FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	FITCOIN: struktura peněženky
Student:	Lukáš Dang
Vedoucí:	Mgr. Jan Starý, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

1. Popište strukturu peněženky kryptoměny Bitcoin, případně i dalších kryptoměn, pokud se u nich tato struktura podstatně liší.
2. Navrhněte a implementujte strukturu peněženky pro triviální kryptoměnu Fitcoin, která momentálně sestává jen ze souborů v jednom adresáři: obsahem souboru je soukromý klíč, jménem souboru je odpovídající veřejný klíč, který je zároveň platební adresou. Taková struktura je únosná jen při malém počtu klíčů.
3. Proveďte stručnou rešerši "malých" databází dostupných pro UNIXové systémy (BerkeleyDB, SQLite, ...) a diskutujte, které z nich (pokud nějaké) jsou k tomuto účelu vhodné.
4. Implementaci proveďte v jazyce C. Dbejte na korektnost, čitelnost a přenositelnost kódu, přinejmenším na POSIX platformách (BSD, MacOS, Linux, Solaris).
5. Implementaci řádně otestujte, své testování zdokumentujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 7. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

FITCOIN: struktura peněženky

Lukáš Dang

Katedra softwarového inženýrství
Vedoucí práce: Mgr. Jan Starý, Ph.D.

30. července 2020

Poděkování

Rád bych tímto poděkoval svému vedoucímu, Mgr. Janu Starému, PhD., za jeho vstřícnost, trpělivost a čas, který mi věnoval při vedení mé bakalářské práce. Dále bych chtěl poděkovat své rodině, která mě při psaní podporovala.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 30. července 2020 . . . . . .

Odkaz na tuto práci

Dang, Lukáš. *FITCOIN: struktura peněženky*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce se zabývá strukturou peněženky používanou pro kryptoměnu Bitcoin. Současně popisuje kryptografické metody, které jsou v kryptoměnách využívány. Práce je součástí projektu FITCOIN, který si klade za cíl vybudovat co možná nejjednodušší kryptoměnu. Práce do projektu přispívá návrhem a implementací nové struktury peněženky, která je únosná i pro větší počet klíčů. Pro tento účel práce analyzuje databáze dostupné pro UNIXové systémy a diskutuje, které z nich jsou k tomuto účelu vhodné.

Klíčová slova FITCOIN, kryptoměna, peněženka, transakce, Bitcoin, kryptografie, implementace

Abstract

This thesis aims to analyze the wallet structure used in the Bitcoin cryptocurrency. The cryptographic methods used in cryptocurrencies are also described. This thesis is part of a project called FITCOIN, whose goal is to create a cryptocurrency, keeping it as simple as possible. This thesis contributes to the project by designing and implementing a new wallet structure, which is practical to use even for a larger number of keys. For this reason, databases available for UNIX systems are analyzed and a discussion about which of them is suitable for this purpose is held.

Keywords FITCOIN, cryptocurrency, wallet, transaction, Bitcoin, cryptography, implementation

Obsah

Seznam obrázků

Seznam tabulek

Cíle a struktura práce

Jelikož je fungování kryptoměny založeno na kryptografii, je prvním cílem bakalářské práce vysvětlit kryptografické metody využívané v kryptoměnách (kapitola 1). Dalším cílem je seznámit čtenáře s obecnými principy fungování kryptoměn (kapitola 2). Následujícím cílem je analyzovat strukturu peněženky kryptoměny Bitcoin (kapitola 3). Spolu s tím práce provede stručnou rešerši databází dostupných pro UNIXové systémy a diskutuje, které z nich se hodí pro účely praktické části (kapitola 4).

Cílem praktické části je pro kryptoměnu FITCOIN (kapitola 5) navrhnout strukturu peněženky, která bude únosná i pro větší počet klíčů (kapitola 6). Součástí práce je i implementace navržené struktury.

Úvod

Digitální měna je označení pro měnu, která existuje čistě v elektronické podobě. Její používání se stalo běžnou součástí našeho života, setkáme se s ní například v internetovém bankovnictví. V minulosti však byly digitální měny vždy spravovány centrální bankou, což sebou neslo několik nevýhod: převody byly pomalé, drahé, netransparentní a neanonymní. Pro odstranění této závislosti na centrální autoritě vznikl nový typ digitální měny, který využívá metody moderní kryptografie a technologii blockchain. Tato měna nese název kryptoměna.

Kryptoměny dnes již mohou nabídnout reálné využití. Některé kamenné a internetové obchody, mezi nimiž patří například Alza.cz, umožňují platit pomocí Bitcoinu, hlavního reprezentantu kryptoměn. Tato měna vznikla v roce 2008 a dodnes je stále vyvíjena a zlepšována. Chceme-li přispět do vývoje této technologie, je nezbytné pochopit základní principy, na kterých je založeno fungování kryptoměn. Za tímto účelem byl vytvořen projekt FITCOIN. Projekt pochází z FIT ČVUT v Praze a klade si za cíl zanalyzovat nezbytné základy pro fungování Bitcoinu a vybudovat tak co možná nejjednodušší funkční kryptoměnu. Tato kryptoměna se nazývá FITCOIN a je vyvíjena v rámci bakalářských prací studenty Fakulty informačních technologií ČVUT. Vedoucím celého projektu je pan Mgr. Jan Starý, PhD.

Hlavním přínosem této práce je její přispění do projektu FITCOIN návrhem a implementací nové struktury peněženky, která slouží k uchování klíčů uživatele a vytváření transakcí. Osobním přínosem výběru tohoto tématu je hlubší pochopení struktury peněženky kryptoměny Bitcoin a obecně principu fungování kryptoměn.

Kapitola 1

Kryptografie

Kryptoměna, jak již vyplývá z jejího názvu, využívá pro svoje fungování kryptografii. Z tohoto důvodu byla v této práci vyhrazena kapitola vysvětlující kryptografické metody využívané v kryptoměnách. Na začátku bude vysvětlen pojem hašovací funkce jakožto základní stavební kámen kryptoměn. Následně bude vysvětlen pojem kryptografický klíč a způsob jeho generování.

Přestože není digitální podpis přímo součástí tématu této práce, je nezbytným základem pro pochopení fungování kryptoměn, a proto byla pro úplnost o něm zahrnuta podkapitola, která stručně popíše jeho princip.

1.1 Kryptografická hašovací funkce

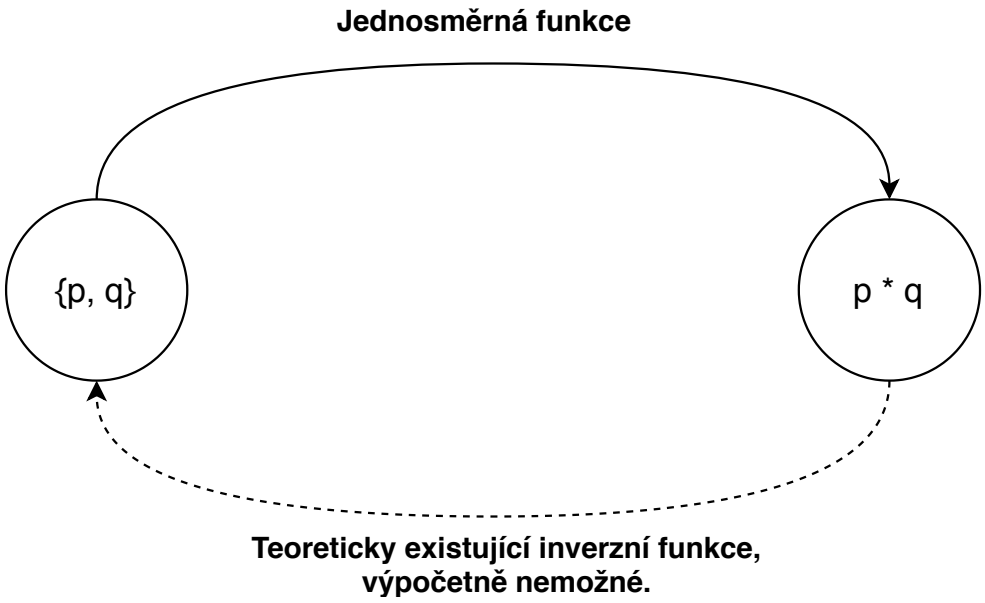
Kryptografická hašovací funkce je silný nástroj moderní kryptografie, který přinesl řadu nových použití. V této práci bude popsáno její využití pro generování klíčů (1.2.3). Na začátku této podkapitoly bude vysvětlen pojem jednosměrná funkce, který následně bude využit pro definování pojmu hašovací funkce. Na konci podkapitoly bude vysvětlena důležitá vlastnost hašovacích funkcí: bezkoliznost.

1.1.1 Jednosměrná funkce

Definice 1.1 (Jednosměrná funkce, podle [1])**.** *Funkce $f : X \mapsto Y$, pro níž je snadné z jakékoli hodnoty $x \in X$ vypočítat $y = f(x)$, ale pro nějaký náhodně vybraný obraz $y \in f(X)$ nelze (je to pro nás výpočetně nemožné) najít její vzor $x \in X$ tak, aby $y = f(x)$.*

Je nutno upřesnit, že slovem "snadné" je myšleno to, že pro výpočet obrazu $f(x)$ ze vzoru x existuje algoritmus pracující v polynomiálním čase. Naopak "výpočetně nemožné" znamená, že pro výpočet vzoru x z obrazu $f(x)$ není obecně znám algoritmus pracující v polynomiálním čase x, a nelze ho tedy pro velká $f(x)$ provést v rozumném čase.

Jednoduchým příkladem jednosměrné funkce je součin $s = pq$ dvou prvočísel (viz 1.1.1). Samotný součin lze spočítat snadno, avšak pro rozklad součinu s na prvočinitele p a q není obecně znám žádný algoritmus mající polynomiální složitost.



Obrázek 1.1: Jednosměrná funkce

1.1.2 Hašovací funkce

Pojem hašovací funkce původně označoval funkci, která pro libovolně dlouhý binární vstup vrátila binární výstup o pevně dané délce. Tomuto výstupu se říká haš či hašový kód. Dnes se pojem hašovací funkce používá pro označení kryptografické hašovací funkce, která navíc splňuje vlastnosti jednosměrnosti a bezkoliznosti[1].

Definice 1.2 (Hašovací funkce, podle [1])**.** *Mějme přirozené číslo d a množinu X všech binárních řetězců délky 0 až d (prázdný řetězec je platným vstupem a má délku 0). Funkci $h\colon X \mapsto \{0,1\}^n$, kde n je přirozené číslo, nazveme hašovací, jestliže je jednosměrná a bezkolizní. Říkáme, že každému binárnímu řetězci z množiny X přiřadí binární hašovací kód (haš, hash) délky n bitů.*

Podle [1] z hlediska bezpečnosti požadujeme, aby se hašovací funkce chovala jako **náhodné orákulum**:

[1] Pojem bezkoliznost bude vysvětlena v následující sekci.

- **Orákulum** nazýváme stroj, který na základě vstupu odpovídá nějakým výstupem a má tu vlastnost, že na tentýž vstup odpovídá stejným výstupem.
- **Náhodné orákulum** je orákulum, které má navíc tu vlastnost, že na nový vstup odpovídá náhodným výběrem výstupu z množiny výstupů.

Důsledkem chování hašovací funkce je to, že hašový kód je číslo náhodně vybrané z množiny výstupů. Kód v (1) ukazuje příklad hašového kódu vytvořeného na příkazovém řádku pomocí příkazu `sha256sum`. Haš vstupního textu závisí mimo jiné na používaném hašovacím algoritmu. Hašovací algoritmy se liší podle způsobu hašování a velikosti haše. V tomto příkladě byl použit algoritmus SHA256, který vytváří 256 bitové haše. Zároveň jakákoliv změna na vstupním textu změní kompletně i výsledný haš (viz 2).

```
$ echo -n foobar | sha256sum
c3ab8ff13720e8ad9047dd39466b3c8974e592c2fa383d4a3960714caef0c4f2
```

Listing 1: Haš vytvořený pomocí algoritmus SHA256

```
$ echo -n foobas | sha256sum
808e8296cfa36d6ae9956c06e70bad3a872d6c63e83fe19cda5186f96703f854
```

Listing 2: Změna haše při změně vstupního textu

1.1.3 Bezkoliznost hašovacích funkcí

Z definice hašovací funkce (1.1.2) může být vstupní množina větší než výstupní množina. Z toho plyne, že mohou existovat dva různé vstupy, které se zobrazí na stejný výstup. Taková situace se nazývá kolize a je pro nás, jak bude ukázáno v sekci o kryptografických klíčích a digitálních podpisech, z hlediska bezpečnosti nežádoucí. Cílem však není zajistit nemožnost nalezení kolize, nýbrž snížit pravděpodobnost jejího výskytu natolik, aby bylo výpočetně nemožné ji nalézt. Taková hašovací funkce se pak nazývá bezkolizní.

V sekci (1.1.2) bylo zároveň zmíněno, že z hlediska bezpečnosti by se hašovací funkce měla chovat jako náhodné orákulum. Uvažujeme-li příklad takové hašovací funkce $f\colon \{0,1\}^{0\ldots d} \mapsto \{0,1\}^n$, kde d a n jsou přirozená čísla a n je dostatečně velké (např. 256 bitů), pak je složitost nalezení kolize 2^n. Taková složitost není polynomiální, a nalezení kolize je tedy výpočetně neproveditelné.

Rozlišujeme bezkoliznost 1. řádu a 2. řádu:

Definice 1.3 (Bezkoliznost 1. řádu, podle [1])**.** *Hašovací funkce h je odolná proti kolizi 1. řádu, jestliže je výpočetně nezvládnutelné nalezení libovolných dvou různých zpráv M a M' tak, že $h(M) = h(M')$. Pokud se toto stane, říkáme, že jsme nalezli kolizi.*

Je nutno upozornit, že pokud se hašovací funkce $f\colon \{0,1\}^{0\dots d} \mapsto \{0,1\}^n$ bude chovat jako náhodné orákulum, složitost nalezení kolize 1. řádu s 50% pravděpodobností je $\approx 2^{\frac{n}{2}}$, narozdíl od očekávané složitosti 2^{n-1} . Tato složitost vyplývá z narozeninového paradoxu [2]. I přes tento fakt je nalezení kolize výpočetně neproveditelné.

Definice 1.4 (Bezkoliznost 2. řádu, podle [1])**.** *Hašovací funkce h je odolná proti kolizi 2. řádu, jestliže pro jakýkoliv vzor x je výpočetně nezvládnutelné nalézt 2. vzor $y \neq x$ tak, že $h(x) = h(y)$.*

Pokud se hašovací funkce $f\colon \{0,1\}^{0\dots d} \mapsto \{0,1\}^n$ bude chovat jako náhodné orákulum, pak složitost nalezení 2. vzoru je $\approx 2^n$. Tento fakt opět plyne ze skutečnosti, že velikost výstupní množiny je 2^n.

1.2 Kryptografický klíč

Tato podkapitola se věnuje pojmu kryptografický klíč a způsobu jeho generování. Jeho využití bude popsáno v sekci o digitálních podpisech (1.3) a v sekci o kryptoměnových adresách (2.3).

1.2.1 Soukromý a veřejný klíč

Fungování kryptoměn je založeno na asymetrické kryptografii, která pro šifrování a vytváření digitálních podpisů (1.3) využívá dva různé a mezi sebou matematicky svázané kryptografické klíče: soukromý klíč a veřejný klíč.

Soukromý klíč je náhodně vygenerované číslo obvykle mezi 1 a přirozeným číslem n. Z bezpečnostního hlediska je po generátoru náhodných čísel vyžadováno, aby n bylo dostatečně velké (např. 256 bitů). Díky tomu je nově vytvořený soukromý klíč prakticky vždy unikátní (viz sekce o bezkoliznosti 1.1.3). Soukromý klíč by měl být uchován v tajnosti. Kód 3 ukazuje příklad, jak může vypadat vygenerovaný soukromý klíč.

```
$ openssl rand -hex 32
47d1c82eb6eb83f583defcc66f7fb76b20a28aa37f8f8269d3e6d71dc820c541
```

Listing 3: 256 bitový soukromý klíč (v hexadecimální soustavě, 32 bajtů, 64 hexadecimálních číslic) vygenerovaný pomocí kryptografické knihovny OpenSSL.

Ze soukromého klíče lze pomocí matematického algoritmu odvodit klíč veřejný. Veřejný klíč může být na rozdíl od soukromého klíče veřejně dostupný. Existuje více algoritmů pro odvození veřejného klíče jako např. RSA[2] a ECC (z angl. překladu Elliptic Curve Cryptography). Pro účely dalšího výkladu bude popsán algoritmus odvození pomocí ECC, neboli kryptografie nad eliptickými křivkami.

[2] Z inicálů autorů Rivest, Shamir a Adleman.

1.2.2 Kryptografie nad eliptickými křivkami

Eliptická křivka E je graf na reálných číslech vyhovující rovnici:

$$y^2 = x^3 + ax + b \tag{1.1}$$

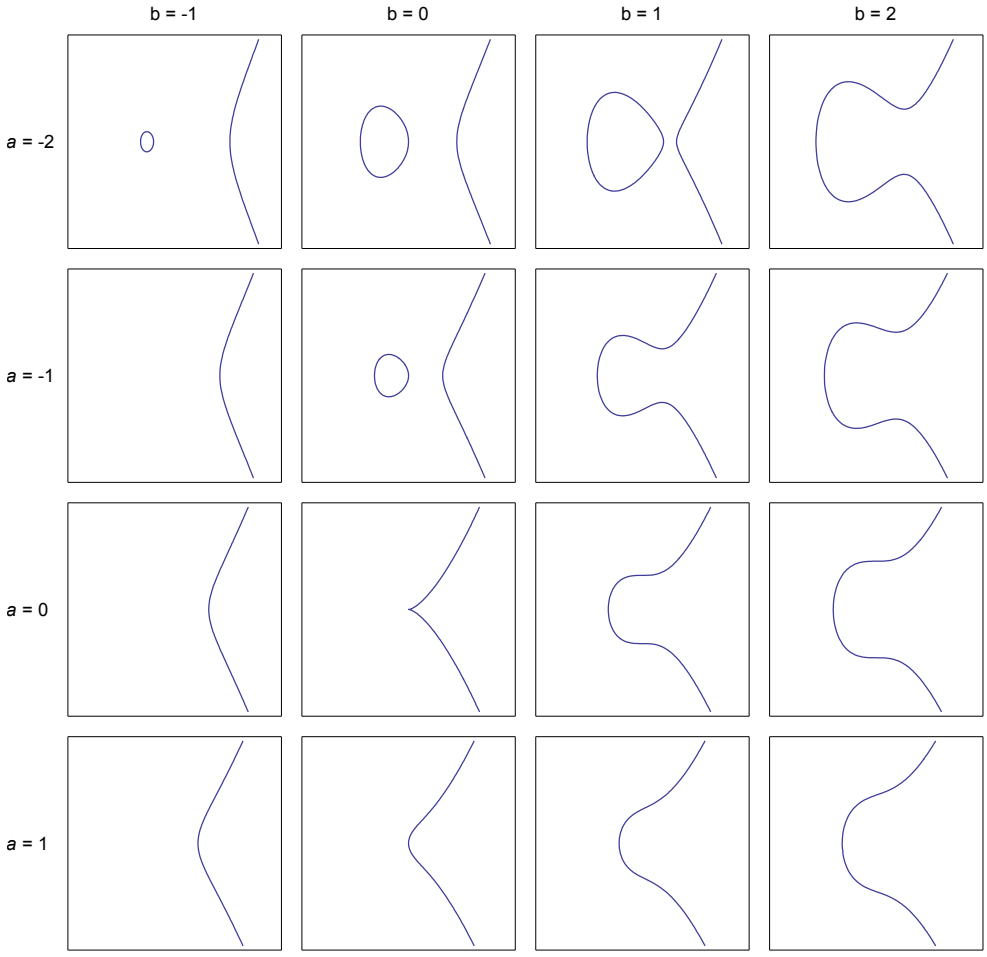
kde a, b jsou reálná čísla a $4a^3 + 27b^2 \neq 0$. Podmínka $4a^3 + 27b^2 \neq 0$ zaručuje, že daná křivka bude nesingulární, a tedy diferencovatelná na každém bodu z jejího definičního oboru [3, s. 8]. Singulární eliptická křivka nemá pro operaci sčítání bodů[3] na eliptické křivce rozumné chování. Obrázek (1.2) ilustruje příklady eliptických křivek pro různé parametry a, b včetně singulární křivky s parametry $a = b = 0$.

V kryptografii je používána speciální varianta eliptické křivky, která je navíc definovaná na konečném tělese: kryptografická eliptická křivka. Kryptografická eliptická křivka E nad tělesem F_p prvočíselného řádu $p > 3$, je graf vyhovující rovnici:

$$y^2 = x^3 + ax + b \bmod p \tag{1.2}$$

kde a, b jsou prvky tělesa F_p, x je libovolné reálné číslo a $4a^3 + 27b^2 \neq 0$. Kryptografická eliptická křivka ve skutečnosti není křivka, nýbrž množina diskrétních bodů, jak je ilustrováno na obrázku (1.3). Pro následující výklad bude dále uvažována pouze kryptografická eliptická křivka.

[3] Operace sčítání bodů na eliptické křivce bude vysvětlena později.



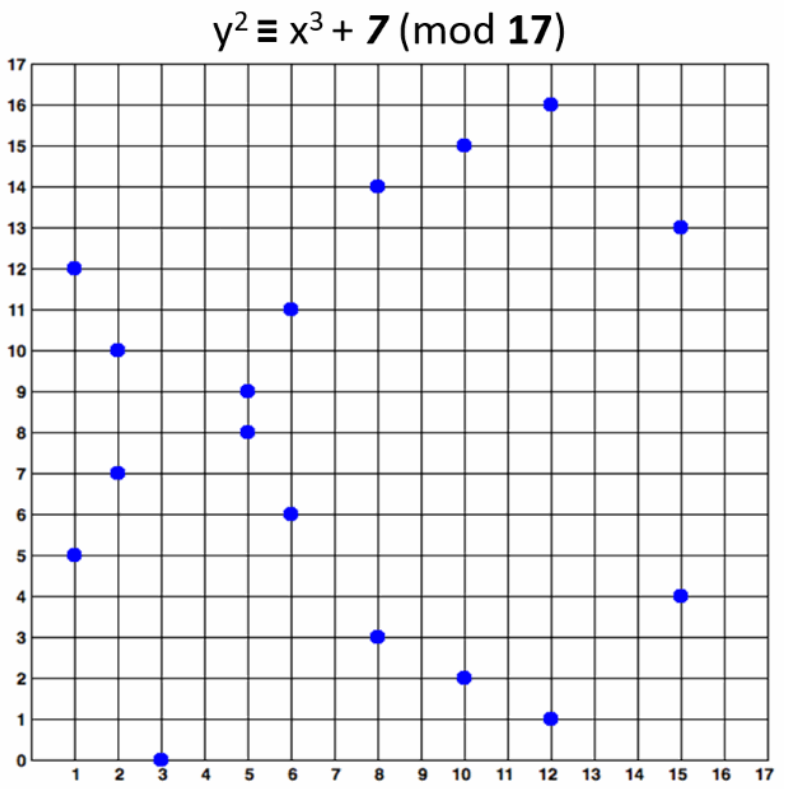
Obrázek 1.2: Příklady eliptických křivek, podle [4]

Křivka pro $a = b = 0$ není nesingulární, a není tedy eliptickou křivkou.

Pro body na eliptické křivce je definována operace sčítání. Podle [6] vypadá geometrická interpretace součtu dvou bodů $P = [x_p, y_p]$ a $Q = [x_q, y_q]$ na křivce E tak, že spojíme body P a Q přímkou. Tato přímka protne křivku E v bodě $-R$. Výsledkem sčítání je potom bod R, který je symetrický k $-R$ podle osy x. Body symetrické podle osy x nazýváme opačné.

[6] zároveň uvádí způsob, jak spočítat souřadnice bodu R. Využívá se k tomu směrnice přímky spojující body P a Q. Pokud jsou tyto body navzájem různé, pak má daná směrnice tvar:

$$s = \left| \frac{y_Q - y_P}{x_Q - x_P} \right|_p \tag{1.3}$$



Obrázek 1.3: Kryptografická eliptická křivka na tělese řádu 17, podle [5].

V opačném případě má směrnice tvar:

$$s = \left| \frac{3x_P^2 + a}{2y_P} \right|_p \tag{1.4}$$

Souřadnice bodu $R = [x_R, y_R]$ jsou poté spočteny následovně:

$$x_R = \left| s^2 - x_P - x_Q \right|_p \tag{1.5}$$

$$y_R = \left| s(x_P - x_R) - y_P \right|_p \tag{1.6}$$

Přímka procházející dvěma opačnými body P a $-P$ křivku E již neprotne, a proto je nutné dodefinovat:

$$P + (-P) = O \tag{1.7}$$

$$P + O = P \tag{1.8}$$

$$O + O = O \tag{1.9}$$

$$O = -O \tag{1.10}$$

Bod O se nazývá nulový bod. Eliptická křivka E spolu s operací sčítání bodů $+$ a nulovým bodem O pak tvoří konečnou cyklickou grupu $(E, +, O)$.

Násobení bodu P přirozeným číslem k je pak definován jako:

$$P \times k = \underbrace{P + P \cdots + P}_{k\text{-krát}} \tag{1.11}$$

[6] dále definuje pojmy řád bodu, řád křivky a kofaktor. Řád bodu P na křivce E je nejmenší r takové, pro které je $P \times r = 0$. Jinými slovy, po určitém r násobku bodu P se musíme zase vrátit k nulovému bodu P. Existence r je dána faktem, že E je definován na konečném tělese. Řádem křivky je počet bodů na E a kofaktor je podíl: řád křivky / řád generátoru dané grupy.

1.2.3 Generování veřejného klíče

Zatímco soukromý klíč je přirozené číslo, veřejný klíč je bod na eliptické křivce E. Pro generátor P na eliptické křivce E a soukromý klíč k lze veřejný klíč Q vygenerovat následujícím způsobem:

$$P \times k = Q \tag{1.12}$$

Zatímco spočítat $P \times k$ je jednoduchá úloha, spočtení k na základě znalosti P a Q je výpočetně neproveditelné. Problém nalezení k na základě P a Q se nazývá problém diskrétního logaritmu nad eliptickou křivkou.

1.2.4 Doménové parametry

Různé eliptické křivky poskytují různou úroveň bezpečnosti, rychlost výpočtů a velikost klíče [7]. Liší se vlastnostmi, které byly uvedeny v této podkapitole: řád tělesa, řád bodu, řád křivky, konstanty funkce křivky, generátor a kofaktor. Existují kryptografické standardy, jako např [8], které definují různé parametry pro eliptické křivky a které byly prozkoumány a analyzovány kryptografy, a jsou tudíž považovány za bezpečné.

1.3 Digitální podpis

Digitální podpis je označení dat, které slouží jako důkaz vlastnictví soukromého klíče, a to bez toho, aniž by musel být zveřejněn. Jedním ze způsobů, jak takový podpis vytvořit, je podepisovací schéma ECDSA (z angl. Elliptic Curve Digital Signature Algorithm), které popisuje princip vytváření digitálních podpisů za použití kryptografie nad eliptickými křivkami.

Podle [9, s. 37] vypadá schéma ECDSA následovně:

1. **(sk, vk) := vygenerujKlíče(*velikostKlíče*)** Metoda na základě velikosti klíče vygeneruje dvojici klíčů *sk* a *vk*. Soukromý klíč *sk* je uchováván v tajnosti a slouží k podepisování zprávy, neboli k vytvoření jejího podpisu. Veřejný klíč *vk* je naopak všeobecně dostupný a slouží k ověření podpisu vytvořeného odpovídajícím soukromým klíčem *sk*.

2. **sig := podepiš(*sk*, *zpráva*)** Metoda vytvoří podpis dané zprávy za pomocí soukromého klíče. Říkáme, že daná zpráva je podepisována soukromým klíčem.

3. **jeValidní := ověř(*vk*, *zpráva*, *podpis*)** Metoda ověří, že daný podpis byl vytvořen podepsáním dané zprávy soukromým klíčem příslušným k danému veřejnému klíči *vk*. V kladném případě říkáme, že podpis je validní. V opačném případě je nevalidní.

Podle [10] musí digitální podpis splňovat následující vlastnosti:

- **Nezfalšovatelnost** - podpis se nedá napodobit jiným subjektem než podepisujícím.
- **Ověřitelnost** - příjemce dokumentu musí být schopen ověřit, že podpis je platný.
- **Integrita** - podepsaná zpráva se nedá změnit, aniž by se zneplatnil podpis.
- **Nepopiratelnost** - podepisující nesmí mít později možnost popřít, že dokument podepsal.

[5] detailně popisuje algoritmus ECDSA. Algoritmus na základě zprávy a soukromého klíče vytvoří dvojici hodnot (r, s), která reprezentuje daný podpis. Z r a s přitom nelze díky problému diskrétního logaritmu nad eliptickou křivkou zjistit soukromý klíč sk. Nicméně lze za pomocí veřejného klíče vk snadno ověřit, že r a s byly vytvořeny příslušným soukromým klíčem sk. Jelikož soukromý klíč je tajný, lze tímto způsobem ověřit autentičnost podpisu.

Kapitola 2

Kryptoměna

Před analyzováním struktury peněženky kryptoměny Bitcoin je potřeba vysvětlit, co to kryptoměna vůbec je, které problémy řeší a co vedlo k jejímu vzniku. Na začátku kapitoly bude proto stručně uvedena motivace ke vzniku kryptoměny. Následně bude definován pojem kryptoměna. Dále budou vysvětleny pojmy v kryptoměnách: adresa, transakce a distribuovaný konsensus.

2.1 Motivace

V tradičním platebním systému (TPS) jsme omezeni tím, že lze platbu provést pouze za fyzické přítomnosti účastníků. Takový systém může být problémový, pokud člověk potřebuje provést platbu do zahraničí. Alternativou TPS je proto elektronický platební systém (EPS), pomocí kterého lze peníze převádět elektronicky prostřednictvím internetové sítě. Díky tomu lze s přístupem k internetu platby provádět kdykoliv a odkudkoliv.

Jelikož EPS využívá k převodu peněz internetovou síť, je přirozeně potřeba zajistit vysoké zabezpečení proti krádežím a podvodům. Jeden z nejznámějších problémů EPS je problém dvojího utrácení, kdy jsou z dat reprezentujících elektronické (také digitální) peníze vytvořeny kopie a ty jsou následně zaslány různým osobám. Pokus o vyřešení tohoto problému lze spatřit již ve 20. století, kdy David Chaum založil digitální měnu Ecash.[11, str. 8] Problém dvojího utrácení se mu podařilo vyřešit, nedokázal však zcela eliminovat další problém, kterým je anonymita uživatele.

Ukázalo se, že jedním ze způsobů, jak vytvořit anonymní systém, je z něho zcela odstranit roli centrální autority a pro dosáhnutí distribuovaného konsensu[4] použít technologii blockchain[5] jakožto globální účetní knihu pro všechny provedené transakce. Tento přístup lze spatřit např. v publikacích b-money a BitGold [12, str. 17], na základě kterých Satoshi Nakamoto v roce

[4]Pojem distribuovaný konsensus bude vysvětlen později

[5]Pojem blockchain bude vysvětlen detailněji později.

2008 publikoval dokument *Bitcoin: A Peer-to-Peer Electronic Cash System.* [12] Na základě této publikace v roce 2009 vytvořil Satoshi první implementaci kryptoměny Bitcoin.

2.2 Obecná definice kryptoměny

Definice 2.1 (Kryptoměna, podle [13])**.** *Kryptoměna je platební systém, který splňuje následující vlastnosti:*

1. *Systém nepotřebuje centrální autoritu, společného konsensu dosáhne distribuovaně.*
2. *Systém udržuje přehled o mincích a jejich vlastnictví.*
3. *Systém definuje, zda mohou být nové mince vytvořeny. Pokud ano, systém definuje podmínky jejich vzniku a způsob, jak rozhodnout o jejich vlastnictví.*
4. *Vlastnictví mincí může být dokázáno výhradně kryptograficky.*
5. *Systém umožňuje provádět transakce, které mění vlastnictví mincí. Provádění transakcí může být uskutečněno pouze entitou, která prokáže vlastnictví těchto jednotek v danou chvíli.*
6. *Pokud jsou provedeny dvě instrukce měnící vlastnictví týchž mincí, vykoná se nejvýše jedna z nich.*

2.3 Adresa

Uživatel kryptoměny je identifikován svoji kryptoměnovou adresou, která slouží k přijímání peněz. Tato adresa by měla splňovat některé důležité vlastnosti: unikátnost a nemožnost z ní odvodit skutečnou identitu uživatele. Obě tyto vlastnosti zvyšují anonymitu uživatele a jsou zajištěny způsobem, kterým je adresa generována. Adresa je obvykle odvozena z veřejného klíče. Může to bých např. haš veřejného klíče. Adresa je kvůli své délce a pro lepší čitelnost obvykle převedena do kratšího a čitelnějšího formátu, např. do hexadecimální soustavy.

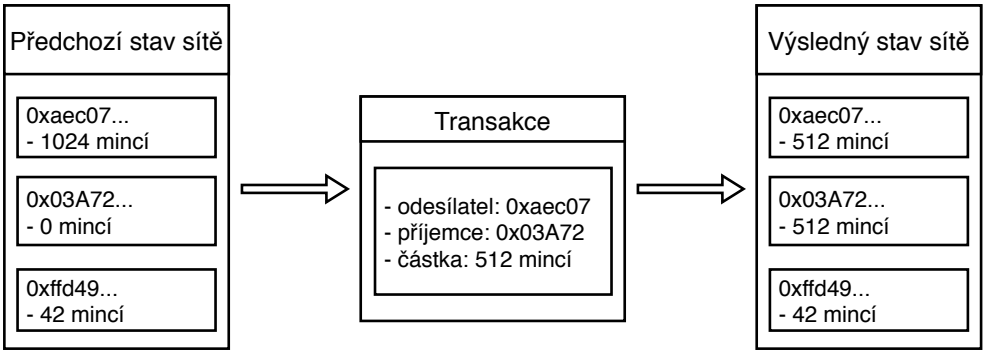
2.4 Transakce

Transakci popisuje [11] jako: *„Transakce je datová struktura, která reprezentuje přenos hodnoty mezi účastníky kryptoměnového systému“*. Transakce v kontextu kryptoměn tedy vyjadřuje platbu mezi dvěma uživateli.

Struktura transakce se liší podle toho, který model pro reprezentaci stavu adres je používán. Rozlišujeme účetní model a model UTXO.

2.4.1 Účetní model

Podle [14] je v tomto modelu stav sítě reprezentován množinou účtů. Každý účet je přitom reprezentován svojí adresou a má zůstatek. Při provedení transakce jsou mince z účtu odesílatele odečteny a k účtu příjemce přičteny. Aby byla transakce validní, musí být na účtu odesílatele dostatek mincí. Obrázek (2.1) ilustruje takovou transakci a související změny stavu sítě.



Obrázek 2.1: Transakce v účetním modelu

Po provedení dané transakce je od adresy `0xaec07...` odečteno 512 mincí a k adrese `0x03A72...` 512 mincí přičteno.

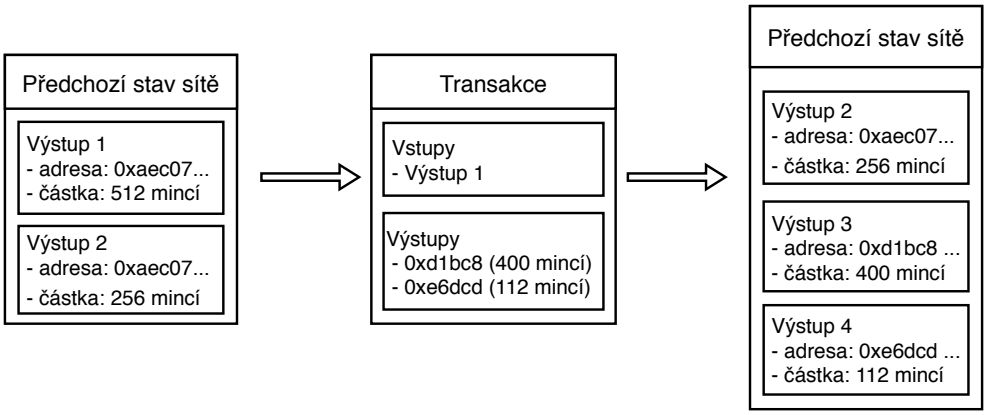
2.4.2 Model UTXO

V tomto modelu není u adresy uživatele evidován zůstatek, nýbrž množina UTXO[6]. UTXO reprezentuje částku peněz a adresu, která tuto částku aktuálně vlastní. Struktura transakce pak vypadá tak, že má seznam vstupů a seznam výstupů. Vstupem transakce je odkaz na UTXO, tedy výstup, který zatím nebyl využit v žádné jiné transakci jako vstup. Jakmile je UTXO v nějaké transakci utracen, je odstraněn z množiny UTXO, a nemůže již být utracen. Výstupem transakce je seznam nových UTXO, které mohou být novými majiteli využity v dalších transakcích. Zůstatek na adrese je pak spočítán jako součet UTXO, které patří této adrese. Tímto způsobem lze převádět finanční prostředky mezi adresami, jak je ilustrováno na obrázku (2.2).

2.4.3 Autentičnost transakce

Aby bylo možné čerpat finanční prostředky, ať už ve formě výstupů či zůstatků na účtu, je nutné dokázat jejich vlastnictví, tedy vlastnictví příslušné adresy a soukromého klíče, aby bylo zabráněno použití finančních prostředků

[6]Nevyužitý transakční výstup (zkratka z angl. *unspent transaction output*).



Obrázek 2.2: Transakce v modelu UTXO

V předchozím stavu sítě jsou dva výstupy příslušící adrese `0xae...` a po provedení dané transakce je utracen `Výstup 1` a vytvořen `Výstup 3` a `Výstup 4`.

neoprávněou osobou. K tomuto účelu slouží digitální podpisy, které byly popsány v 1.3. V případě modelu UTXO obsahují výstupy tzv. zamykací skript, který definuje podmínky, za kterých lze daný výstup utratit. Obvykle je danou podmínkou důkaz vlastnictví adresy. Vstupy pak obsahují odpovídající odemykací skripty, které pokud splňují podmínky v zamykacím skriptu, mohou dané výstupy utrácet. Tyto skripty však mohou mít i složitější formu. Více informací o této problematice lze najít na [11].

2.5 Distribuovaný konsensus

Kryptoměny fungují na bázi peer-to-peer sítě, kde centrální autorita nehraje žádnou roli. Všichni uživatelé (také uzel či node) si jsou rovni a komunikace mezi nimi probíhá přímo. O aktuálním stavu sítě je rozhodováno hromadně prostřednictvím blockchainu: spojový seznam bloků dat, kde ukazatelem
v každém bloku (s výjimkou prvního) je hašový kód vytvořený z dat předchozího bloku. Díky tomuto zřetězení je zajištěna integrita dat, neboť nelze změnit data v bloku, aniž by tím byl zneplatněn jeho haš. Kryptoměny využívají blockchain jako globální účetní knihu evidující veškeré provedené a sítí akceptované transakce. Blockchain je veřejný, a každý node může tudíž vlastnit jeho kopii. Aby bylo zabráněno dvojímu utrácení, jsou používány různé metody. Např. v případě Bitcoinu je nutné k bloku transakcí přidat tzv. proof-of-work, aby mohl být sítí považován za validní. K získání proof-of-work je potřeba vyřešit „těžký“ matematický problém jakožto důkaz vynaloženého

výpočetního výkonu. Čím má zařízení větší výkon, tím je větší šance, že vyřeší daný problém, a získá tak proof-of-work daného bloku. K uskutečnění dvojího utrácení by musel útočník vynaložit obrovské množství výpočetního výkonu, kterého je prakticky nemožné dosáhnout, pokud se většina uživatelů sítě chová „slušně“. Uzlu, který se snaží vyřešit daný problém se říká těžař a procesu nalezení řešení problému se říká těžení. Více informací o problematice distribuovaného konsensu, blockchainu a těžení lze najít na [11].

Kapitola 3

Bitcoin

Tato kapitola analyzuje strukturu peněženky kryptoměny Bitcoin. Na začátku bude Bitcoin popsán obecně. Následně bude vysvětleno, co je to peněženka. Pak bude analyzována struktura peněženky konkrétní implementace Bitcoin-Core.

3.1 Obecně o Bitcoinu

Bitcoin je peer-to-peer platební síť a zároveň v této síti používaná kryptoměna, kterou v roce 2008 popsal Satoshi Nakamoto ve svém dokumentu *„Bitcoin: A Peer-to-Peer Electronic Cash System“* [12]. Následně v roce 2009 vydal Satoshi první implementaci Bitcoinu s názvem Bitcoin-Qt. Tato implementace byla od verze 0.5 zintegorvána s implementací bitcoind, která umožňovala konat instrukce přes textové uživatelské rozhraní. Výsledný program byl nazván Bitcoin-Core, který je používán dodnes a slouží jako referenční implementace pro další klienty. Pro popis struktury peněženky bude proto předpokládána tato konkrétní implementace.

3.2 Struktura peněženky

Peněženka je struktura, která spravuje adresy a soukromé klíče uživatele a umožňuje vytvářet transakce.

Klíče spravované peněženkou jsou uložené ve složce *wallets*, která má následující strukturu (podle [15]):

```
/wallets
├── wallet.dat
├── database
├── db.log
└── .walletlock
```

Popis jednotlivých souborů (podle [15] a [16]):

- **wallet.dat** - wallet.dat slouží jako úložiště klíčů a souvisejících transakcí. Úložiště je implementováno jako databáze Berkley DB.
- **database** - database je složka obsahující logovací soubory pro udržení databáze wallet.dat v konzistentní stavu v případě neočekávaných havárií jako např. náhlý pád systému.
- **db.log** - db.log slouží pro logování chybových zpráv z operací prováděné na databázi wallet.dat.
- **.walletlock** - .walletlock slouží k zamykání složky. Slouží pro synchronizaci přístupů ke klíčům v případě, že jsou spuštěny a používány 2 a více instancí bitcoinového klienta zároveň. Soubor je skrytý.

Nejdůležitějším souborem je *wallet.dat*, který obsahuje veškeré soukromé klíče a adresy spravované peněženkou. Ztráta či odcizení tohoto souboru znamená ztrátu všech klíčů v něm obsažených. Z tohoto důvodu je doporučeno tento soubor pravidelně zálohovat a uchovávat v bezpečí. V souboru jsou dále uloženy transakce, které obsahují UTXO náležící daným adresám. Platby jsou pak rychlejší, neboť při vytváření transakce není potřeba projít celý blockchain, aby byly nalezeny všechny UTXO uživatele. Soubor je implementován jako databáze Berkeley DB verze 4.8.30. Přestože již existují novější verze (v době psaní verze 18.1.), Bitcoin stále zůstává u starší verze kvůli zpětné kompatibilitě [17]. [15] se dále zmiňuje o tom, že složka *database* by měla uchována stejně v bezpečí jako *wallet.dat*. Toto je pochopitelné z důvodu uvedeného v popisu. Tato složka je vytvořena při spuštění klienta a po vypnutí smazána.

Dále je nutno zmínit, že bitcoin umožňuje mít více peněženek zároveň. Každá peněženka pak může sloužit pro jiný účel. V takovém případě vypadá struktura peněženky tak, že pro každou nově vytvořenou peněženku se do složky *wallets* přidá další složka reprezentující další peněženku. Pokud tedy kromě defaultní peněženky vytvoříme 2 další peněženky s názvy *walletA* a *walletB*, vypadá struktura složky *wallets* následovně:

```
/wallets
├── walletA
├── walletB
├── wallet.dat
├── database
├── db.log
└── .walletlock
```

Peněženky *walletA* a *walletB* maji pak podobnou strukturu jako defaultní peněženka a veškeré soubory v nich se vztahují jenom k těmto peněženkám:

```
/walletA
├── wallet.dat
├── database
├── db.log
└── .walletlock

/walletB
├── wallet.dat
├── database
├── db.log
└── .walletlock
```

Kapitola 4

Databáze dostupné pro UNIXové systémy

V této kapitole budou stanoveny požadavky na strukturu peněženky. Následně bude provedena rešerše „malých“ databází dostupných pro UNIXové systémy. Na konci bude diskuse o tom, které z těchto se hodí pro implementaci struktury s ohledem na dané požadavky.

4.1 Požadavky

Cílem je navrhnout strukturu peněženky, která bude únosná i při větším počtu klíčů (řádově miliónů). Aktuální struktura peněženky kryptoměny FITCOIN aktuálně sestává jen ze souborů v jednom adresáři. Obsahem souboru je soukromý klíč a jménem souboru je příslušný veřejný klíč. Při miliónech klíčů by adresář obsahoval milióny souborů. Taková struktura může být problematická. Např. některé souborové systémy stále používají pomalejší algoritmus pro vyhledávání souborů, a tudíž např. u ext2 se projevují výkonnostní problémy již při 10 000 souborech v jednom adresáři. [18].

Z pohledu uživatele může být taková struktura problematická, pokud chce na linuxu zobrazit obsah adresáře pomocí příkazu *ls*. Příkaz defaultně zobrazí soubory v adresáři v seřazeném pořadí, což může podle [19] trvat značně dlouho. V [19] jsou poskytována různá řešení, pro uživatele je to však zbytečná práce navíc.

Další nevýhodou je zálohování. Zkopírovat milión souborů je značně pomalejší než zkopírovat jeden soubor o stejné velikosti. Pak je otázka, jak zkontrolovat, že se úspěšně zkopírovaly všechny soubory, nebo že žádný z nich nechybí.

Požadavky na strukturu peněženky tedy jsou:

1. Rychlé otevírání a vyhledávání klíčů.

2. Možnost ukládat větší počet klíčů, v řádech miliónů.

3. Příjemnější práce z pohledu uživatele (např. příkaz *ls*)

4. Snadnější zálohování.

V této práci byly analyzovány databáze Berkeley DB a SQLite.

4.2 Berkley DB

Berkley DB (dále jen BDB) je open-source knihovna a vestavěná databáze typu klíč-hodnota. Databáze běží ve stejném adresovém prostoru jako samotná aplikace, a není tedy potřeba zařizovat žádný databázový server. Díky tomu je komunikace mezi aplikací a databází velmi rychlá. První dojem z použití této knihovny byl takový, že rozhraní bylo zpočátku trochu neintuitivní. Knihovna nepodporuje SQL dotazy, místo toho pro operace nad databází poskytuje svoje vlastní API, které je jednoduché ale nestandardní. Programátor musí např. při vkládání či čtení specifikovat adresu proměnné a datový typ. Na druhou stranu uživatel může do databáze vkládat data jakéhokoliv datového typu (i strukturu). Tímto je možné emulovat databázový záznam s dvěma a více sloupci. Navíc díky absenci SQL dotazů není potřeba parsovat a vykonávat dotazy, čímž jsou databázové operace rychlejší. BDB je velmi konfigurovatelný, poskytuje např. možnost nastavit datovou strukturu pro ukládání dat (BTree, Hash). Ke knihovně existuje výborná dokumentace, která poskytuje i konkrétní příklady, jak ji používat. BDB umožňuje ukládat více než 4 miliardy klíčů-hodnot a celkově až 256 terabytů dat [20]. Je portabilní na téměř všech unixových systémech. Podporuje programovací jazyk C. Nemá žádné další povinné závislosti.

4.3 SQLite

SQLite je knihovna a vestavěná relační databáze umožňující provádět operace pomocí SQL dotazů. Stejně jako BDB lze SQLite zprovoznit i bez serveru. Rozhraní knihovny je velmi intuitivní a kód používající toto rozhraní je pro někoho, kdo umí pracovat s SQL, velmi čitelný. Knihovna umožňuje ukládat následující datové typy: TEXT, NUMERIC, INTEGER, REAL a BLOB. Pro spuštění není potřeba nic konfigurovat. Vyhledavání záznamu má časovou složitost $log(n)$ vzhledem k počtu záznamů. Knihovna nemá žádné další závislosti navíc. Má na veřejné doméně SQLite.org velmi dobrou dokumentaci. Operace nad SQLite databází lze provádět i na příkazovém řádku. Maximální velikost databáze je 130 terabytů a maximální počet záznamů je 2^{64} [21]. Tohoto maxima nelze dosáhnout, předtím narazíme na hranic 130 terabytů. Častěji jsme však omezeni kapacitou diskového úložiště.

4.4 Diskuse

Jak BDB tak i SQLite splňují uvedené požadavky na strukturu peněženky. Obě dokáží vyhledat záznam v databázi v logaritmickém čase vzhledem k celkovému počtu záznamů, což je i pro větší počet klíčů akceptovatelná složitost. Umožňují ukládat milióny záznamů. Databáze tvoří jeden soubor, a tím je snadno zálohovatelná. Pro implementaci však byl zvolen SQLite. První důvod je ten, že je snadno dostupný, neboť lze nainstalovat potřebné programy prostřednictvím správce balíčků. Dále není na začátku potřeba nic konfigurovat. Zároveň lze s databází manipulovat pomocí SQL dotazů, se znalostí jazyka SQL je tudíž implementace čitelnější a jednodušší. Knihovna také umožňuje manipulovat s databází pomocí SQL na příkazovém řádku. V implementaci se dále velmi hodí SQL konstrukce `OUTER JOIN`, která slouží ke spojení výsledků dotazu `SELECT`. Lze ji využít např. v následující situaci. Předpokládejme tabulku zůstatků `Amounts` a tabulku klíčů `Keys`:

Address	Amount
A	1024
X	700
C	512
Z	0

Address	Key
A	a
B	b
C	c
D	d

Pro získání všech adres, které jsou v tabulce klíčů, a k nim příslušné zůstatky v databázi zůstatků, lze vykonat následující dotaz: `SELECT x.Address, IFNULL(y.Amount, 0) FROM keys x LEFT OUTER JOIN address y ON x.Address = y.Address;"`. Výsledkem dotazu je následující tabulka:

Address	Amount
A	1024
B	0
C	512
D	0

Kapitola 5

FITCOIN

Projekt FITCOIN má za cíl analyzovat nezbytné základy pro fungování kryptoměny a na základě nich vytvořit co možná nejjednodušší funkční kryptoměnu. Tato kryptoměna nese stejný název jako samotný projekt, tedy FICOIN. Účelem je pochopit vnitřní fungování kryptoměn. V projektu je kladen důraz na přenositelnost kódu. Z tohoto důvodu je v implementaci používáno co nejméně externích knihoven. Implementace je psána v jazyce C.

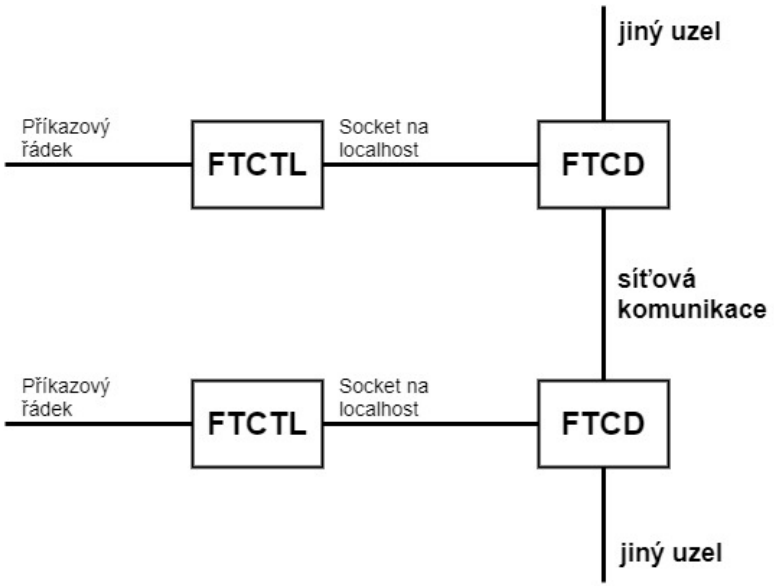
Tato práce přispívá do projektu návrhem a implementací nové struktury peněženky. Pro úplnost bude obecně popsána kryptoměna FITCOIN. V první sekci této kapitoly budou popsány jednotlivé programy, z kterých je FITCOIN složen. Následně budou uvedeny využité kryptografické metody. Dále budou vysvětleny základní datové struktury. V posledních dvou sekcích bude popsána adresářové struktura projektu a navržená struktura peněženky.

5.1 Architektura

FITCOIN je složen ze dvou základních programů: `ftcd` a `ftctl`. Program `ftcd` (Fitcoin Daemon) tvoří páteř celého projektu a reprezentuje síťový uzel v peer-to-peer síti. Umožňuje komunikovat s ostatními uzly v dané síti a vyměňovat si s nimi informace o aktuální podobě blockchainu a nově vytvořených transakcích. Dále dokáže spustit těžení bloků a využívat funkce peněženky. Program po spuštění běží jako démon, který běží neustále na pozadí počítačů uživatelů. Program nemá žádné uživatelské rozhraní a požadavky uživatele přijímá pomocí síťových zpráv zaslaných na *localhost*.

Program `ftctl` (Fitcoin Control) slouží pro komunikaci mezi uživatelem a `fitcoind`. `ftctl` přijímá požadavky uživatele prostřednictvím příkazového řádku a tyto požadavky převádí do síťové podoby a následně pošle běžící instanci `ftcd`, který daný požadavek vykoná. Po vykonání požadavku `ftcd` vrátí odpověď zpátky programu `ftctl`, který zobrazí její obsah v lidsky čitelné podobě. `ftctl` dokáže zpracovat např. požadavky pro generování nových klíčů,

vytváření nových transakcí a výpis celkového zůstatku uživatele. Obrázek (5.1) ilustruje komunikaci mezi uživatelem, `ftcd` a `ftctl`.



Obrázek 5.1: Programy ftcd (Fitcoin Daemon) a ftctl (Fitcoin Control)

5.2 Kryptografie

Jelikož FITCOIN klade důraz na přenositelnost mezi systémy unixového typu, používá co nejmenší počet externích knihoven. Jednou z používaných externích knihoven je OpenSSL [22], případně LibreSSL [23], která poskytuje potřebné kryptografické metody. Z těchto metod je používána hašovací funkce SHA256 pro vytvoření 256-bitového haše a algoritmus ECDSA s doménovými parametry *secp256k1* pro generování veřejných klíčů. FITCOIN z bezpečnostních a praktických důvodů neimplementuje vlastní kryptografické funkce.

5.3 Adresa

Adresa ve FITCOINu je reprezentována jako haš veřejného klíče. Veřejný klíč je přitom odvozen z klíče soukromého pomocí algoritmu ECDSA. Haš je vytvořen pomocí hašovací funkce SHA256, a má tedy délku 256 bitů, resp. 32 bajtů. Pro čitelnost je daný haš uživatelovi reprezentován v hexadecimální podobě. Pro názornost je níže je uveden příklad soukromého klíče, příslušného veřejného klíče a odvozené FITCOINové adresy.

Soukromý klíč:
`A5887C0EE27C5F5BF712B94B0482CFF052F1F011EE37E4D34E6C5C472C06F28B`

Veřejný klíč:
`04FA70ADAFD9DB379C9A93F85A41CBC6E422098014593AD96FC414E661F4ED2F CAD4D5E03FC7EEC8A8B1B80ED0A5738B4E9032E8A015E31AF61A05627035C765 BE`

Adresa:
`e6c929886c3d140023de37bc3174a27950019e10ba7cdf7c278351ab93467458`

5.4 Základní datové struktury

FITCOIN používá mnoho struktur, pro další výklad však bude stačit znát základní datové struktury, které budou využity v implementaci této práce. Pro úplnost byl uveden i popis struktury tx reprezentující transakci, který bude využit pro vysvětlení struktury io, není však přímo využit v implementaci této práce.

Tabulka 5.1: Struktura tx

Název	Typ	Velikost	Popis
size	uint16_t	2 B	velikost transakce
numi	uint8_t	1 B	počet vstupů transakce
numo	uint8_t	1 B	počet výstupů transakce
io	struct io[]	32*n B	seznam vstupů a výstupů transakce

Struktura tx reprezentuje transakci. Obsahuje informaci o její velikosti, počtu vstupů a výstupů a seznamu vstupů a výstupů. Transakce může obsahovat až 256 vstupů a výstupů, FITCOIN však prozatím dovoluje mít v transakci maximálně dva výstupy. Jeden pro příjemce a druhý pro sebe v případě přeplatku.

Tabulka 5.2: Struktura io

Název	Typ	Velikost	Popis
addr	unsigned char[32]	32 B	adresa
amount	uint32_t	4 B	částka

Struktura io obsahuje FITCOINovou adresu a hodnotu. Účel struktury závisí na konkrétním využití. Strukturu lze využít jako:

- vstup transakce
- výstup transakce

- adresu a její zůstatek

Pokud je struktura io využita jako vstup do transakce, reprezentuje zdrojovou adresu, z jejíhož zůstatku se čerpá daná hodnota. Naopak pokud je použita jako výstup transakce, reprezentuje cílovou adresu, k jejímuž zůstatku se přičítá daná hodnota. Pokud je io vyzužita jako zůstatek na adrese, reprezentuje adresu a její zůstatek.

Tabulka 5.3: Struktura src

Název	Typ	Velikost	Popis
have	struct io	36 B	io
prev	struct src*	8 B	ukazatel na předchozí src
next	struct src*	8 B	ukazatel na následující src

Struktura src reprezentuje prvek obousměrného spojového seznamu zůstatků na adresách uživatele. Seznam je využit např. k vytváření seznamu vstupů do transakce. Seznam je podle nově vytěžených bloků průběžně aktualizován, přičemž je po celou dobu seřazen sestupně podle zůstatku na adresách. Takové seřazení má tu výhodu, že všechny adresy s nulovými zůstatky se nachází na konci seznamu. Při vybírání vstupů do transakce je pak možné seznam projít zleva doprava bez toho, aniž by se procházelo zbytečně přes adresy s nulovými zůstatky.

Teoreticky by bylo možné zůstatek na adrese odvodit ze samotného blockchainu. Pro větší počet bloků je to však příliš neefektivní.

5.5 Adresářová struktura

Při prvním spuštění démona `ftcd` je vytvořen adresář `.ftc`, který má následující strukturu:

- `chain` ..... Bloky blockchainu
- `coins` ..... Adresy a jejich zůstatky
- `fresh` ..... Transakce, které zatím nebyly vytěženy
- `peers` ..... IP adresy peerů
- `wallet` ..... Soukromé klíče s příslušnými adresami

Pro tuto práci je nejdůležitější adresář `wallet`, který bude popsán detailněji v následující kapitole.

Kapitola 6

Implementace

V této kapitole bude popsána implementace navržené struktury peněženky, která nyní obsahuje jediný soubor `wallet.db` implementovaný jako databáze `sqlite3`. Databáze obsahuje jedinou tabulku `Keys` pro soukromé klíče. Tyto klíče jsou indexované odpovídajícími adresami. Změna struktury peněženky měla za důsledek i změnu struktury adresáře `coins`, která uchovává informace o zůstatcích na adresách. Přestože je to jiný adresář než `wallet`, slouží právě pro využití peněženkou, a tudíž byla pro ni kvůli konzistentnosti implementována podobná struktura: `sqlite3` databáze zůstatků. Výsledná adresářová struktura vypadá následovně:

```
chain
├── coins
│   └── coin.db
├── fresh
├── peers
└── wallet
    └── wallet.db
```

6.1 Obecný přehled

Popis implementovaných funkcionalit bude pro lepší přehled uveden ve stejném pořadí, v kterém jsou používány při spuštění programu `ftcd`.

Při spuštění programu `ftcd` nemusí adresářová struktura `.ftc` či některé její podadresáře existovat. V takovém případě program adresáře vytvoří. V případě prvotního vytvoření adresáře `coins` je na základě aktuální podoby blockchainu aktualizována databáze zůstatků. Musí být proto předem zajištěno, aby daná databáze v `coins` existovala, a proto je před tím zavolána funkce **`init_coindb`**, která databázi zůstatků vytvoří.

Před připojením do peer-to-peer sítě musí být vytvořena peněženka a do ní načteny informace o zůstatcích na adresách uživatele. K tomu slouží funkce `loadwallet`, která inicializuje peněženku a načte do ní dané adresy a zůstatky. V rámci inicializace peněženky je inicializována databáze klíčů a specifikovány parametry eliptické křivky podle `secp256k1`. Pro načtení adres uživatele a příslušných zůstatků byla použita SQL konstrukce JOIN mezi tabulkou klíčů a tabulkou zůstatků. Pro práci se dvěma tabulkami z různých databází bylo potřeba databázi zůstatků dočasně připojit k databázi klíčů pomocí SQL příkazu `ATTACH DATABASE`. Po načtení potřebných hodnot již není potřeba mít databázi zůstatků připojenou, a proto je odpojena.

Uzel `ftcd` může po svém připojení do peer-to-peer sítě těžit nové bloky či vytěžené bloky od ostatních uzlů přijímat. V obou případech musí být na základě těchto bloků aktualizována databáze zůstatků. Byly proto implementovány funkce **`deposit`** a **`withdraw`**, které na základě transakcí ve vytěžených blocích aktualizují stav adres odebíráním či přičítáním k zůstatku. Dále byla implementována funkce **`insert_key`**, která uloží vygenerovaný klíč do databáze klíčů. Klíč je buď vygenerován uživatelem pomocí příkazu nebo v rámci vytvoření druhého výstupu v transakci.

6.2 Struktura databází

Tabulka 6.1 znázorňuje tabulku klíčů. Obsahuje soukromé klíče indexované příslušnými adresami.

Tabulka 6.1: Tabulka klíčů

Address	Key
0xf28...	0xB48...
0x265...	0x508...
0x802...	0x03A...
0x16e...	0x17D...

Tabulka 6.2 znázorňuje tabulku zůstatků. Obsahuje informace o zůstatcích na adresách.

Tabulka 6.2: Tabulka zůstatků

Address	Amount
0xf28...	1024
0xf95...	0
0x802...	42
0xbd8...	999

6.3 Seznam změněných souborů

V rámci této práce byly vytvořeny soubory `dbs.h` a `dbs.c`. Dále byly změněny existující soubory `wallet.c`, `wallet.h`, `io.c`, `io.h` a `key.c`. V `wallet.c` byly změněny funkce `initwallet` a `loadwallet`. Z `io.c` byly změněny funkce `deposit` a `withdraw`.

6.4 Inicializace databáze zůstatků

Inicializace databáze zůstatků pomocí funkce **`init_coindb`** probíhá ve dvou krocích: otevření, případně vytvoření databáze, a vytvoření tabulky zůstatků. Tabulka je typu klíč-hodnota, kde klíčem je adresa a hodnotou je zůstatek na této adrese. Pro vytvoření tabulky je používán příkaz `CREATE TABLE`.

6.5 Inicializace databáze klíčů

Inicializace databáze klíčů probíhá ve funkci **`init_keydb`** a probíhá obdobně jako inicializace databáze zůstatků s tím rozdílem, že příslušná tabulka obsahuje adresu a odpovídající soukromý klíč. Zároveň jsou pro tuto databázi nastavena přístupová práva, a to tak, aby do něho mohl zapisovat či z něho číst pouze majitel souboru. Ostatní uživatelé nesmí mít možnost se souborem nijak manipulovat či z něho číst.

6.6 Inicializace peněženky

Inicializace peněženky, nebo-li struktury `wallet`, probíhá v interní funkci **`initwallet`**. `wallet` obsahuje informace potřebné pro správu klíčů uživatele a poskytuje rozhraní pro vytváření transakcí. Transakce byly implementovány v rámci jiné práce, a proto nebude jejich struktura či vytvoření v této práci popisována. Tabulka (6.3) popisuje položky struktury `wallet`. V rámci inicializace peněženky je inicializována databáze klíčů pomocí **`init_keydb`** a parametry eliptické křivky.

Tabulka 6.3: Struktura wallet

Název	Typ	Velikost	Popis
`keydb`	`sqlite3*`	4-8 B	ukazatel na databázi klíčů
`balance`	`uint32_t`	4 B	celkový zůstatek na peněžence
`num`	`uint32_t`	4 B	celkový počet spravovaných adres
`head`	`struct src*`	4-8 B	ukazatel na první prvek spojového seznamu struktur src
`tail`	`struct src*`	4-8 B	ukazatel na poslední prvek spojového seznamu struktur src

6.7 Připojení a odpojení databáze

Pro připojení a odpojení databází slouží funkce `attach_db` a `detach_db`. Využívají k tomu SQL příkazy `ATTACH DATABASE` a `DETACH DATABASE`.

6.8 Načtení adres a zůstatků do paměti

Načítání adres a zůstatků do paměti má na starosti funkce `select_sources`, která využívá SQL konstrukci `OUTER JOIN` k spojení výsledků dotazu `SELECT` na tabulkách klíčů a zůstatků. Některé uživatelovy adresy nemusí mít záznam v databázi zůstatků. V takovém případě by ve výsledné tabulce měla mít nulový zůstatek. `sqlite3` však defaultně nastaví hodnotu zůstatku na `NULL`, a proto je potřeba pomocí příkazu `IFNULL` nastavit defaultní hodnotu zůstatku na nulu.

Nalezené výsledky jsou funkcí vraceny ve formě spojového seznamu `result_list`, kde prvkem seznamu je struktura `result` obsahující adresu a odpovídající zůstatek (viz tabulky 6.4 a 6.5). Tyto výsledky jsou jakožto zdroje načteny do peněženky funkcí `loadwallet`. Důvodem, proč nejsou zdroje načteny rovnou do peněženky v `select_sources`, je zabránění cirkulární závislosti mezi `wallet.h` a `dbs.h`, uchování úrovně abstrakce rozhraní souboru `wallet.h` a zvýšení „loose coupling“ kódu. Soubor `wallet.h` importuje soubor `dbs.h`. Kdyby načítání zdrojů do peněženky mělo být provedeno v `select_sources`, musel by `dbs.h` mít přístup ke struktuře `wallet` a funkcím ve `wallet.h`. Tím pádem by musela importovat `wallet.h`, čímž by vznikla cirkulární závislost. Takový návrh je však obecně nedoporučený. Navíc by musel mít `dbs.h` přístup k funkci `addsrc`, která přidává zdroje do peněženky. Tato funkce je však používána interně ve `wallet.c` a pro použití v jiných souborech by musela být zahrnuta do rozhraní souboru `wallet.h`. Zároveň by funkce v `dbs.h` měly sloužit pouze pro přímou práci s databází a neměly by mít žádnou znalost o peněžence.

Tabulka 6.4: Struktura result_list

Název	Typ	Velikost	Popis
`head`	`struct *result`	4-8 B	ukazatel na první prvek seznamu
`tail`	`struct *result`	4-8 B	ukazatel na poslední prvek seznamu

Tabulka 6.5: Struktura result

Název	Typ	Velikost	Popis
`address`	`char[256]`	256 B	adresa
`amount`	`uint32_t`	4 B	zůstatek
`next`	`struct result*`	4-8 B	ukazatel na následující prvek

6.9 Aktualizace databáze zůstatků

K aktualizaci databáze zůstatků na základě nově vytěžených bloků slouží funkce **deposit** a **withdraw**. deposit na začátku pomocí funkce **coindb_addr_exists** kontroluje, zda zadaná adresa v databázi již neexistuje. Pokud ano, je zavolána funkce **deposit_to_coindb**, která pomocí SQL příkazu UPDATE k zůstatku na dané adrese přičte danou částku. V opačném případě je zavolána funkce **insert_to_coindb**, která pomocí INSERT do databáze přidá nový záznam se zadanou adresou a zůstatkem. Teoreticky by přičtení k zůstatku a vytvoření zůstatku mohlo být provedeno pomocí jednoho sql příkazu INSERT OR REPLACE. Ten však záznam neaktualizuje, nýbrž nahrazuje, a proto nebyl využit. withdraw naopak slouží k odebírání peněz z adresy. Na začátku kontroluje, zda zadaná adresa v databázi existuje. Následně pomocí funkce **read_amount** přečte z databáze příslušný zůstatek a kontroluje, zda-li je na adrese dostatek peněz. Poté je zavoláním funkce **withdraw_from_coinsdb** od zůstatku odebírána zadaná částka.

6.10 Aktualizace databáze klíčů

Databázi klíčů lze měnit dvěma způsoby: buď si uživatel nechá vygenerovat nový klíč, nebo je nový klíč vygenerován v rámci druhého výstupu uživatelem vytvořené transakce. K přidání vygenerovaného klíče do databáze je nejdřív pomocí funkce keydb_addr_exists kontrolováno, zda adresa v databázi již neexistuje. Poté je pomocí funkce **insert_key** do databáze vložen klíč.

6.11 Testování

Součástí zadání je i řádné otestování implementace. Testování bylo provedeno pomocí unit testů rozdělených do souborů: test-dbs.c, test-wallet.c a test-io.c. Testy lze spustit pomocí příkazu make test.

6.11.1 test-dbs.c

V tomto souboru jsou testovány funkce souboru dbs.c, kde jsou funkce pro práci s databází sqlite3. Jsou testovány následující scénáře:

- Inicializace databází
- Vkládání záznamů do databází
- Kontrola existence záznamů
- Kontrola dostatku peněz na adrese
- Vklad peněz na adresu

- Čtení zůstatků adresy
- Připojování a odpojování databází
- Vybírání seznamu adres a zůstatků

6.11.2 test-wallet.c

V tomto souboru jsou testovány funkcionality ve `wallet.c` implementované v rámci této práce. Podle domluvy s vedoucím byly do tohoto souboru zakomponovány některé testy ze souboru `test-wallet.c` z větve wallet_havrujir na gitlabu. Testy testují následující scénáře:

- Vytvoření peněženky a načtení adres a zůstatků do ní
- Generování nových klíčů a adres
- Vklad a výběr peněz z adresy
- Vytváření transakcí a jejich synchronizace s peněženkou

6.11.3 test-io.c

V tomto souboru jsou testovány funkce ze souboru `io.c`. Testovány jsou následující scénáře:

- Vklad peněz na adresu
- Výběr peněz z adresy

6.11.4 Testování platforem

Implementace byla testována na různých platformách. Testováno bylo zejména to, jestli se program úspěšně zkompiluje a spustí. FITCOIN byl testován na následujících platformách:

- MacOSX, x86_64, Little endian
- Ubuntu, x86_64, Little endian
- OpenBSD, amd64, Little endian

```
$ sudo apt-get install libssl-dev
$ sudo apt-get install libsqlite3-dev
```

Listing 4: Instalace externích knihoven OpenSSL a Sqlite3 na Debianu

6.12 Instalace

K instalaci programu FITCOIN je potřeba mít kompilátor a linker jazyka C a nástroj make, které jsou standardní pro většinu UNIXových operačních systémů. Dále je potřeba mít nainstalovanou kryptografickou knihovnu OpenSSL a a knihovnu pro databázi Sqlite3. V případě Debianu lze knihovny nainstalovat následovně:
Samotný program FITCOIN lze poté instalovat jednoduchým příkazem:

```
$ make install
```

Listing 5: Instalace programu FITCOIN

6.13 Kompilace

Program je kompilován jednoduchým příkazem `make`:

```
$ make
```

Listing 6: Kompilace programu FITCOIN

6.14 Spuštění

Program `ftcd` je spuštěn pomocí:

```
$ ./ftcd
```

Druhá možnost je program nechat běžet na pozadí. Preferovaný způsob je ponechán na uživatelovi. Je však doporučeno pustit `ftcd` na popředí a komunikovat s nim prostřednictvím `ftctl` v druhém terminálu, a to pro oddělení zpráv, které programy vypisují do terminálu.

Při prvotním spuštění je inicializována adresářová struktura `.ftc`. Jelikož je v tuto chvíli databáze klíčů prázdná, nemá peněženka žádné zdroje. Z tohoto důvodu je na přiloženém CD dostupný adresář `wallet`, který obsahuje soukromý klíč[7], jehož adresa je ohodnocena v rámci genesis bloku. V

[7]Tento soukromý klíč je schválně veřejně dostupný a slouží pouze pro experimentální účely.

případě klónování repozitáře z gitlabu lze adresář najít na větvi této práce (wallet_structure). Tento adresář uživatel zkopíruje do adresáře `.ftc`. Při dalším spuštění `ftcd` bude adresa ohodnocená 1024 mincemi přidána do peněženky pro použití.

Programu `ftcd` jsou posílány zprávy prostřednictvím `ftctl`. Pro tuto práci jsou relevantní pouze příkazy `newaddr` a `pay`. Příkaz `newaddr` vygeneruje nový klíč a uloží ho do databáze klíčů:

```
$ echo newaddr | ./ftctl
b69b19a22523fca3211735f4a099ffd4704f0cf4debcf2c554c1a4f9620c6115
```

Po vykonání příkazu `newaddr` následuje výpis vygenerované adresy. To se hodí pro případ, kdy chceme s adresou dále pracovat. Např. lze na ni posílat peníze:

```
$ echo "pay b69b19a22523fca3211735f4a099ffd4704f0cf4debcf2c554c1
$ a4f9620c6115" 512 | ./ftctl
```

Příkaz provede platbu na zadanou adresu. Následně je podle toho aktualizována databáze zůstatků.

Pro zobrazení obsahu databází je lze otevřít pomocí příkazu `sqlite3`:

```
$ sqlite3 $HOME/.ftc/wallet/wallet.db

$ sqlite3 $HOME/.ftc/coins/coin.db
```

Po otevření lze na databázích provádět SQL dotazy.

Závěr

Prvním cílem práce bylo vysvětlit kryptografické metody využívané v kryptoměnách. Dalším cílem bylo seznámit čtenáře s obecnými principy fungování kryptoměn. Následujícím cílem bylo analyzovat strukturu peněženky kryptoměny Bitcoin. Spolu s tím měla být provedena stručná rešerše databází dostupných pro UNIXové systémy a diskutováno, které z nich se hodí pro účely praktické části. Cílem praktické části bylo pro kryptoměnu FITCOIN navrhnout strukturu peněženky, která je únosná i pro větší počet klíčů. Součástí práce má být i implementace navržené struktury.

Cíle bakalářské práce byly splněny. Kryptografické metody využívané v kryptoměnách byly vysvětleny. Obecné principy fungování kryptoměn byly taktéž popsány. Dále byla analyzována struktura peněženky kryptoměny Bitcoin a provedena rešerše databází dostupných pro UNIXové systémy. Zároveň bylo diskutováno, které z těchto databází se hodí pro účely praktické části. Byla navržena a implementována nová struktura peněženky, která je únosná i pro větší počet klíčů.

Bibliografie

1. LÓRENZ, Róbert. Bezpečnost: 5. Hašovací funkce, MD5, SHA-x, HMAC [online]. 2020. Dostupné také z: https://courses.fit.cvut.cz/BI-BEZ/media/bez5.pdf. [cit. 2020-04-15].
2. AZAD, Kalid. *Understanding the Birthday Paradox*. Dostupné také z: https://betterexplained.com/articles/understanding-the-birthday-paradox/. [2020-06-21].
3. MARTÍNEZ, V. Gayoso; ENCINAS, L. Hernández; ÁVILA, C. Sánchez. A Survey of the Elliptic Curve Integrated Encryption Scheme [online]. 2010. Dostupné také z: https://www.researchgate.net/publication/255970113_A_Survey_of_the_Elliptic_Curve_Integrated_Encryption_Scheme. [cit. 2020-07-12].
4. COMMONS, Wikimedia. *File:EllipticCurveCatalog.svg — Wikimedia Commons, the free media repository*. 2014. Dostupné také z: https://commons.wikimedia.org/w/index.php?title=File:EllipticCurveCatalog.svg&oldid=138599442. [Online; accessed 21-June-2020].
5. NAKAROV, Svetlin; MILEN, Stefanov; SHIDEROFF, Marina. *Practical Cryptography for Developers, ECDSA: Elliptic Curve Signatures* [online]. Sofia, 2018. ISBN 978-619-00-0870-5. Dostupné také z: https://github.com/nakov/practical-cryptography-for-developers-book. [cit. 2020-04-15].
6. LÓRENZ, Róbert. *Bezpečnost: 8. Základy kryptografie eliptických křivek a kvantové kryptografie* [online]. 2020. Dostupné také z: https://courses.fit.cvut.cz/BI-BEZ/media/bez8.pdf. [cit. 2020-04-15].
7. NAKAROV, Svetlin; MILEN, Stefanov; SHIDEROFF, Marina. *Practical Cryptography for Developers, Elliptic Curve Cryptography (ECC)* [online]. Sofia, 2018. ISBN 978-619-00-0870-5. Dostupné také z: https:

//github.com/nakov/practical-cryptography-for-developers-book. [cit. 2020-07-13].

8. BROWN, Daniel R. L. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters [online]. 2010. Dostupné také z: http://www.secg.org/sec2-v2.pdf. [cit. 2020-07-13].
9. NARAYANAN, Arvind. *Bitcoin and cryptocurrency technologies: a comprehensive introduction* [online]. Princeton: Princeton University Press, 2016. ISBN 978-0691171692. Dostupné také z: https://bitcoinbook.cs.princeton.edu/. [cit. 2020-04-15].
10. LÓRENZ, Róbert. *Bezpečnost: 6. RSA, kryptografie s veřejným klíčem, DSA, El-Gamalův algoritmus* [online]. 2020. Dostupné také z: https://courses.fit.cvut.cz/BI-BEZ/media/bez6.pdf [cit. 2020-05-23].
11. ANTONOPOULOS, Andreas M. *Mastering Bitcoin: Programming the Open Blockchain* [online]. 2. vyd. Sebastopol, CA: O'Reilly, 2017. ISBN 978-1491954386. Dostupné také z: https://github.com/bitcoinbook/bitcoinbook. [cit. 2020-04-15].
12. NAKAMOTO, Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System [online]. 2008. Dostupné také z: https://bitcoin.org/bitcoin.pdf. [cit. 2020-04-15].
13. LANSKY, Jan. Possible State Approaches to Cryptocurrencies [online]. 2018. Dostupné také z: http://www.si-journal.org/index.php/JSI/article/viewFile/335/325. [cit. 2020-07-29].
14. BUTERIN, Vitalik. Design Rationale [online]. Dostupné také z: https://eth.wiki/en/fundamentals/design-rationale. [cit. 2020-07-13].
15. Bitcoin Core file system [online]. Dostupné také z: https://github.com/bitcoin/bitcoin/blob/master/doc/files.md. [cit. 2020-06-27].
16. Data directory [online]. 2017. Dostupné také z: https://en.bitcoinwiki.org/wiki/Data_directory#database_subdirectory. [cit. 2020-06-27].
17. LONGSHAO007. why not update berkeley-db from 4.8.30 to 6.1 ? [online]. 2017. Dostupné také z: https://github.com/bitcoin/bitcoin/issues/6253. [cit. 2020-06-27].
18. ISQ. How many files can I put in a directory? [online]. 2018. Dostupné také z: https://stackoverflow.com/questions/466521/how-many-files-can-i-put-in-a-directory. [cit. 2020-06-27].
19. RAMESH; AHRENS, Raphael. The ls command is not working for a directory with a huge number of files [online]. 2017. Dostupné také z: https://unix.stackexchange.com/questions/120077/the-ls-command-is-not-working-for-a-directory-with-a-huge-number-of-files. [cit. 2020-06-28].

20. DB, Berkeley. Database limits [online]. Dostupné také z: https://docs.oracle.com/cd/E17275_01/html/programmer_reference/am_misc_dbsizes.html. [cit. 2020-07-23].

21. SQLITE. Appropriate Uses For SQLite [online]. Dostupné také z: https://www.sqlite.org/whentouse.html. [cit. 2020-07-23].

22. OPENSSL. OpenSSL: Cryptography and SSL/TLS Toolkit [software]. 1999-2017. Dostupné také z: https://www.openssl.org/. [cit. 2020-07-06].

23. LIBRESSL. LibreSSL [software]. 2014-2017. Dostupné také z: http://www.libressl.org/. [cit. 2020-07-06].

Příloha A

Seznam použitých zkratek

BTC Bitcoin

ECC Elliptic Curve Cryptography

UTXO Unspent Transaction Output

TPS Tradiční Platební Systém

EPS Elektronický Platební Systém

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `fitcoin` ..... adresář se spustitelnou formou implementace
- `wallet` ..... adresář se soukromým klíčem a adresou v genesis bloku
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF