FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	blockchain pro FITCOIN
Student:	Mikuláš Dvořák
Vedoucí:	Mgr. Jan Starý, Ph.D.
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Vypracujte následující části systému digitální měny FITCOIN (BP Tománek, Dvořák, Pajskr):

1. Popište technologii blockchainu tak jak je používána
 - v současných kryptoměnách
 - v turingovsky úplných výpočetních prostředích (Ethereum)
 - v dalších aplikacích nesouvisejících s platbami.
2. Navrhněte co nejjednodušší strukturu blockchainu pro FITCOIN.
3. Implementujte metody pro základní práci s blockchainem: kontrola integrity, těžení nových bloků.
4. Implementaci vytvořte v jazyce C++. Dbejte na korektnost a přenositelnost kódu přinejmenším mezi POSIX systémy.
5. Aplikaci řádně zdokumentujte, nejlépe ve formě standardní manuálové stránky.
6. Aplikaci řádně otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

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

V Praze dne 16. ledna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Blockchain pro FITCOIN

Mikuláš Dvořák

Katedra teoretické informatiky
Vedoucí práce: Mgr. Jan Starý, Ph.D.

15. května 2018

Poděkování

Rád bych poděkoval Mgr. Janu Starému, Ph.D., za velkou trpělivost, rady a čas, který mi věnoval při vedení mé bakalářské práce. Dále bych rád poděkoval své rodině a jmenovitě svému otci za opravu pravopisných chyb.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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 předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, 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 způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2018 ……

Odkaz na tuto práci

Dvořák, Mikuláš. *Blockchain pro FITCOIN*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Cílem této bakalářské práce je zahrnout technologii blockchain do školní kryptoměny FITCOIN. Blockchain je decentralizovaná databáze, do které se dají přidávat pouze nové záznamy. V teoretické části práce je zanalyzovaná technologie blockchainu v kryptoměnách Bitcoin a Ethereum. Dále jsou v práci navrženy struktura blockchainu pro školní kryptoměnu FITCOIN a způsob na přidávání nových bloků do blockchainu. V praktické části je pak v jazyce C++ implementované řešení, které je zahrnuto ve výsledném projektu FITCOIN.

Klíčová slova blockchain, fitcoin, bitcoin, kryptoměna

Abstract

The aim of this bachelor thesis is to include the blockchain technology into the FITCOIN (trivial cryptocurrency based on Bitcoin). Blockchain is a decentralized append-only database on a P2P network. In the theoretical part of the thesis the blockchain technology in Bitcoin and Ethereum is analyzed. In the next part I am explaining the design behind a blockchain in FITCOIN. In the practical part solution is implemented in programming language C++.

Keywords blockchain, fitcoin, bitcoin, cryptocurrency

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Kryptoměny jsou velmi diskutované téma, protože představují platební systém, který nespoléhá na autority. Kvůli rostoucí popularitě tohoto tématu je užitečné rozumět strukturám, na kterých jsou kryptoměny postaveny.

Jednou z takových struktur je blockchain. Blockchain představuje databázi, do které se zaznamenávají veškeré transakce prováděné uživateli sítě. Díky této struktuře a mnoha technickým podrobnostem lze vybudovat platební systém, který funguje mezi uživateli bez nutnosti existence bank.

Tato práce se zaměřuje převážně na technologii blockchain a její použití ve školní kryptoměně FITCOIN. Práce je prospěšná pro osoby, které zajímá technická stránka a možné využití technologie blockchain, a pro budoucí studenty, kteří naší nově vznikající školní kryptoměnu budou chtít vylepšovat.

V této bakalářské práci se zaměřím na technologii blockchain a její využití v kryptoměnách i jiných aplikacích.

V práci se zabývám analýzou struktury blockchainu v kryptoměně Bitcoin a navrhuji, jak tuto strukturu upravit a použít ve školní kryptoměně FITCOIN.

FITCOIN je projektem, který postupně vzniká z několika bakalářských prací. Výsledkem těchto prací by měla být funkční kryptoměna, mým úkolem je vytvořit pro tuto kryptoměnu blockchain a jednoduchého těžaře. Součástí projektu jsou bakalářské práce:

- Jan Tománek - FITCOIN: Stromy transakcí,
- Bc. Karel Pajskr - FITCOIN: *peer-to-peer communication*,
- Mikuláš Dvořák - Blockchain pro FITCOIN,
- Rostislav Kaufman - FITCOIN: webový náhled na blockchain,
- Minh Trieu Quang - FITCOIN: *an Android light wallet*.

Kapitola 1

Teoretický úvod

V této kapitole nadefinuji potřebné základy pro pochopení principů technologie blockchain. Hašovací funkci jsem nadefinoval s přihlédnutím do knihy *Handbook of Applied Cryptography* [15] a informací získaných na přednášce předmětu Bezpečnost [14] vyučovaném na naší škole. Při definování Merkleových stromů jsem vycházel z patentu [17], který tyto stromy využívá. Informace k peer-to-peer síti jsem získal z článku [22].

1.1 Hašovací funkce

Hašovací funkce je matematická funkce, která převede libovolná vstupní data na data fixní velikosti.

Předpisem takové funkce může být $f : \{0,1\}^m \to \{0,1\}^n$, kde m je velikost vstupních dat v bitech a n je fixní délka výstupního haše.

Rozšířením takové funkce jsou kryptografické hašovací funkce, které mají ještě další vlastnosti, jako jsou bezkoliznost a jednosměrnost. Dodržením těchto vlastností má výsledná hašovací funkce funkci náhodného orákula.

1.1.1 Náhodné orákulum

Náhodným orákulem nazýváme libovolný stroj, který na základě vstupu odpovídá náhodným výběrem výstupu z množiny výstupů. Má pouze tu vlastnost, že na tentýž vstup odpovídá stejným výstupem.

1.1.2 Jednosměrnost

Jednosměrná funkce je taková funkce, pro kterou je výpočetně příliš náročné zjistit z výstupu, jaký byl vstup.

Funkce f je jednosměrná, pokud je pro každý vzor x definičního oboru lehké vypočítat obraz $f(x)$, ale pro kterýkoliv y z oboru hodnot funkce f je výpočetně příliš náročné vyřešit rovnici $y = f(x)$, za znalosti obrazu y.

1.1.3 Bezkoliznost

Bezkoliznost dělíme na dva typy. Správná kryptografická hašovací funkce musí oba typy splňovat.

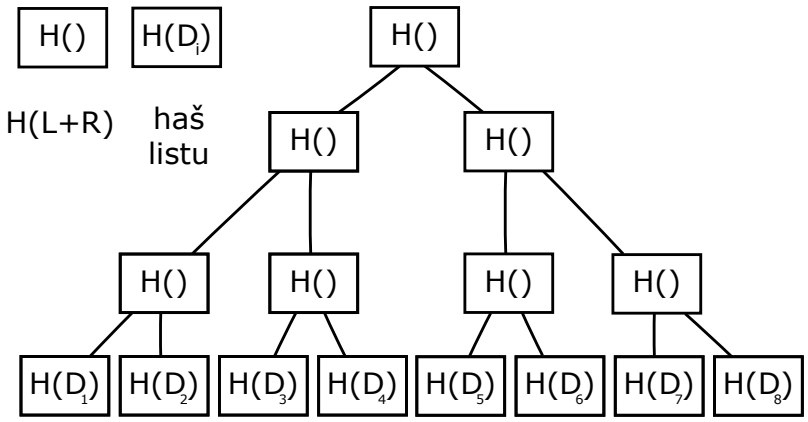
Bezkoliznost 1. řádu
Pro hašovací funkci odolnou proti kolizím prvního řádu je výpočetně nemožné najít rozdílný vstup x_2, který má stejný výsledek jako kterýkoliv předem specifikovaný vstup x_1. To znamená, že za předpokladu zadaného x_1 je nemožné najít takové x_2, aby platilo: $x_1 \neq x_2 \wedge f(x_1) = f(x_2)$.

Bezkoliznost 2. řádu
Pro hašovací funkci odolnou proti kolizím druhého řádu je výpočetně nemožné najít dva rozdílné vstupy x_1 a x_2, které mají stejný výstup. To znamená, že nedokážeme najít dva volně vybrané vstupy x_1 a x_2, pro které platí: $x_1 \neq x_2 \wedge f(x_1) = f(x_2)$.

1.2 Merkleův strom

Merkleův strom představil Ralph C. Merkle ve svém patentu [16] v roce 1979.

Merkleův strom může být obecně jakýkoliv n-ární strom, ale pro jednoduchost si ho představíme jako úplný binární strom s počtem 2^k listů, který používá kryptografickou hašovací funkci H. Vstupní data D jsou rozdělena do 2^k listů, ze kterých je strom následně zkonstruován. Hodnota těchto listů je vypočítána hašovací funkcí H, a hodnota vnitřních uzlů je vypočítána jako $H(L \oplus R)$, kde L je levý syn a R je pravý syn odpovídajícího uzlu a $\oplus$ je operace zřetězení.



Obrázek 1.1: Merkleův strom pro osm listů.

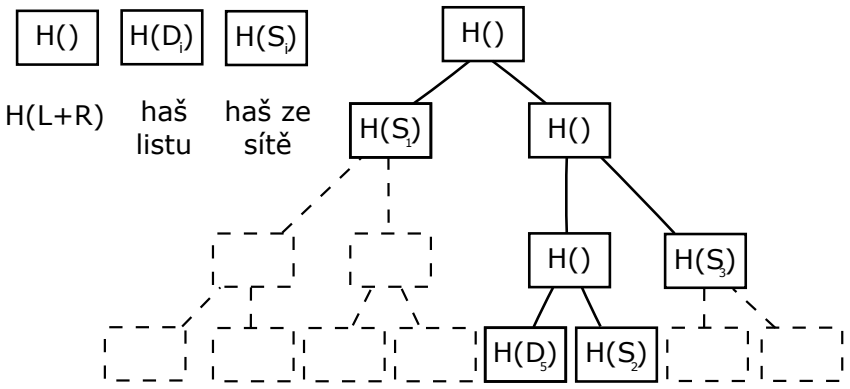
Tato datová struktura má mnohá uplatnění. My ji budeme převážně používat pro kontrolu integrity dat a pro důkaz inkluze vlastněných dat ve stromě, od kterého je znám pouze kořen.

1.2.1 Kontrola datové integrity

Za použití kryptografické hašovací funkce může hodnota kořene Merkleova stromu sloužit jako důkaz, že vlastněná data nejsou poškozena nebo změněna útočníkem. Celá myšlenka se opírá o bezkoliznost hašovací funkce. Protože najít kolizi hašovací funkce je výpočetně příliš náročné, stejně náročné je i najít kolizi haše dvou spojených hašů ve kterémkoliv uzlu Merkleova stromu. Za předpokladu znalosti správné hodnoty kořene Merkleova stromu tedy můžeme lehce ověřit, zda jsou naše data nezměněna, a to právě konstrukcí takového stromu.

1.2.2 Důkaz inkluze

Strukturu Merkleova stromu lze využít jako důkaz existence mnou vlastněných dat v Merkleově stromu bez nutnosti znalosti celého stromu. Pokud například znám hodnotu jednoho listu a někde zjistím hodnoty chybějících vnitřních uzlů, takových, že dokážu vypočítat hodnotu kořene Merkleova stromu, mohu s jistotou prohlásit, že tento list se ve stromě nachází. V kryptoměnách se takové důkazy používají kvůli úspoře místa na disku. V následujícím obrázku je zobrazena demonstrace takového důkazu. Za předpokladu, že uživatel vlastní D_5 a haš kořene stromu, stačí mu získat pouze uzly S_1, S_2 a S_3. Z těchto dat může vypočítat haš kořene Merkleova stromu a porovnat jej s vlastněným hašem vrcholu stromu.



Obrázek 1.2: Důkaz pro Merkleův strom.

1.3 Peer-to-peer síť

Peer-to-peer síť slouží jako alternativa k síťovému modelu založenému na klient-server architektuře. Tato síť využívá decentralizovaného modelu (neexistencí hierarchie klient-server), ve kterém je každý účastník označován jako peer. Tento peer má roli serveru a klienta zároveň. V této síti může peer posílat žádosti jiným peerům a zároveň na žádosti odpovídat (na rozdíl od server-klient, kde server odpovídá a klient se ptá). Jedna z výhod peer-to-peer sítě je schopnost odolávat výpadkům. Pokud se například odpojí peer, ke kterému jste připojeni, aplikace postavená na peer-to-peer síti může požádat jiného peera o spolupráci a pokračovat tam, kde jste s předchozím peerem skončili. Toto je přirozená výhoda oproti klient-server síti, kde při nedostupnosti serveru celá komunikace skončí.

Kapitola 2

Blockchain

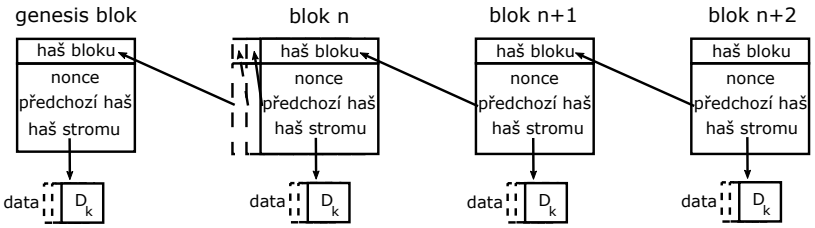
Blockchain jako technologie nebyla pevně definovaná a lze na ní nahlížet různě. Například v článku [20] autoři dělí blockchain technologii do několika kategorií, a to podle existence centrální autority a podle finanční odměny za účast na podílení se na kontrole blockchainu. Nás bude převážně zajímat blockchain, který nemá centrální autoritu a účastníci sítě jsou finančně odměněni za podílení se na kontrole blockchainu.

Pro naše účely budeme na blockchain nahlížet jako na distribuovanou decentralizovanou databázi v peer-to-peer síti, do které lze přidávat bloky záznamů. Všechny tyto záznamy jsou kontrolovány uživateli sítě a musí splňovat pravidla, na kterých je síť domluvena. Jakmile je takový blok záznamů do databáze přidán, nemůže z ní být smazán. Blockchain obsahuje záznamy, které se dají dohledat a tím ověřit jejich existence v databázi. [5]

Blockchain se skládá z na sebe navazujících bloků, které obsahují záznamy (například platby). Aby byl nový blok do blockchainu sítí schválen, musí splňovat předem dohodnutá pravidla. Těmto pravidlům se říká konsenzus. Implementace různých blockchainů se z velké části liší právě těmito pravidly.

2.1 Blok

Záznamy v blockchain databázi jsou shlukovány do bloků. Blok je datová struktura, která tyto záznamy společně s odkazem na předchozí blok obsahuje. Ze záznamů se většinou vytvoří Merkleův strom a blok obsahuje haš kořene tohoto stromu. Při vytváření nového bloku se pak zahašuje blok předchozí a nový blok na ten předchozí odkazuje tímto hašem. Pomocí tohoto procesu jsou všechny po sobě jdoucí bloky do sebe vloženy a nedají se změnit bez přepočítání všech následujících bloků. Na následujícím obrázku je vyobrazeno, jak jsou tyto bloky propojeny.



Obrázek 2.1: Řetězení bloků v blockchainu.

2.2 Genesis blok

Tímto pojmem se obecně označuje začátek blockchainu, tedy úplně první blok. Tento blok není přidán uživateli, ale je pevně nastaven v rámci protokolu.

2.3 Konsenzus

Jeden z problémů decentralizované databáze v peer-to-peer síti spočívá v domluvě na jedné společné historii. Kdyby tato databáze byla umístěna v klient-server síti, stačilo by poslat žádost na server a ten by nám poslal odpověď. Nevýhoda takové databáze je právě v centralizaci, uživatelé musí serveru důvěřovat. V blockchain databázích neexistuje centrální autorita, uživatelé si databázi spravují sami a nepotřebují si navzájem věřit. Aby došlo ke stavu, kdy si uživatelé nemusí věřit, ale vzájemně se dohodnou na společné historii transakcí, je potřeba dohodnout pravidla, která takovou historii zaručí. V principu potřebuji taková pravidla, která zaručují, že když je dodržuji a většina sítě se mnou, nemůže nikdo naši databázi změnit.

Metod, jak zaručit společnou historii, je více, já se v práci zaměřím na metodu proof-of-work a proof-of-stake.

2.3.1 Proof-of-work

Proof-of-work je algoritmus, který byl použit v systému Hashcash [2] na detekci spamu. V blockchainu se používá pro schvalování nových bloků v blockchainu.

Tento koncept využívá kryptografickou hašovací funkci a nějaká data. Tato data jsou v našem kontextu bloky z blockchainu, a aby takový blok byl algoritmem proof-of-work uznán za validní, musí po zahašovaní mít alespoň n počátečních bitových nul. Číslo n se v implementacích blockchainů často liší, ale aby bylo zajištěno, že se blok najde v síti až po určitém čase, dynamicky se mění s přihlédnutím k tomu, jak rychle byly nalezeny bloky předchozí. Aby

takový blok byl nalezen, je potřeba v hlavičce bloku mít atribut, který se může měnit, a tím se mění výsledný haš, dokud blok nesplňuje požadované omezení na počátečních n bitových nul. [18]

Tímto algoritmem se zajistí, že přidávání nových bloků je náročný a drahý proces, protože na generování nových pokusů o blok je potřeba výpočetní síla.

Hlavní motivací pro přidávání nových bloků je většinou odměna v podobě kryptoměny.

2.3.2 Proof-of-stake

Proof-of-stake navrhl na bitcoin fóru [21] uživatel QuantumMechanic jako alternativu k proof-of-work algoritmu. Dnes existuje několik kryptoměn, které z tohoto nápadu vychází.

Hlavní motivací pro změnu proof-of-work algoritmu je energetická náročnost spjatá s přidáváním nových bloků do blockchainu. Na rozdíl od proof-of-work algoritmu, kde má největší šanci vytvořit nový blok uživatel s největším výpočetním výkonem, v proof-of-stake algoritmu je to uživatel s největším množstvím měny.

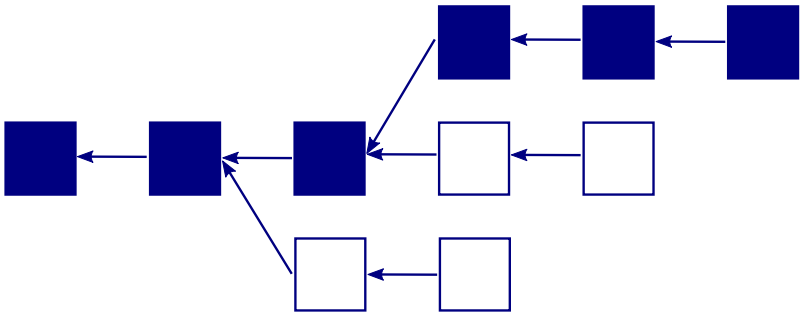
Jedna z možných implementací je použití hašovací funkce, se kterou zahašuji blok a zároveň aktuální čas. Pokud výsledný haš bude mít určitý počet počátečních nul, který se vypočítá na základě předchozích bloků a množství měny, jež vlastním, je uznán algoritmem jako validní. [6]

V tomto algoritmu má každý uživatel šanci vygenerovat nový blok právě jednou za vteřinu, takže není potřeba vlastnit velké množství výpočetní síly. Upřednostňuje to uživatele s větším množstvím měny, protože je u nich menší motivace k nevhodnému chování (například neschvalování transakcí), kterým by síť zpomalovali, a tím snižovali hodnotu celé měny.

2.4 Přidávání nových bloků

Obecně do blockchainu přidávají bloky uživatelé a dostávají za to nějakou ekonomickou odměnu (většinou v podobě kryptoměny). Když daný blockchain používá proof-of-work konsenzus, pak těžaři závodí o nalezení nového bloku a ten, kterému se to podaří, pošle do sítě nově nalezený blok rozšiřující aktuální blockchain.

Problém nastává, když dva těžaři najdou blok ve stejný okamžik a blockchain se tak rozdělí na dva: na blockchain s blokem od prvního těžaře a na blockchain s blokem od druhého těžaře. V takové chvíli se těžaři rozdělí podle toho, který blockchain právě vlastní, a snaží se najít blok nad svým blockchainem. Vznikne tím závod, a komu se podaří najít nový blok dříve, vytvoří blockchain s největší investovanou prací, a síť tento blockchain považuje za ten správný. Ve výsledku je tedy blockchain stromová struktura, ve které je historie přidávání bloků označena větví s nejvíce investovanou prací. Na následujícím obrázku je tato větev označena vyplněním bloků v nejdelší větvi.



Obrázek 2.2: Větvení blockchainu.

Kapitola 3

Blockchain v Bitcoinu

Bitcoin je peer-to-peer digitální měna, kterou vymyslel autor pod pseudonymem Satoshi Nakamoto. Tuto digitální měnu popsal v dokumentu [18]. Považuje se za první aplikaci využívající technologii blockchain.

Pro většinu informací v této kapitole jsem čerpal z neoficiální dokumentace kryptoměny Bitcoin [4] a knihy *Mastering Bitcoin* [1], a to převážně z kapitol *Mining and Consensus* a *The Blockchain*.

V Bitcoinu je blockchain struktura po sobě jdoucích bloků, které na sebe zpětně odkazují. Novější blok tedy odkazuje právě na jeden bezprostředně předchozí blok. Každý takový blok obsahuje transakce, které v síti uživatelé posílají. Uživatelé, kteří tyto bloky do sítě přidávají, se označují těžaři a jejich úkolem je nalezení správné nonce 3.1 pro proof-of-work 2.3.1 algoritmus a ověřování, že takové transakce mohou být poslány. V bloku je vždy alespoň jedna transakce, ve které jsou odměny za vytěžení bloku pro těžaře. Odměna se skládá z poplatků transakcí, které jsou zahrnuty v bloku, a z nových Bitcoinů, které jsou odměnou za nalezení bloku.

Protože celá databáze je decentralizovaná a nové záznamy mohou účastníci přidávat sami, může nastat situace, kdy vzniknou dva bloky odkazující na stejný předchozí blok. V návrhu se s takovou situací samozřejmě počítá a v praxi běžně stává.

Pro většinu identifikátorů a proof-of-work používá Bitcoin hašovací funkci SHA256. Tato hašovací funkce převádí libovolný vstup na data o velikosti 256 bitů.

3.1 Struktura bloku

Blok je objekt, který má hlavičku 3.1 a sérii transakcí, jež si uživatelé sítě mezi sebou posílají.

Hlavička bloku obsahuje informace o bloku a slouží jako unikátní identifikátor. Má pevnou velikost 80 bajtů a její struktura je popsána v tabulce.

velikost v bajtech	název	popis
4	verze	Verze protokolu, ve které byl blok vytěžen
32	identifikátor předchozí hlavičky 3.2	SHA256(SHA256()) haš předchozího bloku
32	haš Merkleova stromu 3.3	SHA256(SHA256()) haš Merkleova stromu
4	čas 3.4	Čas vytěžení bloku v unix epoch formátu. Vyjadřuje čas, kdy těžař začal těžit daný blok
4	bits 3.5	Číslo, které udává náročnost nalezení správné nonce pro proof-of-work 2.3.1 algoritmus
4	nonce	Libovolné číslo, které těžař může měnit, aby splnil proof-of-work 2.3.1 algoritmus

Tabulka 3.1: Struktura hlavičky bloku v kryptoměně Bitcoin. Volně přeloženo z [4]

3.2 Identifikátor bloku

V Bitcoinu se blok identifikuje podle hlavičky bloku za použití hašovací funkce SHA256, která se použije na všechna data spojená za sebou z hlavičky bloku. Hašovací SHA256 funkce se použije dvakrát, poprvé na data v hlavičce a podruhé na výsledný haš.

3.3 Merkleův strom

V Bitcoinu se Merkleův strom používá na kontrolu integrity transakcí. K tomuto účelu slouží haš kořene tohoto stromu, který se nachází v hlavičce bloku.

Důležitou roli hrají Merkleovy stromy pro uživatele, kteří nemají dostatek místa na disku a používají odlehčeného klienta (např. peněženku pro mobilní telefon). Tito uživatelé si nemusí stahovat celý blockchain, ale stačí jim pouze hlavičky bloků celého blockchainu. V okamžiku, kdy si tento odlehčený klient chce ověřit, zda transakce, kterou odeslal, již proběhla, zeptá se na důkaz peera, který vlastní celý blockchain. Důkazem je část Merkleova stromu, ve kterém jsou haše okolních uzlů. Z těchto hašů si pak klient spočítá haš kořene Merkleova stromu, a pokud se shoduje s kořenem z hlavičky bloku, pak se transakce v daném bloku nachází. Toto využití popsal autor Bitcoinu Satoshi Nakamoto v dokumentu [18], ve kterém byl Bitcoin poprvé představen. [12]

Bohužel má tento přístup nevýhodu ve ztrátě anonymity. Pokud se odlehčený klient zeptá klienta s celým blockchainem na jednu transakci, pak je

pravděpodobné, že tuto transakci provedl právě tento odlehčený klient. V praxi se s tímto problémem Bitcoin protokol vypořádává pomocí Bloom filtru a použitím částečného Merkleova stromu. Vylepšení bylo navrhnuto v BIP-0037 [11] a později implementováno v Bitcoin Core aplikaci [3]. Já toto řešení budu využívat ve své aplikaci. Podrobnější popis je v sekci Odlehčený klient 6.7.

3.4 Čas

Atribut čas je údaj o čase, ve kterém začal těžař tento blok těžit v unix epoch formátu. Celá síť jako čas používá časové pásmo UTC. K tomuto údaji je Bitcoin protokol celkem benevolentní. Může se stát, že časy v blocích nebudou monotónní.

3.5 Bits

Bits je atribut, který určuje, jak moc těžké bylo tento blok vytěžit. V proof-of-work algoritmu by mohl značit nejmenší počet počátečních nul pro zahašovanou hlavičku bloku. Nevýhoda je, že počtem počátečních nul můžete náročnost vytěžení bloku jenom dvakrát zvýšit nebo dvakrát snížit, a proto Bitcoin tento algoritmus mírně upravil.

Bitcoin protokol používá 4bajtové hodnoty, aby přesněji vyjádřil, jak těžké bude blok vytěžit. Z této hodnoty lze odvodit, jaký maximální SHA256 haš může blok mít, aby se mohl stát součástí blockchainu. Pro tento účel můžeme na haš nahlížet jako na 256bitové číslo.

Na atribut bits lze nahlížet jako na číslo s pohyblivou řádovou čárkou. Když číslo bits rozdělíme na koeficient a exponent, kde exponent je první bajt čísla bits a koeficient jsou následující tři bajty, můžeme vypočítat cílový haš podle rovnice:

$$hash = koeficient * 2^{8*exponent-3}$$

Tento údaj je v protokolu dynamický a mění se na základě rychlosti těžení bloků předchozích. Nová náročnost sítě, a tedy úprava čísla bits, se provádí každých 2016 bloků a upravuje se tak, aby v průměru byl nový blok vytěžen jednou za deset minut. Výpočet nového cílové haše se spočítá například takto: $new\_hash = old\_hash * (time/20160)$, kde $time$ je počet minut, za který bylo vytěženo posledních 2016 bloků, a konstanta 20160 značí počet minut, za který si síť přeje, aby trvalo těžení 2016 bloků. 20160 minut je v přepočtu 14dní a přesně tolik času by měla zabrat těžba 2016 bloků.

Z nového cílového haše lze spočítat novou hodnotu bits nalezením prvního nenulového bajtu. U tohoto kroku může dojít ke ztrátě přesnosti, ale i tak jde o výrazné zlepšení oproti původnímu proof-of-work algoritmu.

3.6 Přidávání nových bloků

V Bitcoinu přidávají do sítě bloky uživatelé, kterým se říká těžaři. Těžař má za úkol sestavit blok z transakcí, které uživatelé sítě chtějí zařadit do databáze, a pak najít nonce (jediná položka z hlavičky bloku, kterou může těžař volně měnit), pro níž při zahašování hlavičky bloku vznikne haš splňující upravený proof-of-work algoritmus 3.5. Za svoje snažení dostane odměnu v podobě poplatků u přidaných transakcí a nové Bitcoiny v hodnotě, kterou stanovuje protokol.

Složitost blockchainu se počítá z atributu bits u jednotlivých bloků a nejsložitější blockchain je ten s největším součtem jednotlivých složitostí.

Aby byl vytěžený blok sítí schválen a přidán všemi do blockchainu, musí splňovat následující kritéria:

- Blok, na který nový blok odkazuje, existuje a je validní.
- Čas v bloku větší než medián časů z předchozích jedenácti bloků a menší než dvě hodiny do budoucnosti.
- Atribut bits je správně nastaven s přihlédnutím na historii blockchainu.
- Proof-of-work algoritmus odpovídá náročnosti bloku.
- Všechny transakce jsou validní.
- Haš Merkleova stromu v hlavičce je stejný jako haš Merkleova stromu zkonstruovaného z transakcí.
- Transakce nezabírají více než jeden megabajt paměti.

Kapitola 4

Blockchain v Ethereu

Ethereum je peer-to-peer digitální měna, ve které lze vytvořit programovatelné objekty. Programovatelné objekty se spouští v turingovsky úplném programovacím jazyce postaveném nad virtuálním strojem (Ethereum Virtual Machine). Metody objektů lze volat pomocí transakcí nebo pomocí takzvaných zpráv, které si tyto programovatelné objekty mohou mezi sebou posílat. Dále budu tyto programovatelné objekty označovat jako *chytré kontrakty*.

Při psaní této kapitoly jsem vycházel z oficiálních dokumentů [27][25] popisujících kryptoměnu Ethereum.

V Ethereu je blockchain podobný blockchainu z kryptoměny Bitcoin, ale jednotlivé bloky mají více atributů a na transakce se nahlíží jiným způsobem.

Síť využívá proof-of-work algoritmu a v budoucnu má v plánu přejít na upravený proof-of-stake algoritmus. Tyto algoritmy jsou vysvětleny v kapitole o blockchainu v sekci Konsenzus 2.3.

Důležitou změnou je, že Ethereum používá pojmy *účet*, *stav*, upravené Merkleovy stromy 4.1 a že síť přidává bloky do blockchainu mnohem častěji, než tomu bylo u kryptoměny Bitcoin. Tohoto zrychlení je dosaženo pomocí modifikovaného protokolu GHOST [24] [26].

Pro většinu identifikátorů používá kryptoměna Ethereum hašovací funkci Keccak256. Ta převádí libovolný vstup na data o velikosti 256 bitů. Na proof-of-work algoritmus je použita speciální hašovací funkce Ethash.

4.1 Merkle Patricia tree

Ethereum používá upravené Merkleovy stromy, a to i pro jiné účely než Bitcoin. Těmto stromům se říká *Merkle Patricia tree* a jsou detailně popsané v oficiálním dokumentu [27] popisujícím kryptoměnu Ethereum. Pro naše potřeby nám bude stačit hrubá představa o této datové struktuře.

Jedná se o stromovou strukturu, která uchovává dvojice klíč – hodnota. Výhodou této struktury je možnost mazání, přidávání a upravování dvojic

bez nutnosti přepočítávání celého stromu. Kořen tohoto stromu se může hodit třeba jako otisk stavu celého blockchainu.

Ethereum tyto stromy používá v hlavičce bloku hned třikrát, a to pro:

- transakce,
- stvrzenky,
- stav.

4.2 Účet

V Ethereu existují dva druhy účtů, které se identifikují pomocí 20bajtové adresy. Účet obecně obsahuje množství vlastněného etheru, kód v jazyce EVM a data.

Účty se liší na základě existujícího kódu a dat u daného objektu. Pokud má účet kód, pak je chytrým kontraktem, a jeho kód se může zavolat pomocí zprávy od jiného chytrého kontraktu nebo pomocí transakce od jakéhokoliv uživatele. Pokud účet kód nemá, je to uživatelský účet, ze kterého uživatel může posílat měnu ether jiným účtům. Aby uživatel mohl posílat měnu ether, musí vlastnit privátní klíč k tomuto účtu.

4.2.1 Chytrý kontrakt

Chytrý kontrakt je programovatelný účet a lze jej vytvořit pomocí zaslání speciální transakce do sítě. Za kód a data, který kontrakt uchovává, se platí v podobě měny ether.

Uživatelé, kteří znají adresu chytrého kontraktu, mohou poslat transakci, a tím zavolat nějakou část kódu.

Pro zavolání kódu musí uživatel do transakce přidat atributy *gas* a *gasprice*. Tyto atributy slouží k zaplacení běhu kódu. *Gas* je jednotka, která se vyčerpává za každou spuštěnou instrukci. *Gasprice* si uživatel nastaví jako hodnotu, kterou je ochoten za každou jednotku atributu *gas* zaplatit. Tímto se vyřeší problém nekonečné smyčky, kterou lze v turingovsky úplném jazyce napsat.

Kód tedy vždy doběhne, jenom se nemusí dostat až na konec programu. Pokud se kód nedostane na konec programu, vrátí se blockchain do stavu před spuštěním kódu. Měna, kterou uživatel za spuštění kódu zaplatil, se nevrací.

4.2.2 Uživatelský účet

Uživatelským účtem se nazývá účet, který vlastní některý uživatel sítě. Tento účet obsahuje množství vlastněné měny a uživatel může z účtu posílat transakce. Uživatelské účty nejsou na rozdíl od chytrých kontaktů ovladatelné všemi uživateli Posílat transakce může jen uživatel, který k účtu vlastní privátní klíč.

4.3 Stav

Stavem se myslí všechny dvojice adresa a účet, kde účet může být uživatelským účtem nebo chytrým kontraktem. Tyto stavy nejsou uloženy přímo v blockchainu, ale uživatelé by je měli mít uloženy ve vhodné databázi na disku.

Každá transakce je přechod, který z aktuálního stavu vytvoří stav nový. Mezi těmito stavy pak lze lehce přepínat, a tím měnit kontext blockchainu.

4.4 Nezahrnuté bloky

Nalezení nového bloku trvá v průměru 20 vteřin. Proto se často stane, že dva těžaři naleznou nový blok ve stejný okamžik.

Těžař za blok, který není zahrnut v aktivní větvi blockchainu, dostane menší odměnu za jeho nalezení. To se stane ve situaci, kdy v některém z následujících bloků přidá těžař do bloku informace o tomto nezahrnutém bloku z minulosti.

Nezahrnuté bloky lze uvést v novém bloku, jen pokud je to potomek bloku z minulosti, a to až sedm bloků nazpět. Těžař nezahrnutého bloku i těžař, který informaci o tomto bloku přidal do aktuálního bloku, dostane zlomek odměny z klasické odměny za nalezení bloku.

4.5 Struktura bloku

Blok se stejně jako v kryptoměně Bitcoin skládá z hlavičky a transakcí. Hlavička je popsaná v tabulce 4.1.

velikost v bajtech	název	popis
32	haš předchozího bloku	Keccak256 haš rodičovského bloku
32	haš nezahrnutých bloků	Keccak256 haš listu nezahrnutých bloků
20	adresa těžaře	Adresa, na kterou se připsala odměna za vytěžení bloku
32	kořen stromu stavů	Keccak256 haš *Merkle Particia Tree* stavů po aplikaci všech transakcí
32	kořen stromu transakcí	Keccak256 haš *Merkle Particia Tree*, ve kterém se nacházejí všechny transakce
32	kořen stromu stvrzenek	Keccak256 haš kořene *Merkle Particia Tree*, který obsahuje stvrzenky transakcí
32	log	Bloom filtr zkonstruovaný z logů, které vygenerovaly transakce v tomto bloku
	náročnost	Práce, kterou museli těžaři v průměru vykonat, aby vytěžili tento blok
	výška	Počet bloků v této větvi blockchain
	gas limit	Limit možného použití *gas* v tomto bloku
	použitý gas	Množství použitého *gas* v tomto bloku
	čas	Čas začátku těžení bloku v unix epoch formátu
32	extra data	Data, ve kterých může být cokoliv od těžaře
32	mix haš	Haš, který v kombinaci s nonce dokazuje, že blok splňuje proof-of-work algoritmus
8	nonce	Libovolné číslo, které těžař může měnit, aby splnil proof-of-work algoritmus

Tabulka 4.1: Struktura hlavičky bloku v kryptoměně Ethereum. Volně přeloženo z [27]

4.5.1 Identifikátor bloku

V Ethereu se blok identifikuje podle hlavičky bloku za použití hašovací funkce Keccak256, která se použije na všechna data spojená za sebou z hlavičky bloku.

4.5.2 Haš nezahrnutých bloků

Bloky, které jsou potomky rodičů tohoto bloku, ale nejsou v hlavní větvi blockchainu.

Těžař tohoto bloku i těžař potomka za takový blok dostanou malou odměnu v podobě měny ether.

4.5.3 Kořen stromu stavů

Kořenem stromu stavů je Keccak haš kořene *Merkle Particia Tree* stavů po aplikaci všech transakcí v tomto bloku.

Ve stavu jsou uloženy všechny dvojice adresa – účet a díky *Merkle Particia Tree* struktuře slouží haš kořene tohoto stromu jako identifikátor stavu všech účtů v blockchainu.

4.5.4 Kořen stromu transakcí

Kořenem stromu transakcí je Keccak256 haš kořene *Merkle Particia Tree* všech transakcí obsažených v tomto bloku.

Ve stromu transakcí jsou uloženy všechny dvojice id transakce – transakce v *Merkle Particia Tree* struktuře.

4.5.5 Kořen stromu stvrzenek

Kořenem stromu stvrzenek je Keccak256 haš kořene *Merkle Particia Tree* všech stvrzenek vygenerovaných tímto blokem.

Stvrzenka obsahuje informace o přidané transakci. Tyto informace obsahují například logy z běhu kódu, které tato transakce mohla vyvolat, nebo použitý *gas*.

Ve stromu stvrzenek jsou všechny dvojice index – stvrzenka uloženy v *Merkle Particia Tree* struktuře, kde index je pozice transakce, která tuto stvrzenku vytvořila v seznamu transakcí tohoto bloku.

Data z tohoto stromu není potřeba ukládat, protože se dají vygenerovat přepnutím na předchozí blok a simulací přidání bloku. Z části jsou tato data indexována pomocí Bloom filtru 6.7.2 a tento Bloom filtr je uložen v hlavičce bloku.

4.5.6 Log

V rámci úspory místa na disku slouží log jako instance Bloom filtru 6.7.2 nastaveného na data vygenerovaná stvrzenkami bloku.

Díky tomuto atributu lze snadno hledat již zpracované transakce.

4.5.7 Náročnost

Oproti kryptoměně Bitcoin je zde náročnost (množství práce pro nalezení správné nonce) uložena přímo v hlavičce bloku a nemusí se přepočítávat z atributu bits 3.5. Další změnou je dynamičtější upravování náročnosti těžení. V Bitcoinu se náročnost upravuje každých 2016 bloků (v průměru každých 14 dní), ale v Ethereu se náročnost mění pro každý nový blok. Náročnost lze spočítat z atributů náročnost a čas předchozího bloku.

4.5.8 Gas limit

Gas limit je číslo, které určuje, kolik maximálně mohou transakce v tomto bloku použít jednotek *gas*.

4.5.9 Čas

Atribut čas je údaj o čase, ve kterém začal těžař tento blok těžit v unix epoch formátu. Celá síť jako čas používá časové pásmo UTC.

Tento atribut musí být větší než čas předchozího bloku, ale nesmí být větší než 15 minut do budoucnosti.

4.5.10 Mix haš a nonce

Důležitým rozdílem Etherea oproti Bitcoinu je, že používá upravenou hašovací funkci, která se snaží být odolná proti specializovanému hardwaru určenému přímo pro těžení nových bloků. Tím je těžba měny přístupnější pro veřejnost.

Pro nás to znamená, že identifikátor bloku neslouží jako kontrola splnění proof-of-work algoritmu, jako tomu bylo v případě kryptoměny Bitcoin. Pro kontrolu proof-of-work a těžení etheru používá hašovací funkci Ethash [8].

Tato hašovací funkce má velkou paměťovou složitost. Pro těžení je potřeba mít v paměti mnoho dat, která rostou lineárně s množstvím hlaviček bloků v blockchainu. Těžař musí mít tato data v paměti a za pomocí změny nonce hledat hlavičku bloku, která splňuje náročnost.

Pro ověření správnosti proof-of-work algoritmu již není potřeba mít všechna data, ale stačí jen určitá část, kterou těžař přidal do atributu mix haš.

4.6 Přidávání nových bloků

Přidávání bloků do blockchainu je podobné jako u kryptoměny Bitcoin. Do blockchainu přidávají bloky těžaři, kteří hledají takový blok, aby splňoval proof-of-work algoritmus. Hlavní změnou je použití jiné hašovací funkce, kterou se síť snaží zpřístupnit těžení většímu počtu uživatelů, a existence chytrých kontraktů.

Těžař vezme nezpracované transakce v síti a pokusí se z nich sestavit blok. Za vytěžený blok získá odměnu v podobě etheru a poplatků za přidané transakce. Poplatky za transakce se spočítají z atributů *gas* a *gasprice* u jednotlivých transakcí.

Aby byl vytěžený blok sítí schválen a přidán všemi do blockchainu, musí splňovat následující kritéria:

- Blok, na který nový blok odkazuje, existuje a je validní.
- Čas v bloku je větší než čas v bloku předchozím a je menší než 15 minut do budoucnosti.
- Číslo bloku je správné, náročnost odpovídá historii, nezahrnuté bloky jsou validní a *gas* celého bloku je menší než *gas limit*.
- Proof-of-work algoritmus odpovídá náročnosti bloku.
- Všechny transakce jsou validní.
- Haš kořene *Merkle Particia Tree* všech stavů je stejný jako haš kořene *Merkle Particia Tree* po přidání všech transakcí do blockchainu.

4.7 Decentralizované aplikace

Pro vytváření aplikací se v Ethereu používají chytré kontrakty. Jsou to programovatelné objekty, jejichž kód můžou uživatelé spouštět pomocí transakcí. Pro napsání decentralizované aplikace existuje mnoho programovacích jazyků, jejichž kompilátor dokáže vygenerovat bajtkód spustitelný EVM. Jeden z takových jazyků je Solidity [23], ve kterém lze napsat například aplikace, která uživatele pozdraví. Ukázku jsem našel na webových stránkách [7] Etherea.

```
contract Mortal {
    /* Define variable owner of the type address */
    address owner;

    /* This function is executed at initialization
    and sets the owner of the contract */
    function Mortal() { owner = msg.sender; }

    /* Function to recover the funds on the contract */
    function kill() { if (msg.sender == owner)
                          selfdestruct(owner);
                    }
}

contract Greeter is Mortal {
    /* Define variable greeting of the type string */
    string greeting;

    /* This runs when the contract is executed */
    function Greeter(string _greeting) public {
        greeting = _greeting;
    }

    /* Main function */
    function greet() constant returns (string) {
        return greeting;
    }
}
```

Listing 1: Greeter

V principu uživatel tento kód přeloží a s argumentem pro konstruktor třídy *Greeter* pošle v transakci do sítě. K této transakci přidá *gas* a *gasprice*, čímž zaplatí za existenci tohoto chytrého kontraktu v blockchainu. Uživatel pak může volat metodu *greet* tohoto chytrého kontraktu. Všichni peerové tuto metodu spustí a budou pozdraveni řetězcem, který byl nastaven při inicializaci

této třídy. Díky tomu, že třída zdědila metody třídy *Mortal*, může uživatel tento chytrý kontrakt zase z blockchainu vymazat, a to pomocí metody *kill*.

Kapitola 5

Jiná použití

Toto odvětví IT je velmi nové, dynamické a nápady na nové aplikace přicházejí každý den. Pokusím se zaměřit na starší projekty postavené nad blockchainem kryptoměny Bitcoin nebo nad kryptoměnou Ethereum. Vynechám projekty, které používají svůj vlastní blockchain, nebo blockchain jiných než výše uvedených kryptoměn.

V Bitcoinu jsou transakce zpětně dohledatelné a uživatelé si je mezi sebou posílají. Lehce se dá zjistit, že transakce, kterou mi někdo poslal, vznikla z transakcí jiných. A historie předávání bitcoinů se dá dohledat. Dokonce se v Bitcoinu dají k transakcím přidat nějaká data. Tato data většinou označují nového majitele měny, ale lze k nim připojit i nějakou další informaci. Aplikace postavené nad kryptoměnou Bitcoin jsou vybudované právě nad těmito přidanými daty.

Nad kryptoměnou Ethereum se přirozeně vytváří mnoho aplikací díky chytrým kontraktům. Na využívání aplikací v této síti potřebuje uživatel vlastnit ether, aby mohl posílat transakce a spouštět kód chytrých kontraktů.

5.1 Obarvování mincí

Tento koncept se využívá převážně v kryptoměně Bitcoin. Když do transakce přidáme nějakou informaci (například číslo), můžeme tuto transakci prohlásit za obarvenou. Lze dohledávat, které transakce z této obarvené transakce vznikly, a dát jim nějaký speciální účel. Hypotetickým příkladem by mohlo být například kino. Majitel kina, který prozradí svoji veřejnou adresu, pošle zájemcům o určité představení část obarvené mince. Těchto částí může poslat tolik, kolik je sedadel v kině, a diváci se pak v kině prokážou vlastnictvím obarvené mince.

5.2 Namecoin

Namecoin [19] slouží pro zaznamenávání dvojic klíč – hodnota, lze nad tím například vybudovat decentralizovaný DNS server. Upravený software sleduje speciální transakce, které se mohou v blockchainu vyskytovat (například obarvené nějakou informací v datové složce transakce) s ještě dodatečnými informacemi, jako je klíč a hodnota. Tyto informace si uloží do vlastní databáze, které se pak uživatel lehce zeptá například na přeložení jména domény na IP adresu, pokud takový blockchain vlastní. Tento nápad byl nejdříve zrealizovaný nad kryptoměnou Bitcoin, ale dnes má svůj vlastní blockchain a měnu.

5.3 EtherTweet

Jeden ze zajímavých nápadů je decentralizovaný Twitter [9], kde je ale přirozená nevýhoda v nutnosti použití ethera za posílání nových příspěvků do sociální sítě.

Kapitola 6

Návrh blockchainu pro FITCOIN

V aplikaci použiji zjednodušenou verzi formátu bitcoinového bloku se zásadními rozdíly, které jsou popsány ve následující sekci. Jako konsenzus používám proof-of-work algoritmus a rychlost těžení nového bloku je v průměru deset minut.

Na ukládání dat o blockchainu používám LevelDB [13] databázi a na ukládání transakcí soubory na disku. Z velké části je tento návrh převzat z Bitcoin Core [3] aplikace.

Pro většinu hašovaní používám kryptografickou hašovací funkci SHA256.

6.1 Struktura bloku

Blok je objekt, který má hlavičku 6.1 a sérii transakcí.

Hlavička bloku obsahuje informace o bloku a slouží jako unikátní identifikátor. Hlavička má pevnou velikost 77 bajtů.

6.2 Identifikátor bloku

Identifikátorem bloku je haš SHA256 aplikovaný dvakrát na data z hlavičky bloku spojená za sebe.

6.3 Merkleův strom

Ve FITCOINu je Merkleův strom stejný jako v kryptoměně Bitcoin. Do listů tohoto stromu jsou přidány haše transakcí v bloku. Pokud počet transakcí v bloku není mocninou dvou, pak haš poslední transakce se zkopíruje tolikrát, aby počet listů odpovídal právě nejbližší mocnině čísla dva.

Z těchto listů je pak zkonstruován Merkleův strom, jehož kořen se používá v hlavičce bloku pro kontrolu integrity dat. Pro podporu odlehčeného klienta,

velikost v bajtech	název	popis
4	verze	Verze protokolu, ve které byl blok vytěžen
32	identifikátor předchozí hlavičky	SHA256(SHA256()) haš předchozího bloku
32	haš Merkleova stromu	SHA256(SHA256()) haš Merkleova stromu
4	čas	Čas vytěžení bloku v unix epoch formátu. Vyjadřuje čas, kdy těžař začal těžit daný blok
1	bits	Číslo, které udává náročnost nalezení správné nonce pro proof-of-work algoritmus
4	nonce	Libovolné číslo, které těžař může měnit, aby splnil proof-of-work algoritmus

Tabulka 6.1: Struktura hlavičky bloku ve FITCOINu

kterým může být například android peněženka, používám částečný Merkleův strom, který popisuji v sekci Odlehčený klient 6.7.

6.4 Čas

V konceptu by měl čas označovat chvíli, kdy těžař začal s těžbou bloku. Celá síť jako čas používá časové pásmo UTC a unix epoch formát.

Aby byl blok validní, musí mít pozdější čas než blok předchozí, ale nesmí být větší než 10 minut do budoucnosti. Tímto omezením se trochu vzdalujeme od návrhu bloku v kryptoměně Bitcoin, kde čas může být až dvě hodiny do budoucnosti a nemusí být monotonní. Můj návrh je v tomto ohledu blíž ke kryptoměně Ethereum.

6.5 Bits

Zásadním rozdílem oproti Bitcoinu je atribut bits. Bitcoin používá 4bajtové číslo, ze kterého lze jednoduše spočítat množinu hašů splňujících proof-of-work algoritmus. Tento postup je popsán v kapitole Blockchain v Bitcoinu 3.5.

Já v aplikaci přistupuji k atributu bits jednodušším způsobem a můj návrh je stejný jako v originálním proof-of-work algoritmu popsaným v sekci Konsenzus 2.3.

Pro naši kryptoměnu označuje tento atribut minimální počet počátečních bitových nul, které musí obsahovat haš hlavičky bloku. Tímto ústupkem můžeme práci na nalezení bloku pouze zvětšovat a zmenšovat v řádu mocniny dvou.

V síti se toto číslo mění tak, aby se nalezl nový blok v čase mezi pěti až dvaceti minutami. Změna bits se spočítá z času vytěžení posledních dvou bloků a z atributu bits posledního bloku.

Výpočet o novém bits lze rozdělit do tří kategorií, a to podle časového rozdílu z předchozích dvou bloků. Pokud je rozdíl menší než pět minut, zvyš číslo bits o jedna. Pokud je rozdíl větší než dvacet minut, sniž číslo bits o jedna. Pokud je rozdíl v rozmezí pěti a dvaceti minutami, bits neupravuj.

6.6 Transakce

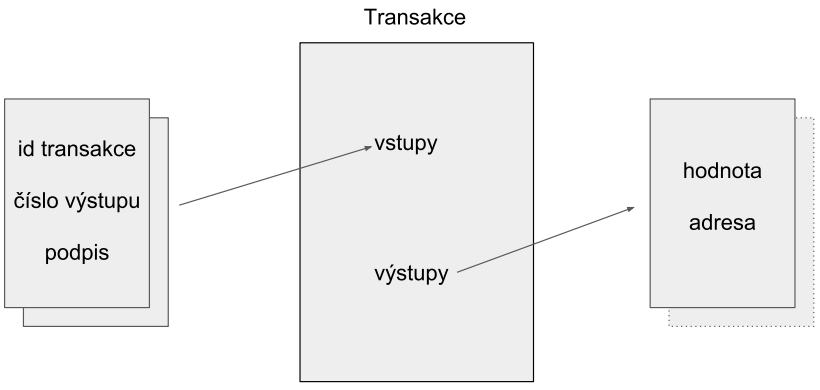
V blockchain databázi budeme ukládat transakce uživatelů krytoměny FITCOIN. Transakce jsou podrobně popsány v bakalářské práci Jana Tománka. Pro návrh blockchainu je potřeba chápat, jak taková transakce vypadá a co představuje.

Transakce se skládají ze série vstupů a série výstupů.

Výstupem je dvojice adresa a hodnota, kde adresa je zahašovaný veřejný klíč uživatele, který tento bude vlastnit, a hodnota znamená, kolik fitcoinů se k tomuto výstupu váže. Tedy znamená to, komu a kolik fitcoinů posílám.

Vstupem je id transakce, číslo výstupu a podpis, kde id transakce je unikátní identifikátor transakce v rámci celého blockchainu, číslo výstupu označuje, který výstup této transakce uživatel chce utratit, a podpis slouží pro ověření vlastnictví tohoto výstupu.

Pro návrh blockchainu to znamená, že umíme snadno ověřit, že uživatel je vlastníkem daného výstupu. Mým úkolem je ověřování, že daný výstup nebyl v historii již utracen.



Obrázek 6.1: Transakce ve FITCOINu.

6.7 Odlehčený klient

Protože jedna z bakalářských prací v projektu FITCOIN je android peněženka, bylo potřeba v aplikaci vytvořit podporu pro odlehčeného klienta.

Tento odlehčený klient dostává od sítě upravený blockchain, který zabírá méně místa na disku, a tím šetří paměť. Paměťové zlepšení je zrealizováno pomocí částečného Merkleova stromu, v němž odlehčený klient nemá uloženy všechny transakce, ale pouze nějakou nadmnožinu transakcí, která odlehčeného klienta zajímá. Nadmnožinu transakcí zajišťuje Bloom filtr.

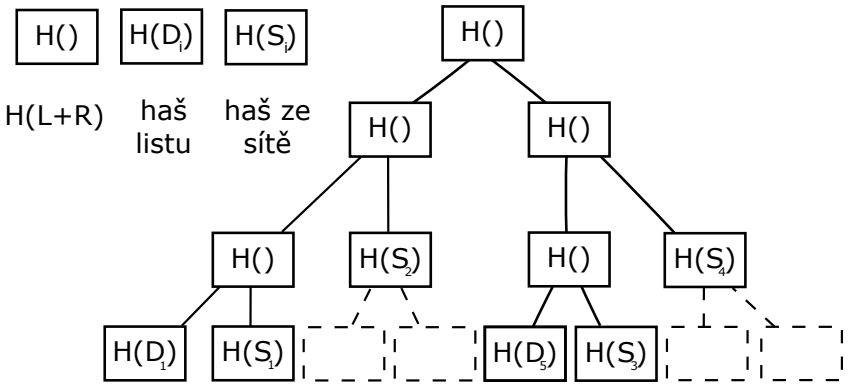
V aplikaci jsem tuto část naprogramoval stejně, jako bylo navrhnuto v BIP-0037 [11].

6.7.1 Částečný Merkleův strom

Tento strom vychází z návrhu Merkleova stromu, ale pro jeho zkonstruování není potřeba znát všechny listy.

Slouží pro uchovávání nějaké podmnožiny transakcí z Merkleova stromu, výhodou je paměťová úspora. Aplikace umí takové stromy vytvářet a pro testování je umí i zkonstruovat.

Částečný Merkleův strom je na následujícím obrázku. Strom lze zkonstruovat z dvou polí dat. V prvním poli jsou uloženy jedničky a nuly, v druhém haše některých uzlů. První pole popisuje průchod prohledávání do hloubky tímto stromem a ve druhém jsou haše potřebné ke konstrukci částečného Merkleova stromu.



Obrázek 6.2: Částečný Merkleův strom.

6.7.2 Bloom filtr

Bloom filtr je paměťově efektivní datová struktura, která se používá na testování členství prvků v nějaké množině. Struktura dosahuje velké komprese dat na úkor falešně pozitivních výsledků.

Jinými slovy jsem schopen lehce ověřit, zda transakce, na kterou je Bloom filtr nastaven, je transakcí, kterou si odlehčený klient přeje zahrnout do částečného Merkleova stromu. Protože má Bloom filtr falešně pozitivní výsledky, dostane odlehčený klient nadmnožinu výsledků. Získá tím hlavně anonymitu, protože já nemohu z jistotou prohlásit, že ho zajímá právě určitá transakce, a tím například zjistit jeho veřejný klíč.

Bloom filtr je součástí aplikace a zdrojové kódy pro implementaci mi poskytl Minh Trieu Quang, který dělal bakalářskou práci na téma FITCOIN: *an Android light wallet*.

6.8 Přidávání nových bloků

Pro přidání nových bloků do sítě jsem nadefinoval kritéria z velké části inspirovaná kryptoměnou Bitcoin. Aby byl vytěžený blok sítí schválen, musí splňovat následující omezení:

- Blok, na který nový blok odkazuje, existuje a je validní.
- Atribut bits je nastavený na správnou hodnotu s přihlédnutím k předchozím blokům.
- Hlavička bloku splňuje proof-of-work algoritmus.
- Čas bloku je větší než čas předchozího bloku a menší než deset minut do budoucnosti.
- Blok nemá více než padesát transakcí.
- Blok má alespoň jednu transakci a první transakce je těžaře.
- Všechny transakce v bloku jsou validní.
- Haš Merkleova stromu v hlavičce je stejný jako haš Merkleova stromu zkonstruovaného z transakcí.

Prvních pět omezení lze ověřit jen ze znalosti hlavičky bloku a dvou předchozích hlaviček bloku, ale omezení o validitě transakcí je složitější. Pro takové ověření je potřeba znát historii všech transakcí a určit, zda ověřovaná transakce ještě nebyla utracena.

V návrhu jsem se vydal stejnou cestou jako Bitcoin Core [3], a to přidáním nové databáze, která si pamatuje všechny neutracené výstupy transakcí v aktuální větvi. Tuto databázi budu označovat UTXO.

6.9 Datová vrstva

Pro ukládání blockchainu používám kombinaci LevelDB [13] databáze a souborů na disku.

LevelDB databáze je jednoduchá databáze na ukládání dvojic klíč – hodnota, a to přímo na disk.

V aplikaci se takové databáze používají dvě. První na ukládání upravených hlaviček bloků 6.9.1 a druhá na ukládání UTXO databáze.

6.9.1 Databáze bloků

V této databázi jsou uloženy všechny známé bloky. Slouží pro vyhledávání transakcí spojené s blokem a pro nahrání informací o blockchainu do paměti aplikace.

V této databázi jsou uloženy dvojice popsané v tabulce níže.

klíč	hodnota
identifikátor bloku	hlavička bloku, výška bloku v blockchainu, pozice transakcí a pozice utracených výstupů transakcí
„l“	poslední použitý soubor

Tabulka 6.2: Uložené hodnoty v databázi.

Výškou bloku v blockchainu se myslí, kolikátý je to blok od genesis bloku. Pozice transakcí určuje, kde se nachází transakce daného bloku. Je to dvojice čísel, kde první číslo odpovídá názvu souboru s prefixem „block“ a druhé značí začátek uložených transakcí. Pozice utracených výstupů transakcí určuje, kde se nacházejí utracené výstupy transakcí daného bloku. Je to dvojice čísel, kde první číslo odpovídá názvu souboru s prefixem „rev“ a druhé značí začátek uložených utracených výstupů transakcí. Seznam utracených výstupů transakcí je potřebný pro přepínání kontextu blockchainu 6.10.

6.9.2 Databáze UTXO

V této databázi jsou uloženy všechny neutracené výstupy transakcí vzhledem k poloze aktivní větve 6.10.1.

Do databáze se přidávají nové neutracené výstupy transakce, jež vzniknou přidáním nového bloku do aktivní větve, a mažou se takové výstupy transakcí, které tyto nové transakce použily jako vstupy.

Díky této databázi lze rychle ověřit, zda transakce, kterou se snaží uživatel utratit, ještě nebyla utracena.

6.10 Přepínání kontextu

Aplikace mění databázi UTXO na základě toho, v jakém bloku se v rámci blockchainu nachází. V principu může docházet ke dvěma druhům přepínání kontextu. V prvním se rozšiřuje aktivní větev a ve druhém se přepíná na novou aktivní větev. Pro pochopení je potřeba rozlišovat mezi pojmy *blockchain* a *aktivní větev*.

6.10.1 Aktivní větev

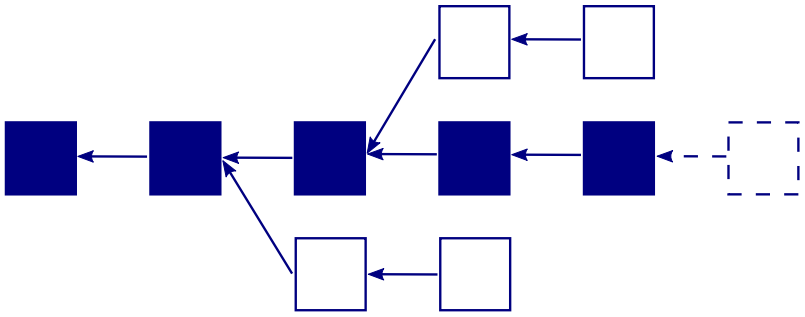
Blockchainem se rozumí celý blockchain, a to ve smyslu stromové datové struktury, kde bloky splňují pravidla popsaná v sekci Přidávání nových bloků 6.8. Aktivní větví je část blockchainu, která tvoří řetěz s největším součtem prací jednotlivých bloků.

6.10.2 Rozšíření aktivní větve

K tomuto stavu by mělo docházet mnohem častěji, a to ve chvíli, kdy nějaký těžař nalezne nový blok rozšiřující vrchol aktivní větve blockchainu.

V tomto případě aplikace zkontroluje, zda vstupy transakcí v tomto bloku ještě nebyly utraceny, a to za pomocí UTXO databáze. Potom aplikace zapíše do souboru všechny utracené vstupy transakcí, které tento blok použil pro vytvoření svých transakcí, a vrcholem aktivní větve se označí nově přidaný blok.

Tento stav je vyobrazen na následujícím obrázku.



Obrázek 6.3: Rozšíření aktivní větve.

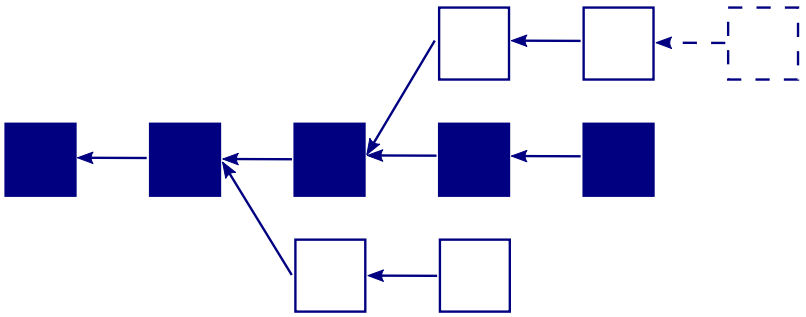
6.10.3 Přepnutí aktivní větve

K tomuto stavu dochází, pokud se do blockchainu přidá nový blok, který nerozšiřuje aktivní větev, ale jeho celková práce až ke genesis bloku je větší než celková práce aktivní větve.

V takovém případě aplikace najde společného předka vrcholu aktivní větve a nově přidaného bloku a odpojí vrchol aktivní větve, dokud se nedostane ke společnému předku. Následně přidá nové bloky z větve, jehož vrcholem je nově přidaný blok.

Novou operací je odpojování vrcholu. V této operaci se musí odebrat záznamy z UTXO databáze, jež odpojovaný blok vytvořil, a přidat do UTXO databáze záznamy, které tento blok při přidávání smazal.

Tento stav je vyobrazen na následujícím obrázku.



Obrázek 6.4: Přepnutí aktivní větve.

6.11 Těžba

Pro těžení nových bloků v aplikaci slouží jednoduchá funkce, která u bloku mění nonce a kontroluje, zda pro tuto nonce haš hlavičky splňuje proof-of-work algoritmus. Pokud se všechny možné nonce vyčerpají a žádná nesplňuje proof-of-work algoritmus, přidá jednu vteřinu k atributu čas u tohoto bloku a zkusí naleznout nonce znovu.

Pro složení nového bloku používá těžař pole všech transakcí, které chtějí uživatelé potvrdit. Vybere jich maximálně 49, které přidá do nového bloku společně s jednou transakcí pro sebe.

Kapitola 7

Testování

V průběhu vývoje jsem používal framework *Google Test* [10] na *unit testing* a přirozeně s přibývajícím kódem jsem testoval funkčnosti nových metod. Všechny testy jsou na flash disku, který je přiložený k této bakalářské práci. Testy by se daly shrnout do následujících kategorií:

- počítání správné proof-of-work,
- upravování náročnosti na základě historie,
- vytváření nových bloků těžařem,
- těžba nových bloků,
- testování implementace Merkleových stromů,
- testování částečných Merkleových stromů pro odlehčeného klienta,
- přidávání nových bloků,
- získávání transakcí z databáze,
- získávání neutracených transakcí pro adresu uživatele,
- přepínaní na větev s větší odvedenou prací,
- utracení stejné transakce vícekrát.

Nejzásadnější jsou testy demonstrující funkčnost třídy *Blockchain*, ve kterých nejdříve testuji, zda se po přidání nového bloku do blockchainu tento blok stane vrcholem blockchainu. V dalších testech testuji, zda se blockchain přepíná do správného stavu, pokud přidám větev, která má větší součet prací než větev aktuální. Třída je schopná odhalit i opětovné utracení transakce v rámci aktivní větve a do takové větve se potom nepřepíná. Třída *Blockchain* v těchto testech kontroluje všechna pravidla, která jsou nadefinovaná pro naši

kryptoměnu a je společně se třídou *Miner* připravena na integraci do výsledné aplikace.

Aplikaci jsem nemohl otestovat v plném provozu, protože hlavní aplikace byla vyvíjena třemi studenty a student, který pracuje na síťové komunikaci, nestihl vypracovat finální aplikaci. Nemohli jsme proto otestovat funkčnost v rámci peer-to-peer sítě a posílání transakcí mezi uživateli.

Závěr

V práci jsem zanalyzoval technologii blockchain v kryptoměnách Bitcoin a Ethereum. V těchto kryptoměnách se mi podařilo prozkoumat některé aplikace nesouvisející s platbami.

V praktické části jsem navrhl blockchain pro kryptoměnu FITCOIN a tento návrh jsem zrealizoval v jazyce C++. Blockchain umí přijímat nové bloky, měnit svůj stav a kontrolovat historii transakcí. Kód jsem v průběhu implementace testoval a zdokumentoval. Zdrojové kódy, dokumentace a testovací soubory přikládám na flash disku, který je součástí této práce.

Bibliografie

[1] Andreas M. Antonopoulos. *Mastering Bitcoin, 2nd Edition*. "O'Reilly Media, Inc.", 2017.

[2] Adam Back. *Hashcash*. [navštíveno 13. 05. 2018]. URL: http://www.hashcash.org/.

[3] *Bitcoin Core*. [navštíveno 25. 04. 2018]. 2018. URL: https://bitcoincore.org/.

[4] *Bitcoin Developer Reference*. [Online; navštíveno 19. 04. 2018]. URL: https://bitcoin.org/en/developer-reference.

[5] *BlockChain technology: beyond bitcoin. Applied Innovation Review*. [navštíveno 5. 05. 2018]. URL: http://scet.berkeley.edu/wp-content/uploads/AIR-2016-Final-version-Int.pdf.

[6] Vitalik Buterin. *On Stake - Ethereum Blog*. 2018. URL: https://blog.ethereum.org/2014/07/05/stake/.

[7] *Create a Hello World Contract in ethereum*. [navštíveno 11. 05. 2018]. 2018. URL: https://www.ethereum.org/greeter.

[8] *Ethash*. [navštíveno 11. 05. 2018]. 2018. URL: https://github.com/ethereum/wiki/wiki/Ethash.

[9] *EtherTweet - Microblogging on the Ethereum Blockchain*. [navštíveno 28. 04. 2018]. 2018. URL: http://ethertweet.net/.

[10] *Google Test*. [navštíveno 25. 04. 2018]. 2018. URL: https://github.com/google/googletest.

[11] Mike Hearn a Matt Corallo. *Connection Bloom filtering*. [navštíveno 25. 04. 2018]. 2012. URL: https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki.

[12] *History of bitcoin — Wikipedie: The Free Encyclopedia*. [navštíveno 19. 04. 2018]. URL: https://en.wikipedia.org/w/index.php?title=History_of_bitcoin&oldid=837176283.

[13] *LevelDB*. [navštíveno 25. 04. 2018]. 2018. URL: https://github.com/google/leveldb.

[14] Róbert Lórencz. *Hašovací funkce, MD5, SHA-x, HMAC*. Nepublikovaná přednáška.

[15] Alfred J. Menezes, Scott A. Vanstone a Paul C. Van Oorschot. *Handbook of Applied Cryptography*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 1996, s. 323–325. ISBN: 0849385237.

[16] Ralph Merkle. *Method of providing digital signatures*. 1979.

[17] Silvio Micali. *Tree-based certificate revocation system*. 1995.

[18] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system,"* [Online; navštíveno 19. 04. 2018]. URL: http://bitcoin.org/bitcoin.pdf.

[19] *Namecoin*. [navštíveno 28. 04. 2018]. 2018. URL: https://namecoin.org/.

[20] Hitoshi Okada, Shigeichiro Yamasaki a Vanessa Bracamonte. „Proposed classification of blockchains based on authority and incentive dimensions". In: *Advanced Communication Technology (ICACT), 2017 19th International Conference on*. IEEE. 2017, s. 593–597.

[21] QuantumMechanic. *Proof of stake instead of proof of work*. 2018. URL: https://bitcointalk.org/index.php?topic=27787.0.

[22] R. Schollmeier. „A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications". In: *Proceedings First International Conference on Peer-to-Peer Computing*. Srp. 2001, s. 101–102. DOI: 10.1109/P2P.2001.990434.

[23] *Solidity*. [navštíveno 11. 05. 2018]. 2018. URL: https://solidity.readthedocs.io/en/v0.4.23/.

[24] Yonatan Sompolinsky a Aviv Zohar. „Secure high-rate transaction processing in bitcoin". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2015, s. 507–527.

[25] *White Paper*. [navštíveno 11. 05. 2018]. 2018. URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[26] *White Paper*. [navštíveno 11. 05. 2018]. 2018. URL: https://github.com/ethereum/wiki/wiki/White-Paper#modified-ghost-implementation.

[27] Gavin Wood. *Ethereum: A secure decentralised generalised transaction ledger*. [navštíveno 11. 05. 2018]. 2018. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

Dodatek A

Seznam použitých zkratek

SHA Secure Hash Algorithm

UTC Coordinated Universal Time

BIP Bitcoin Improvement Proposals

EVM Ethereum Virtual Machine

GHOST Greedy Heaviest Observed Subtree

Dodatek B

Obsah přiloženého flash disku

- `readme.txt` ..... stručný popis obsahu flash disku
- `fitcoin`
  - `src` ..... zdrojové kódy implementace
  - `tests` ..... zdrojové kódy testů
  - `doc` ..... dokumentace
  - `Readme.md` ..... instrukce k instalaci
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text`
  - `thesis.pdf` ..... text práce ve formátu PDF