a následně se každých 210 000 bloků sníží na polovinu (což vychází zhruba na každé 4 roky vzhledem k tomu, že se vytěží v průměru 6 bloků za hodinu). Odměna slouží jako motivace k těžbě, prostřednictvím které se zajišťuje bezpečnost celé bitcoin sítě. Druhá část odměny jsou všechny poplatky obsažené v transakcích, jež jsou součástí vytěženého bloku. Transakční poplatek je impulsem pro těžaře, aby danou transakci do svého bloku zahrnuli.

Množství vytěžitelných bitcoinů je předem omezené na částku 21 000 000 BTC. Až budou všechny tyto bitcoiny vytěženy, budou to právě transakční poplatky, které budou motivovat těžaře, aby ve své činnosti pokračovali.

Výše poplatku závisí čistě na autorovi transakce. Běžné transakce většinou neobsahují žádný. Na druhou stranu je poplatek nástroj, jakým lze zajistit, aby daná transakce byla co nejdříve připojena k block chain. Těžaři sami si určují strategie, podle kterých sestavují své bloky.

5.2.2 Těžařská úloha

Jak už jsme uvedli, ten, kdo usiluje o připojení nějakého bloku B s hlavičkou H, musí splnit výpočetně náročnou úlohu. Úloha spočívá v tom najít takovou hodnotu nonce, aby

$$\text{SHA-256}(\text{SHA-256}(H)) < \text{target} \tag{5.1}$$

Target je 256 bitové veřejné číslo sdílené bitcoin uživateli, které určuje složitost těžařské úlohy. Hlavička bloku představuje tzv. *proof of work*, tedy takovou část dat, která splňuje určité požadavky, jež jsou snadno ověřitelné, a kterou je výpočetně náročné získat. Požadavek na proof of work je dán nerovností 5.1. Protože v ní vystupuje hešovací funkce SHA-256, říkáme, že jde o *hashcash proof of work*.

Funkce SHA-256 je považována za kryptografickou hešovací funkci (viz definice 2.1 v Kapitole 2). Proof of work využívá její vlastnosti preimage resistance, protože chceme, aby bylo náročné najít takovou hodnotu H, kdy obraz H po dvojnásobné aplikaci funkce SHA-256 bude ležet v intervalu $\langle 0, \text{target})$. Také platí, že chování SHA-256 je zcela nepředpověditelné (každá změna nonce úplně změní výsledný heš) a těžařům nezbývá než zkoušet iterovat nonce a ověřovat, zda už je heš dostatečně malý. To vyžaduje velkou výpočetní sílu, pokud chce být těžař úspěšný, a pochopitelně čím je target menší, tím obtížnější a méně pravděpodobné je najít vhodnou nonce. Náročnost hešování bloku nezávisí na tom, kolik obsahuje transakcí, jelikož jsou zhešovány nepřímo prostřednictvím kořene merkle stromu (a ten je vždy stejně velký).

Pokud nonce při iterování překročí maximální hodnotu, jaké může nabývat, pak se vynuluje a zvýší se hodnota tzv. *extranonce*, jež je součástí `scriptSig` (viz sekce 4.3 v Kapitole 4) jediného vstupu coinbase transakce daného bloku. Tím se tato transakce a tedy merkle_root bloku změní a můžeme znovu inicializovat a následně iterovat nonce, aniž bychom hešovali identicky stejnou hlavičku bloku.

Pokud je těžař úspěšný, pak zveřejní blok na bitcoin síti, kde každý klient může snadno ověřit, že těžař skutečně našel vhodnou nonce.

Není žádoucí, aby všichni těžaři zkoušeli těžit stejný blok, protože by si někdo mohl počkat, až těžař blok vytěží, pak mu blok zcizit a vydávat ho za vlastní. Ochranu proti tomuto případu zajišťuje coinbase transakce, která v sobě obsahuje unikátní adresu těžaře, a činí tak každý blok jedinečný (prostřednictvím merkle_root v hlavičce bloku).

Jakmile je blok vytěžen, nesmí být změněn, aniž by byl znovu vytěžen. Znamená to znovu zkoušet řešit těžařskou úlohu. Jak jsou bloky řetězeny za sebe, změna jednoho bloku znamená, že musí být znovu vytěženi i všichni jeho následníci.

5.2.3 Target

Jak už jsme uvedli, target je číslo určující složitost těžařské úlohy. Čím nižší je jeho hodnota, tím vyšší je složitost těžby bloku. V každém bloku je target uveden ve zvláštním formátu v proměnné bits. Ta obsahuje čtyři bajty, první bajt je exponent e, ze kterého se používá pouze 5 nejméně významných bitů, ty označíme e', a další tři bajty představují mantisu m. Výsledný target je pak

$$\text{target} = m \cdot 256^{(e'-3)} = m \cdot 2^{8\cdot(e'-3)} \tag{5.2}$$

Je-li bits = $(1B0404CB)_{16}$, pak $e = (1B)_{16}$ a $m = (0404CB)_{16}$ a target =$(00000000000404CB000000000000000000000000000000000000000000000000)_{16}$

Čím více lidí těží, tím roste tempo těžby bloků. Síť na to reaguje tak, že zvýší obtížnost a tím reguluje tempo těžby, aby byl blok vytěžen zhruba jednou za deset minut. Složitost těžby je tedy dynamicky měněna každých 2016 bloků (cca jednou za dva týdny). Výpočet target probíhá následujícím způsobem. Spočteme dobu Y, za jakou bylo oněch 2016 bloků vytěženo (pomocí proměnné timestamp v hlavičce bloku). Nový target se spočítá podle vzorce:

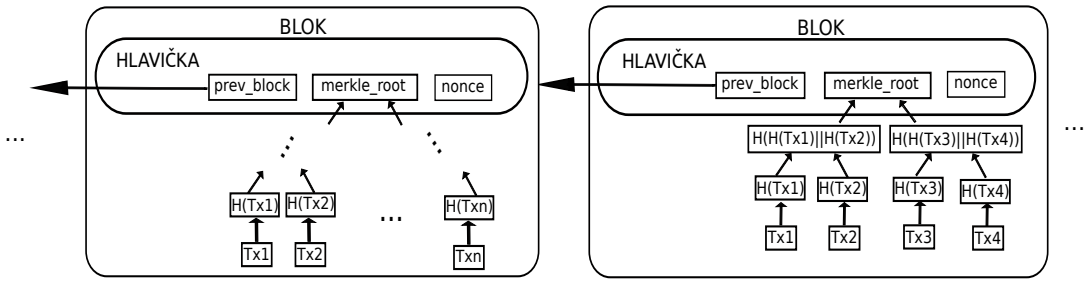
$$\text{target}_{\text{nový}} = \text{target}_{\text{starý}} \cdot \frac{Y}{2 \text{ týdny}} \tag{5.3}$$

Maximální možný target je konstanta MaxTarget =$(00000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)_{16}$

Definice 14. *Složitost (vytěženého) bloku B značíme $d(B)$ a definujeme ji jako $d(B) = \frac{MaxTarget}{target}$, kde target ve jmenovateli je takový target, se kterým byl blok B vytěžen.*

5.2.4 Block chain

Block chain je všemi bitcoin klienty sdílená historie transakcí, jejíž ilustraci vidíme na obrázku 5.1.



Obrázek 5.1: Block chain

Každý blok se vždy odkazuje na nějaký předchozí blok. Výjimkou je první blok, který kdy existoval, tzv. *genesis* blok s datem 3.1.2009, jehož tvůrcem je Satoshi Nakamoto. Z toho plyne, že bloky tvoří strom, jehož kořenem je právě genesis blok. Tento strom budeme nazývat *bitcoin strom*. Větví pak budeme rozumět cestu z kořene do libovolného jeho listu. Každá větev stromu představuje jednu verzi historie transakcí. Všichni bitcoin klienti vždy za platnou historii a tedy block chain považují právě jednu větev bitcoin stromu a to takovou, že součet složitostí všech jejích bloků je největší.

Definice 15. *Jsou-li V_1 a V_2 dvě větve bitcoin stromu, říkáme, že V_1 je delší než V_2, pokud součet složitostí bloků obsažených ve V_1 je větší než součet složitostí bloků obsažených ve V_2.*

Ve smyslu předchozí definice je za block chain považována vždy nejdelší větev bitcoin stromu.

Definice 16. *Řekneme, že transakce je konfliktní, pokud buď není platná, nebo uskutečňuje double-spending (snaží se tedy znovu utratit bitcoiny, které už jednou utraceny byly).*

Protože každá větev bitcoin stromu může být potenciálně block chain, nesmí žádná větev obsahovat konfliktní transakci. Jedna větev však pochopitelně může obsahovat transakci, která je konfliktní s nějakou jinou z odlišné větve.

Definice 17. *Pokud různí bitcoin klienti považují za block chain různé větve bitcoin stromu, říkáme, že nastalo block chain rozvětvení.*

K block chain rozvětvení dochází např. ve chvíli, kdy jsou dva bloky vytěženy souběžně a oba tak prodlužují block chain stejným příspěvkem složitosti. Část sítě si připojí jeden blok, část sítě druhý. Tato dvojakost však nastane vždy jen dočasně, neboť jednou se některá z větví prodlouží a ta druhá bude klienty zneplatněna.

Reprezentace

Block chain je řetězec bajtů tvaru:
...||magic number||block size||blok$_i$||magic number||block size||blok$_{i+1}$||... kde:

- Magic number má hodnotu $(\text{F9BEB4D9})_{16}$ a slouží jakožto identifikátor všech zpráv, které se šíří mezi bitcoin klienty, a zároveň jako oddělovač jednotlivých bloků v block chain.
- Block size je délka bloku, který bude následovat, v bajtech.
- $blok_i$ má pro $\forall i$ podobu uvedenou v sekci 5.1.

5.2.5 Protokol klienta

Zde uvedeme neúplný protokol, kterým se řídí každý bitcoin klient. Úplnou verzi můžete najít na Protocol_rules na [3].

Každý klient si rozděluje transakce do tří skupin:

1. Transakce je součástí block chain
2. Transakce Tx není součástí block chain, ale máme k dispozici všechny transakce, na které se vstupy Tx odkazují. Takovéto množině transakcí říkáme *transaction pool*.
3. Pro transakci Tx nemáme k dispozici všechny transakce, na které se vstupy Tx odkazují. Množinu těchto transakcí nazýváme *orphan transactions*.

I bloky se dělí do tří kategorií:

1. Bloky obsažené v block chain.
2. Bloky obsažené v bitcoin stromu, ale mimo nejdelší větev.
3. Bloky tvořící tzv. *orphan blocks*. To jsou bloky nepřipojené k bitcoin stromu, většinou z toho důvodu, že chybí blok, na něž se odkazují.

Bitcoin síť žije následujícím způsobem:

- Nové transakce se zveřejňují po síti.
- Když klient obdrží novou transakci Tx, pak:
  - Pokud Tx nemá obvyklou podobu tak, jak je popsána v kapitole Transakce 4, klient ji odmítne.
  - Pokud není k dispozici některý z výstupů, na který se Tx odkazuje, pak klient uloží Tx do orphan transactions a pro tuto chvíli s Tx nic dalšího nedělá.
  - Pokud Tx uskutečňuje double-spending nebo není platná, klient ji odmítne.
  - Klient přidá Tx do transaction pool.
  - Klient vyšle Tx po síti.
  - Klient provede výše uvedené kroky pro všechny transakce z orphan transactions, které se na Tx odkazují.
- Každý klient vytváří blok z transakcí obsažených v transaction pool a snaží se jej vytěžit.
- Pokud nějaký klient blok vytěží, pak jej zveřejní po bitcoin síti.
- Obdrží-li klient nový blok B s hlavičkou H, pak provede následující:
  - Pokud B není v obvyklé podobě, jaká je popsána v 5.1, klient ho odmítne.
  - Musí platit, že SHA-256(SHA-256(H)) $<$ target pro aktuální hodnotu target, jinak klient blok odmítne.
  - Pokud blok, na který se B odkazuje, není v bitcoin stromu, klient uloží B do seznamu orphan blocks. Jinak přidá B do bitcoin stromu. Tím mohou nastat tři situace:
    1. Blok B prodlužuje block chain (tj. odkazuje se na blok, který tvoří v aktuální verzi block chain list).
    2. Blok B prodlužuje nějakou jinou větev bitcoin stromu, nikoliv však tak, aby se stala nejdelší.
    3. Blok B prodlužuje nějakou jinou větev V bitcoin stromu, ta se tím stává nejdelší větví a tedy aktuálním block chain.
  - V případě 1 klient ověří, že pro každou transakci Tx (kromě coinbase) bloku platí:

(a) V block chain jsou k dispozici všechny transakce, na které se Tx svými vstupy odkazuje.

(b) Tx je platná a neuskutečňuje double-spending.

Pokud některá z podmínek (a), (b) neplatí, blok je klientem odmítnut. Jinak transakce z B smaže z transaction pool, pokud tam jsou, a pošle B po síti.

- V případě 2 klient nedělá nic.
- V případě 3 klient najde takový blok B_1 z block chain, který je posledním společným blokem pro dosavadní block chain a větev V. Pro každý blok větve V počínaje následníkem B_1 až do listu provede kontrolu, zda jsou bloky ve standardní podobě a zda splňují těžařskou úlohu. Pro všechny transakce těchto bloků se ověří podmínky (a) a (b). Selže-li nějaká kontrola, B je odmítnut a block chain se nemění. Jinak každou transakci (kromě coinbase) bloků původního block chain počínaje následníkem B_1 až do listu vložíme do transaction pool. Vyšleme B po síti.
- Pro každý blok z orphan blocks, který se odkazuje na B, provede všechny kroky, které aplikoval na B.

Jak je vidět výše, každá transakce, jež je součástí block chain, je platná. Odmítnutí transakce, resp. bloku, znamená, že klient už se s danou transakcí, resp. blokem, nebude dále zabývat. Akceptování bloku klientem je realizováno tím, že daný klient začne zkoušet vytěžit blok, který se bude odkazovat právě na akceptovaný blok.

5.2.6 Potvrzení transakce

Součástí těžby bloku je ověření platnosti všech transakcí (kdyby se tak nestalo, ostatní klienti by blok s neplatnou transakcí odmítli). Pokud je blok B úspěšně vytěžen a připojen k block chain, pak o každé transakci, která v něm leží, říkáme, že byla jednou potvrzena. Pro každý další blok, který prodlouží block chain, se počet potvrzení transakcí bloku B zvýší o jedna.

Definice 18. *Počet potvrzení transakce Tx bloku B je roven počtu bloků v block chain na cestě z bloku B do listu.*

My už však víme, že block chain se může dočasně větvit, proto jako ochrana proti double-spending je dobré považovat transakci za tzv. *bezpečně potvrzenou*, pokud dosáhne určitého počtu potvrzení. Jinak řečeno pokud leží v určité hloubce block chain.

Mechanismem těžby je zaručeno, že pro vytěžení daného bloku musela být vynaložena určitá práce. Jednotlivé platné bloky se řetězí za sebe do block chain, který tak představuje velké množství práce, jež musela být při jeho tvorbě uskutečněna. Pokud chce tedy někdo změnit blok B, který je akceptovaný zbytkem bitcoin sítě, pak musí vytvořit nový, jenž je následníkem stejného bloku jako B, a poté musí znovu vytěžit všechny bloky, které ležely v block chain pod B. To však představuje potřebu tak velké výpočetní síly, že to v praxi od určité hloubky není proveditelné. Jde o ochranu před neoprávněným zásahem.

Většina klientů a směnáren považuje transakci za bezpečně potvrzenou, pokud je potvrzena alespoň šesti bloky. Je-li transakce bezpečně potvrzená, pak by její změna vyžadovala velkou výpočetní sílu na znovuvytěžení bloku, jemuž náleží, i všech bloků po něm. Potřeba výpočetní síly na tuto změnu tak roste spolu s hloubkou, v jaké transakce je. Proto je bezpečně potvrzená transakce považována za nezměnitelnou.

Stejně tak nově vytěžené bitcoiny potřebují nějakou dobu, resp. hloubku, která je potvrzuje. Důvod je ten, aby se daná coinbase transakce dostala ke všem uživatelům v síti. U běžného bitcoin klienta je doba potvrzení nejméně 100 bloků.

5.3 Double-spending

Jestliže existuje transakce, na jejíž výstup se odkazuje více než jeden vstup, pak nastává double-spending. Představme si, že uživatel U si chce koupit nějaké zboží od obchodníka O. Proto U vytvoří transakci Tx_1, která uskuteční platbu O za dané zboží. Navíc má však U nekalé úmysly a bude se snažit provést double-spending. To znamená, že poté, co mu bude posláno zboží od O, tak bude chtít pozměnit Tx_1 na Tx_2 tak, aby se posílané bitcoiny vrátily zpátky na jeho adresu místo toho, aby zůstaly obchodníkovi. Jak vidíme, Tx_1 je konfliktní s Tx_2 (a naopak), to znamená, že obě současně nemohou ležet v block chain. Dále vidíme, že Tx_1 musí být obsažena v block chain, jinak by O neposlal zboží, protože by Tx_1 nebyla potvrzená.

Pokud chce U provést útok, pak musí nejprve vytvořit Tx_1 a poslat ji po síti. Dále musí neveřejně vytěžit blok, který je následníkem takového bloku, který je v block chain poslední před blokem obsahujícím Tx_1, čímž si založí vlastní větev bloků V_2. Jeho soukromý blok bude obsahovat Tx_2. Útočník následně čeká do doby, než se Tx_1 v block chain dostane dostatečně hluboko, aby mu O poslal zboží. Dokud není zboží posláno, tak svou soukromou větev prodlužuje, tj. připojuje k ní další neveřejně vytěžené bloky tak, aby ve chvíli, kdy je zboží posláno, mohl svou soukromou větev zveřejnit a ta byla delší než do té doby oficiální větev block chain V_1 obsahující Tx_1. Pokud se mu toto podaří, pak je jeho větev, protože je delší, zaměněna s větví obsahující Tx_1. Rázem je to Tx_2, která je v block chain a tedy potvrzená, a tímto útočník zpátky získává peníze původně poslané obchodníkovi. Navíc však má i zboží, aniž by utratil jediný bitcoin.

Aby byl útok úspěšný, musí útočník disponovat tak velkou výpočetní silou, aby mohl vytvořit soukromou větev, která bude delší než jí odpovídající větev v block chain. Musí být tedy schopen těžit rychleji, než zbytek sítě. Je však jasné, že čím nedůvěřivější obchodník bude, tedy čím větší počet potvrzení Tx_1 bude vyžadovat, tím složitější to bude pro útočníka. Proto jsou požadavky obchodníků na počet potvrzení transakce jedním z důležitých ochranných prvků proti double-spending.

5.3.1 Analýza útoku

Následující analýza byla převzata z [18], [16] a doplněna o kroky a vysvětlení, které ve zdrojích chybí. Mějme útočníka U a ostatní klienty bitcoin sítě S. Předpokládejme, že výpočetní síla H útočníka a sítě dohromady je konstantní taková,

že U má výpočetní sílu qH a S má výpočetní sílu pH tak, že platí:

$$p + q = 1 \tag{5.4}$$

Dále předpokládejme, že složitost všech bloků je konstantní a konstantní je i průměrný čas vytěžení nového bloku při výpočetní síle H. Nechť

$$\begin{aligned} m &:= |V_2| \\ n &:= |V_1| \\ z &:= n - m \end{aligned}$$

Proměnná z představuje náskok sítě S nad útočníkem. Vytěží-li S nový blok, což se stane s pravděpodobností p, pak se z zvýší o jedna. Naopak vytěží-li nový blok U s pravděpodobností q, pak se z sníží o jedna. Nastane-li situace, kdy $z = -1$, pak útočníkova větev V_2 je delší a tedy útočník provedl úspěšný double-spending. Nás bude zajímat, zda někdy (lhostejno kdy) nastane situace, kdy $z = -1$, a to v závislosti na hodnotách p a q.

Řekneme, že útočník vyhraje, bude-li někdy v budoucnosti schopen vytěžit tolik bloků, aby $m > n$. Nezajímá nás, kdy se to stane, ale jestli vůbec. Definujme $a_z := P$(útočník, který je z bloků pozadu, vyhraje). Jistě platí, že je-li $z < 0$, pak $a_z = 1$, protože útočník už má delší větev než S.
Je-li $z \geq 0$, pak buď nový blok vytěží útočník (s pravděpodobností q) a tím pádem sníží z o jedna, nebo nový blok vytěží S s pravděpodobností p a zvýší tak z o jedna. Tedy

$$a_z = p \cdot a_{z+1} + q \cdot a_{z-1} \tag{5.5}$$

Následně použijeme metodu řešení lineárních diferenčních rovnic uvedenou v kapitole 11.2. v knize [19]. Rovnici 5.5 upravíme na tvar (převedeme nenulové členy na levou stranu a využijeme vztahu 5.4)

$$(p - 1) \cdot a_{z-1} + a_z - p \cdot a_{z+1} = 0 \tag{5.6}$$

Charakteristický polynom rovnice 5.6 ve smyslu výše uvedené kapitoly má tvar :

$$\chi \;=\; p - 1 + x - p \cdot x^2 \tag{5.7}$$

Polynom 5.7 má právě dva jednonásobné kořeny, a to :

$$\begin{aligned} x_1 &= 1 \\ x_2 &= \frac{q}{p} \end{aligned}$$

Řešení rovnice 5.6 má tvar :

$$a_z = a \cdot 1^z + b \cdot (\frac{q}{p})^z \tag{5.8}$$

pro nějaké $a, b \in \mathbb{R}$. Pro určení koeficientů a, b potřebujeme alespoň dvě počáteční podmínky. První už máme, tou je $a_{-1} = 1$.

Dále budeme chtít spočítat a_0. To představuje pravděpodobnost, že útočník vyhraje, jestliže začíná se stejně dlouhou větví jako zbytek sítě. Aby U vyhrál,

musí se někdy dostat ze stavu $z = 0$ do stavu, kdy $z = -1$. To se může stát dvěma způsoby. Buď rovnou vytěží následující blok s pravděpodobností q. Nebo následující blok vytěží zbytek sítě (s pravděpodobností p), tím se dostaneme do stavu $z = 1$. Abychom se dostali do $z = -1$, musíme nejprve navštívit stav $z = 0$ a následně projít do $z = -1$. Pravděpodobnost, že projdu ze $z = 1$ do $z = 0$ je stejná, jako že udělám krok ze $z = 0$ do $z = -1$ a to proto, že vše uvažujeme v nekonečném čase (tj. nezáleží na tom, jestli bude nějakou dobu $z > 1$, pokud se do $z = 1$ někdy vrátím). Proto platí

$$a_0 \quad = \quad q + p \cdot a_0 \cdot a_0 \tag{5.9}$$

Tím dostáváme kvadratickou rovnici v proměnné a_0, která má kořeny $a'_0 = 1$ a $a''_0 = \frac{q}{p}$. Je-li $q > p$, pak, protože a_0 představuje pravděpodobnost, dává smysl pouze kořen a'_0. Je-li $q < p$ ukážeme, že kořenem může být pouze a''_0.

Nechť $q < p$ a pro spor předpokládejme, že $a_0 = 1$. Z rovnosti 5.5 plyne, že pak $a_i = 1 \ \forall i > 0$, tedy útočník je schopen vyhrát nezávisle na náskoku zbytku sítě. Ať je U pozadu z bloků a S následně vytěží k bloků. Aby U vyhrál, musí sám vytěžit $z + k + 1$ bloků (dožene náskok sítě a poslední blok musí vytěžit on). U tedy vyhraje po $z + 2k + 1$ vytěžených blocích (S vytěží k, U vytěží $z + k + 1$). Označme P_k pravděpodobnost, že U vyhraje po $z+2k+1$ blocích. Jak odhadnout P_k?

$$P_k \quad \leq \quad \binom{z+2k}{k} p^k q^{z+k} q < 2^z 4^k p^k q^z q^k q < 2^z 4^k p^k q^z q^k \tag{5.10}$$

První nerovnost v 5.10 vyplývá z toho, že na pravé straně nerovnosti jsou započítány i ty případy, kdy by útočník vytěžil svých $z+k+1$ bloků dřív, než by S vytěžila k bloků, takové případy do P_k nezapočítáváme. Protože platí $q < p \wedge p+q = 1$, pak $q < \frac{1}{2} < 1$, což zdůvodňuje poslední nerovnost v 5.10. Pro odhad kombinačního čísla jsme použili nerovnost $\binom{z+2k}{k} \leq 2^{z+2k} = 2^z 4^k$. Odhad plyne z binomické věty takto:

$$\begin{aligned} 2^{z+2k} = (1+1)^{z+2k} \quad &= \quad \sum_{l=0}^{z+2k} \binom{z+2k}{l} 1^{z+2k-l} 1^l = \\ = \sum_{l=0}^{z+2k} \binom{z+2k}{l} \quad &> \quad \binom{z+2k}{k} \end{aligned}$$

Teď spočítáme pravděpodobnost a_z, o které víme, že se z našich předpokladů má rovnat 1:

$$1 = a_z = \sum_{k=0}^{\infty} P_k < \sum_{k=0}^{\infty} 2^z 4^k p^k q^z q^k \leq (2q)^z \sum_{k=0}^{\infty} (4pq)^k \tag{5.11}$$

Budeme chtít ukázat, že řada $\sum_{k=0}^{\infty} (4pq)^k$ konverguje. Jde o geometrickou řadu, která konverguje právě tehdy, když $4pq < 1$. Kdy je $pq < \frac{1}{4}$? Ze 5.4 plyne, že $p = 1 - q$. Tedy $pq < \frac{1}{4} \Leftrightarrow (1-q)q < \frac{1}{4} \Leftrightarrow q^2 - q + \frac{1}{4} > 0$. Vyřešením kvadratické nerovnice dostáváme, že $pq < \frac{1}{4} \Leftrightarrow q \neq \frac{1}{2}$. My máme $q < \frac{1}{2}$, proto řada konverguje, tedy $\sum_{k=0}^{\infty} (4pq)^k =: M$. Dostáváme tak

$$1 = a_z < (2q)^z M \tag{5.12}$$

$q < \frac{1}{2}$, tedy zcela jistě pro nějaké z dostatečně velké je $a_z < 1$, což je spor s tím, že $a_i = 1 \; \forall i > 0$. Proto pro $q < p$ je $1 \neq a_0 = \frac{q}{p}$.
Závěr:
Výpočty výše jsme dostali následující počáteční podmínky:

$$\begin{aligned} a_{-1} = 1 \quad \wedge \quad a_0 = 1 \qquad & \text{pro } q \geq p \\ a_{-1} = 1 \quad \wedge \quad a_0 = \frac{q}{p} \qquad & \text{pro } q < p \end{aligned}$$

Tyto počáteční podmínky použijeme pro výpočet koeficientů a, b v rovnici 5.8 a dostáváme:

$$\begin{aligned} a = 1 \quad \wedge \quad b = 0 \qquad & \text{pro } q \geq p \\ a = 0 \quad \wedge \quad b = \frac{q}{p} \qquad & \text{pro } q < p \end{aligned}$$

Z čehož získáváme finální výsledek výpočtu pravděpodobnosti a_z jako:

$$a_z = 1 \qquad \text{pokud } q \geq p \vee z < 0 \tag{5.13}$$

$$a_z = (\frac{q}{p})^{z+1} \qquad \text{pokud } q < p \wedge z \geq 0 \tag{5.14}$$

Ze vzorce 5.13 vidíme, že drží-li útočník většinu výpočetní síly bitcoin sítě, dokáže provést double-spending nezávisle na náskoku zbytku sítě. Pokud má U jen menšinu celé výpočetní síly (jako v 5.14), pak se jeho šance na úspěšný double-spending exponenciálně zmenšují s rostoucím náskokem zbytku sítě.

Závěr

Cílem práce bylo prozkoumat a pochopit, jak funguje platební systém a virtuální měna Bitcoin, a vytvořit text, který by podával jeho přehledný a ucelený popis. Jediným zdrojem pro takovouto práci je oficiální open source kód bitcoin klienta (viz [1]) a dokumentace v podobě wiki stránek (viz [3]). Oba zdroje však k pochopení vyžadují velké množství času a jistou informačně technologickou zdatnost, informace v nich jsou často uváděny nejasně. Proto si myslíme, že syntéza obou zdrojů, kterou tato práce předkládá, představuje čtivý a ucelený text na dané téma, který čtenáři ušetří mnoho času vyhledáváním jednotlivých informací. Kromě shrnutí tématu jsme navíc dokázali pravděpodobnost kolize bitcoin adres a rozšířili analýzu double-spending, jejíž některé netriviální kroky a postupy byly v [18] a [16] vynechány.

Pro úplné vysvětlení problematiky bylo nezbytné nejprve vysvětlit podobu transakce, popsat bitcoin adresu a následně strukturu všemi uživateli sdílené historie transakcí. Práce se zabývala i tím, jak jsou zajištěny základní bezpečnostní požadavky, bez kterých by si šlo těžko Bitcoin představit jako použitelné platidlo. Zbylé kapitoly poskytují jakýsi matematicko-kryptologický základ, v nichž jsou veškeré používané pojmy zadefinovány i s uvedením jejich základních vlastností.

Virtuální měna Bitcoin představuje složité téma, které tato práce rozhodně nevyčerpala. Například jsme se vůbec nezabývali peněženkami, útoky, jež byly v poslední době provedeny na některé burzy, či tím, jak na Bitcoin pohlížejí jednotlivé státy z právního hlediska. Přesto si myslíme, že cíl, který jsme si vytyčili, jsme naplnili, a text je tak vhodnou příručkou pro každého, kdo by chtěl pochopit princip virtuální měny Bitcoin. Osobní přínos pak spatřujeme v lepší orientaci a čtení zdrojových kódů, ve schopnosti čerpání a kombinace více zdrojů najednou, i ve zvyknutí si na práci s výhradně anglickými matematicky a informaticky zaměřenými texty.

Literatura

[1] bitcoin. In: GitHub [online]. 2013-09-09 [cit. 2013-09-09].
Dostupné z: `https://github.com/bitcoin/Bitcoin`

[2] Bitcoin Block Explorer. In: blockexplorer [online]. 2014-01-09 [cit. 2014-05-05].
Dostupné z: `www.http://blockexplorer.com`

[3] Main page. In: Bitcoin wiki [online]. 2014-05-11 [cit. 2014-05-05].
Dostupné z: `https://en.bitcoin.it`

[4] Merkle tree. In: Wikipedia [online]. 2014-04-27 [cit. 2014-05-05].
Dostupné z: `http://en.wikipedia.org/wiki/Merkle_tree`

[5] Benger, N.; van de Pol, J.; Smart, N. P.; aj. "Ooh Aah... Just a Little Bit" : A small amount of side channel can go a long way. Cryptology ePrint Archive, Report 2014/161, 2014.
Dostupné z: `http://eprint.iacr.org/`

[6] Bernstein, D. J. How to design an elliptic-curve signature system. In: The cr.yp.to blog [online]. 2014-03-23 [cit. 2014-05-05].
Dostupné z: `http://blog.cr.yp.to/20140323-ecdsa.html`

[7] Cerf, V. ASCII format for network interchange. RFC 20, Říjen 1969.
Dostupné z: `http://www.ietf.org/rfc/rfc20.txt`

[8] Certicom Research SEC 2: Recommended Elliptic Curve Domain Parameters. In *Standards for Efficient Cryptography*, 2000.
Dostupné z: `http://www.secg.org/download/aid-386/sec2-final.pdf`

[9] Dobbertin, H.; Bosselaers, A.; Preneel, B. RIPEMD-160: A Strengthened Version of RIPEMD. In *FSE*, *Lecture Notes in Computer Science*, ročník 1039, editace D. Gollmann, Springer, 1996, ISBN 3-540-60865-6, s. 71–82.
Dostupné z: `http://dblp.uni-trier.de/db/conf/fse/fse96.html`

[10] Elgamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, ročník 31, č. 4, 1985: s. 469–472.

[11] Hušková, M.; Dupač, V. *Pravděpodobnost a matematická statistika*. Karolinum, 2013, ISBN 978-80-246-2208-8: s. 1–15.

[12] Johnson, D.; Scott, A. M.; Vanstone The Elliptic Curve Digital Signature Algorithm (ECDSA).
Dostupné z: `http://cs.ucsb.edu/~koc/ccs130h/notes/ecdsa-cert.pdf`

[13] Matoušek, J.; Nešetřil, J.; Karlova, U. *Kapitoly z diskrétní matematiky*. Karolinum, 2009, ISBN 9788024617404: s. 111–154.
Dostupné z: `http://books.google.cz/books?id=TF8MSQAACAAJ`

[14] Menezes, A. J.; Vanstone, S. A.; Oorschot, P. C. V. *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, Inc., první vydání, 1996, ISBN 0-8493-8523-7: s. 321–376, 425–481.

[15] Merkle, R. C. *Secrecy, Authentication, and Public Key Systems*. Dizertační práce, Stanford, CA, USA, 1979, aAI8001972.

[16] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2009. Dostupné z: `http://www.bitcoin.org/bitcoin.pdf`

[17] National institute of standards and technology FIPS 180-2, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-2. Technická zpráva, DEPARTMENT OF COMMERCE, Srpen 2002.

[18] Rosenfeld, M. Analysis of Hashrate-Based Double Spending. 2014. Dostupné z: `http://arxiv.org/abs/1402.2009`

[19] Stanovský, D. *Základy algebry*. Matfyzpress, 2010, ISBN 978-80-7378-105-7: s. 58–63, 75–84.

Seznam obrázků

Seznam tabulek

Příloha A

Následující data pocházejí ze skutečné bitcoin sítě a ilustrují strukturu bloků a transakcí. Z důvodu délky byly z pěti transakcí tohoto bloku uvedeny pouze dvě.

```
Bitcoin protocol
    Packet magic: 0xf9beb4d9
    Command name: block
    Payload Length: 1506
    Payload checksum: 0xc6508a36
    Block message
        version: 2
        prev_block: 6c1f6c6b3a132fc64f807af450106ed30df048f21c6...
        merkle_root: 682be467d745391055bc1f3263ffccb50f1044ce51...
        timestamp: Oct  6, 2013 01:45:31.000000000
        bits: 0x191cdc20
        nonce: 0x95276b5f
        Number of transactions: 5
        Tx 1:
            version: 1
            vin_sz: 1
            in:
                Previous output
                    hash: 0000000000000000000000000000000000000000...
                    index: 4294967295
                script length: 76
                scriptSig: 034aff03094269744d696e746572062f5032...
                    data len: 0x03 (3)
                    data: 4aff03
                    data len: 0x09 (9)
                    data: 4269744d696e746572
                    data len: 0x06 (6)
                    data: 2f503253482f
                    data len: 0x2c (44)
                    data: fabe6d6d69c32771463210c953e913f03da2f0...
                    data len: 0x09 (9)
                    data: 757331b70f00000d63
                nSequence: 4294967295
            vout_sz: 1
            out:
                value: 2500265460
                script length: 25
                scriptPubKey: 76a9145c0e4a6830ff6ea9aea773d75...
                    Opcode: OP_DUP (0x76)
                    Opcode: OP_HASH160 (0xa9)
                    data len: 0x14 (20)
                    data: 5c0e4a6830ff6ea9aea773d75bc207299cd50b74
                    Opcode: OP_EQUALVERIFY (0x88)
```

```
            Opcode: OP_CHECKSIG (0xac)
    lock_time or block ID: 0
Tx 2:
    version: 1
    vin_sz: 2
    in:
        Previous output
            hash: 6a695e62a70581e0cfc80738f81237b064f7c...
            index: 0
        script length: 108
        scriptSig: 493046022100daa480c073e1ae6dab756b156...
            data len: 0x49 (73)
            data: 3046022100daa480c073e1ae6dab756b15634...
            data len: 0x21 (33)
            data: 03e3caad87ec666b936e3a0c60a37a328fda1...
        nSequence: 4294967295
        Previous output
            hash: aff4cbfac86c10078442d6a27ecc92510719...
            index: 0
        script length: 107
        scriptSig: 483045022100a60c49d6072a19beee9c4899...
            data len: 0x48 (72)
            data: 3045022100a60c49d6072a19beee9c48995bd...
            data len: 0x21 (33)
            data: 02e4e5917cc8be60f79f40d99a5fc6e27fdc1...
        nSequence: 4294967295
    vout_sz: 2
    out:
        value: 13430000
        script length: 25
        scriptPubKey: 76a91463a6755aa023f1a7c2ed106f5d9...
            Opcode: OP_DUP (0x76)
            Opcode: OP_HASH160 (0xa9)
            data len: 0x14 (20)
            data: 63a6755aa023f1a7c2ed106f5d9d63d0b95c9b91
            Opcode: OP_EQUALVERIFY (0x88)
            Opcode: OP_CHECKSIG (0xac)
        value: 1000000
        script length: 25
        scriptPubKey: 76a914c7aa893bfa6ccef7593830656e7...
            Opcode: OP_DUP (0x76)
            Opcode: OP_HASH160 (0xa9)
            data len: 0x14 (20)
            data: c7aa893bfa6ccef7593830656e717b160533ff32
            Opcode: OP_EQUALVERIFY (0x88)
            Opcode: OP_CHECKSIG (0xac)
    lock_time or block ID: 0
```