FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Crypto pro FITCoin
Student:	Ihor Salov
Vedoucí:	Mgr. Jan Starý, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

1. Popište mechanismus podpisu založeného na asymetrické kryptografii.
2. Proveďte rešerši knihoven v jazyce C, které takový podpis implementují.
3. Navrhněte způsob, jak zavést podpis jako autorizaci útraty v měně FITCoin.
4. S pomocí zvolené crypto knihovny takové podpisy implementujte.
5. Implementaci proveďte v jazyce C.
6. Implementaci řádně otestujte.
7. Metody testování zdokumentujte.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 25. února 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Crypto pro FITCoin

Ihor Salov

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

26. června 2021

Poděkování

Chtěl bych poděkovat Mgr. Janovi Starému, Ph.D. za vedení mé bakalářské práce, pomoc a rady při jejím zpracování. Děkuji své rodině za to, že mi věřila a trpělivě čekala na okamžik, kdy obdržím diplom. Mé poděkováni patři též Evgenii Golubtsové za morální podporu během vypracovaní bakalářské práce a Bc. Vojtěchu Kulovanému za pomoc při její gramatické kontrole.

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, avšak pouze k nevýdělečným účelum. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 26. června 2021 . . . . . .

Odkaz na tuto práci

Salov, Ihor. *Crypto pro FITCoin*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato bakalářská práce popisuje principy asymetrické kryptografie a použití mechanismu digitálního podpisu k zajištění integrity dat v technologii blockchain. Získané znalosti jsou poté použity k implementaci mechanismu pro autorizaci utrácení prostředků v kryptoměně FITCoin.

Klíčová slova blockchain, FITCoin, autorizace utrácení, asymetrická kryptografie, digitální podpis, kryptoměna, ECDSA, eliptické křivky

Abstract

This bachelor thesis describes the principles of asymmetric cryptography and the usage of digital signature mechanism to provide integrity of blockchain technology. The knowledge gained during this research is then used to implement a mechanism for authorizing coin spending in the FITCoin cryptocurrency.

Keywords blockchain, FITCoin, coin spending, asymmetric cryptography, public-key cryptography, digital signature, cryptocurrency, ECDSA, elliptic curves

Obsah

Seznam obrázků

Úvod

Od příchodu Bitcoinu v roce 2009 technologie blockchain s každým rokem získává větší a větší popularitu. Omezený počet mincí a vysoká volatilita kryptoměn přitahují velkou pozornost investorů po celém světě. Podle agentury Reuters [1] v únoru 2021 kapitalizace Bitcoinu dosáhla 1 biliónu dolarů a denní obrat transakcí činí několik miliard.

Na rozdíl od bankovního systému převod peněz mezi dvěma stranami probíhá bez zprostředkovatelů a všechny informace o transakcích jsou uloženy veřejně v nešifrované podobě. Vzniká otázka, jak si mohou být uživatelé jisti, že jejich mince jsou v bezpečí.

Bezpečnost blockchainu je založena na kryptografických algoritmech. Přítomnost hash řetězců a digitálních podpisů téměř úplně vylučuje jakékoli zpětné manipulace. Integritu dat uložených v blockchainu navíc nepotvrzuje jenom jedna centrální autorita, ale všichni účastníci sítě současně.

FITCoin je kryptoměna, kterou vyvíjeli studenti Fakulty informačních technologií ČVUT v Praze. Při návrhu byl hlavní důraz kladen na to, aby výsledná struktura byla co nejjednodušší.

Tato práce vychází z existujícího projektu, vytvářeného v rámci následujících bakalářských prací:

- FITCOIN: struktura peněženky (Lukáš Dang)
- Peněženka pro Fitcoin (Jiří Havrusevič)
- Správa blockchainu pro Fitcoin (Andrea Zábojníková)

Smyslem mojí bakalářské práce je prozkoumat, jak se digitální podpis používá v technologii blockchain, a na základě získaných znalostí rozšířit existující řešení a implementovat autorizaci útraty v měně FITCoin.

Kapitola 1

Cíl práce

Hlavním cílem této práce je seznámit se se základy asymetrické kryptografie, mechanismem digitálního podpisu, který je na ní založený, a použít získané znalosti pro implementaci autorizace útraty v měně FITCoin.

V rámci dílčích kroků provedu rešerši existujících kryptografických knihoven, které implementují digitální podpis. Dále navrhnu způsob, jak zavést digitální podpis jako autorizaci útraty. V praktické části práce se zaměřím na implementaci samotného algoritmu. Zdrojový kód bude napsán v jazyce C. Jedním z cílů je také otestovat výslednou funkcionalitu a zdokumentovat použité metody.

Kapitola 2

Kryptografie

Jak napovídá název, kryptografie má zásadní význam pro kryptoměny. Tato kapitola obsahuje potřebnou teorii o kryptografických mechanismech, na nichž jsou tyto měny postavené.

2.1 Asymetrická kryptografie

Asymetrická kryptografie (nebo také kryptografie s veřejným klíčem) je skupina kryptografických metod, kde klíč, který je používán pro šifrování (veřejný) je odlišný od klíče, který je používán pro dešifrování (soukromý). [2]

Síla asymetrického kryptosystému je postavena na takzvaných jednosměrných funkcích s padacími dvířky. Jednosměrná funkce s padacími dvířky je invertibilní funkce, kterou lze snadno vypočítat, ale je jí těžké invertovat, pokud nemáme informaci navíc. [5] Takové funkce jsou postavené na matematických problémech, mezi které patří faktorizace celých čísel, problém diskrétního logaritmu (DLP) a nalezení diskrétního logaritmu náhodného bodu eliptické křivky (ECDLP). [4]

2.1.1 Faktorizace celých čísel

Podle základní věty aritmetiky lze každé přirozené číslo větší než 1 jednoznačně rozložit na součin prvočísel. Kryptografické systémy založené na problému faktorizace využívají toho, že pro dostatečně velká čísla stále není známý žádný efektivní (nekvantový) faktorizační algoritmus.

2.1.2 Problém diskrétního logaritmu

V [5] se uvádí následující definice: nechť G je grupa, jejíž grupovou operaci označíme symbolem *. Problém diskrétního logaritmu (DLP) je pro dva dané

prvky $g \in G$, $h \in G$ najít celé číslo x takové, že:

$$g^x = \underbrace{g * g * g * \cdots * g}_{x \text{ krát}} = h \tag{2.1}$$

2.2 Eliptické křivky

V [5] je eliptická křivka E definovaná jako množina všech řešení Weierstrassovy rovnice:

$$E : y^2 = x^3 + Ax + B, \tag{2.2}$$

kde konstanty A a B musí splňovat

$$4A^3 + 27B^2 \neq 0 \tag{2.3}$$

Na eliptické křivce definujeme grupovou operaci, kterou budeme značit jako součet. Všechny body křivky spolu s touto operací tvoří komutativní (abelovskou) grupu. Výsledná grupa je konečná a cyklická: má generátor, totiž bod g takový, že jeho postupným sčítáním dostaneme všechny body, které leží na dané křivce. [3]

Abelovská grupa musí obsahovat neutrální prvek. [3] U eliptické křivky se tento prvek nazývá bodem v nekonečnu (O), a podle [5] splňuje:

$$\begin{aligned} P + O &= P \\ P + (-P) &= O \end{aligned} \tag{2.4}$$

2.2.1 Sčítání bodů

V [2] je sčítání bodů eliptické křivky definované pomocí následující geometrické interpretace:

- Pokud jsou dva body odlišné, vede mezi nimi přímka. Výsledkem operace je průsečík přímky a eliptické křivky zrcadlený podle osy X (viz. obrázek 2.1).
- Jsou-li dva body stejné, pak výsledkem operace bude průsečík eliptické křivky a tečny, zrcadlený podle osy X. Tato operace se často značí jako $2P = P + P$ (viz. obrázek 2.2).

Analyticky podle [5] definujeme sčítání následovně: nechť máme eliptickou křivku E a dva body $P_1, P_2 \in E$.

1. Pokud $P_1 = O$, pak $P_1 + P_2 = P_2$.
2. Pokud $P_2 = O$, pak $P_1 + P_2 = P_1$.

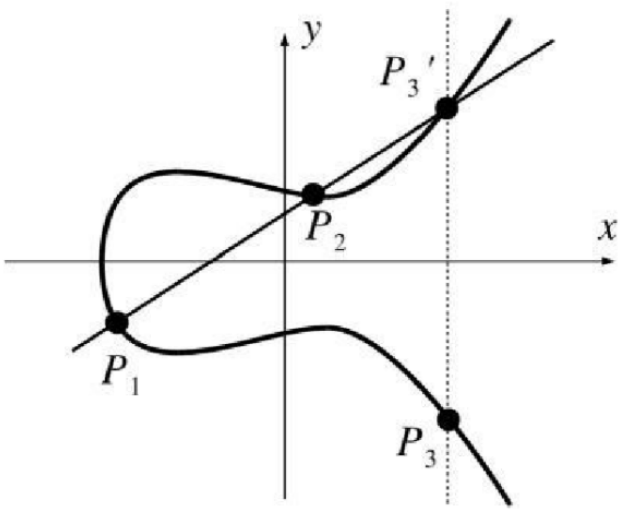
3. Jinak zavedeme $P_1 = (x_1, y_1)$ a $P_2 = (x_2, y_2)$.

4. Když $x_1 = x_2$ a $y_1 = -y_2$, pak $P_1 + P_2 = O$

5. Jinak zavedeme

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{pokud } P_1 \neq P_2 \\ \frac{3x_1^2 + A}{2y_1} & \text{pokud } P_1 = P_2 \end{cases} \tag{2.5}$$

Potom $x_3 = \lambda^2 - x_1 - x_2$ a $y_3 = \lambda(x_1 - x_3) - y_1$. Výsledek sčítání $P_1 + P_2 = (x_3, y_3)$.



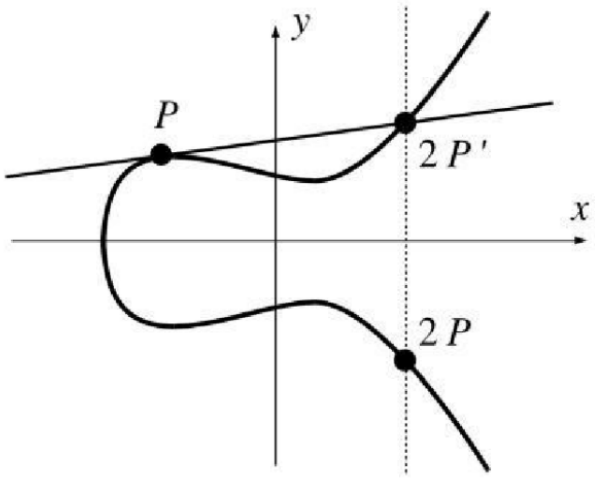
Obrázek 2.1: Sčítání dvou různých bodů [3]

2.2.2 Násobení bodu skalárem

Nechť máme eliptickou křivku E, bod $P \in E$ a přirozené číslo n. Potom v [3] je násobení bodu P číslem n definováno jako bod T, takový, že:

$$T = n \cdot P = \underbrace{P + P + P + \cdots + P}_{n \text{ krát}} \tag{2.6}$$

Protože $E(F_p)$ je konečná množina, všechny body tvaru $P, 2P, 3P, 4P, \ldots$ nemůžou byt odlišné. Proto existují čísla $k, j \in N$, $k > j$, takové, že $kP = jP$. Nejmenší číslo s, takové, že $s = k - j$ potom nazýváme řád bodu P. [5]



Obrázek 2.2: Sčítání dvou stejných bodů (zdvojnásobení) [3]

2.2.3 Eliptické křivky nad konečnými tělesy

V [5] je uvedená následující definice: nechť $p \geq 3$ je prvočíslo. Eliptická křivka nad konečným tělesem F_p je výraz tvaru:

$$E : y^2 = x^3 + Ax + B, \tag{2.7}$$

kde $A, B \in F_p$ a splňují $4A^3 + 27B^3 \neq 0$ (kde 4 je 2^2 a 27 je 3^3). Podle [5] je množina bodů E se souřadnicemi z F_p rovna

$$E(F_p) = \{(x, y) : x, y \in F_p, \text{ splňující } y^2 = x^3 + Ax + B\} \cup \{O\} \tag{2.8}$$

2.2.4 Sčitání bodů eliptické křivky v konečném tělese

Nechť E je eliptická křivka nad F_p, P a Q jsou dva body z $E(F_p)$. Pokud aplikujeme analytický algoritmus pro sčítání, uvedený v 2.2.1 na P a Q, dostaneme bod $P + Q \in E(F_p)$. [5]

2.2.5 Problém diskrétního logaritmu na eliptické křivce

Nechť E je eliptická křivka nad konečným tělesem F_p, P a Q jsou body v $E(F_p)$. Problém diskrétního logaritmu na eliptické křivce (ECDLP) je najít

přirozené číslo n, takové, že $Q = nP$. Říkáme, že n je eliptický logaritmus Q vzhledem k P a značíme jej:

$$n = \log_P(Q) \tag{2.9}$$

Tento úkol pro velká p je výpočetně velmi náročný. Nejrychlejší známý algoritmus pro řešení ECDLP potřebuje $\sqrt{p}$ kroků. [5]

2.3 Hashovací funkce

Hashovací funkce je algoritmus, který jako vstup dostává data libovolné délky, a produkuje řetězec bitů fixní délky — hash. [3] Pro hashovací funkce je podstatná rychlost výpočtu, která je málo ovlivnitelná délkou vstupní zprávy. [2] Další vlastnosti hashovacích funkcí jsou:

- Jednosměrnost: vyžaduje, aby pro známý hash bylo výpočetně neproveditelné získat vstup, z něhož byl tento hash spočítán. [3]
- Odolnost vůči získání jiné předlohy: nechť máme vstup x a hash y, takový, že $h(x) = y$. Odolnost vůči získání jiné předlohy znamená, že je výpočetně neproveditelné najít další vstup $z \neq x$, pro který platí $h(z) = y$. [2][3]
- Odolnost vůči nalezení kolize: tato vlastnost vyžaduje, aby dvě různé zprávy neměly stejný hash. Pokud máme dvě zprávy x, z, přičemž $x \neq z$, potom $h(x) \neq h(z)$.

  Vzhledem k neomezenému počtu možných vstupů a omezené délce výstupu, hashovací funkce vždy bude obsahovat kolize. Bezpečná hashovací funkce přiděluje hashe tak, aby každému hashi odpovídal téměř stejný počet vstupů. Pro takové funkce je typický takzvaný „lavinový“ efekt. Tento efekt spočívá v tom, že i velmi malá změna ve vstupu hashovací funkce způsobí velké změny ve výsledku. [2][3]

2.4 Digitální podpis

Digitální podpis je matematický systém, který má podobné účely jako psaný podpis. Takový podpis je nepopiratelný, zaručuje, že zpráva byla vygenerovaná podepisovatelem a nebyla s ní provedena žádná manipulace. [2]

2.4.1 ECDSA

ECDSA je algoritmus digitálního podpisu, který pro svou činnost používá eliptické křivky. V [5] je popsán následující algoritmus: důvěryhodná strana volí konečné těleso F_p, křivku E nad F_p a bod $G \in E(F_p)$ velkého prvočíselného řádu q. Pak následuje generace klíčů:

1. Zvolí se náhodné číslo d, takové, že $0 < d < q-1$, které bude soukromým klíčem.
2. Pomocí vygenerovaného soukromého klíče d se spočítá veřejný klič $Q = dG \in E(F_p)$.

Tento pár klíčů se použije pro podepisování a následné ověřování zpráv. Algoritmus pro podepisování se skládá z následujících kroků:

1. Vygeneruje se efemerní číslo e, pro které platí $0 < e < q$. Musí být zaručeno, že toto číslo bude použito pouze jednou. Můžeme to téměř úplně vyřešit použitím zabezpečeného generátoru náhodných čísel.
2. Spočítá se $eG \in E(F_p)$, a poté
   - $s_1 = x(eG) \pmod q \in F_p$
   - $s_2 = (h(m)+ds_1)\cdot e^{-1} \pmod q \in F_p$, kde m je podepisovaná zpráva a $h(m)$ je její hash.

Výsledný podpis se potom skládá ze dvojice čísel (s_1, s_2). Tento podpis lze ověřit následujícím způsobem:

1. Spočítá se $v_1 = h(m) \cdot s_2^{-1} \pmod q$ a $v_2 = s_1 s_2^{-1} \pmod q$
2. Dopočítáme bod $P = (x_p, y_p)$, jako $P = v_1 G + v_2 Q \in E(F_p)$
3. Pokud platí, že $x_p \pmod q = s_1$, pak je podpis validní.

Znovu zdůrazním důležitost toho, aby se náhodné číslo e znovu nepoužilo. Podle [3] pokud by bylo stejné e použité ve dvou různých podpisech:

$$\begin{aligned} a_1 &= (h(m_1) + ds_1) \cdot e^{-1} \pmod q \\ a_2 &= (h(m_2) + ds_1) \cdot e^{-1} \pmod q \end{aligned} \tag{2.10}$$

Útočník by mohl odečíst a_1 a a_2 a získat e:

$$\begin{aligned} &a_1 - a_2 = (h(m_1) + ds_1) \cdot e^{-1} - (h(m_2) + ds_1) \cdot e^{-1} \pmod q = \\ &(h(m_1) - h(m_2)) \cdot e^{-1} \pmod q \\ &\implies e = (h(m_1) - h(m_2))/(a_1 - a_2) \pmod q \end{aligned}$$

Jakmile je známo e, lze vypočítat d:

$$d = (a_1 e - h(m_1))/s_1 \pmod q \tag{2.11}$$

Kapitola 3

Analýza kryptografických knihoven

Tato kapitola je věnovaná rešerši kryptografických knihoven. V rámci rešerše bylo vybráno 8 knihoven, podporujících práci s eliptickými křivkami. Zohledněny jsou pouze knihovny s možností bezplatného použití.

3.1 cryptlib

Bezpečnostní software cryptlib byl původně napsán zakladatelem Digital Data Security Limited (DDS) Dr. Peterem Gutmannem v rámci jeho doktorské práce na Aucklandské univerzitě v polovině 90. let a následně komercializován.

Kryptografie eliptických křivek (ECC) je v cryptlib podporována, a to jak na nižší úrovni krypto-mechanismu, tak v protokolech na vysoké úrovni, jako je X.509, SSL/TLS, S/MIME a SSH. Nejnovější verze knihovny 3.4.5 byla vydána v srpnu 2019. [6]

3.1.1 Licence

cryptlib je distribuována na základě dvojí licence, která umožňuje bezplatné použití na základě licence kompatibilní s GPL (aka licence „Sleepycat“) anebo použití na základě standardní komerční licence. Kromě toho použití cryptlib je často zdarma pro použití v levných, neotevřených aplikacích, jako je shareware, a pro osobní a výzkumné použití. [6]

3.1.2 Podepisování zpráv

Funkce `cryptCreateSignature` digitálně podepíše hash dat. Podpis je umístěn do vyrovnávací paměti v přenosném formátu, který umožňuje jeho kontrolu pomocí `cryptCheckSignature`. [7]

```
int cryptCreateSignature
                (void *signature,
                 const int signatureMaxLength,
                 int *signatureLength,
                 const CRYPT_CONTEXT signContext,
                 const CRYPT_CONTEXT hashContext);
```

Parametr	Popis
`signature`	Adresa vyrovnávací paměti, která bude obsahovat podpis. Pokud nastavíte tento parametr na `NULL`, `cryptCreateSignature` vrátí délku podpisu v `signatureLength` bez skutečného generování podpisu
`signatureMaxLength`	Maximální velikost vyrovnávací paměti v bajtech, která bude obsahovat podpis
`signatureLength`	Adresa proměnné s délkou podpisu
`signContext`	Kontext šifrování obsahující soukromý klíč použitý k podepsání dat
`hashContext`	Kontext obsahující hash dat k podpisu

Funkce vrátí `CRYPT_OK`, pokud se podařilo vygenerovat a uložit digitální podpis, nebo jeden z chybových kódů v případě chyby. Seznam všech kódů je uveden v [7].

3.1.3 Ověření podpisu

Funkce `cryptCheckSignature` se používá ke kontrole digitálního podpisu. [7]

```
int cryptCheckSignature
                (const void *signature,
                 const int signatureLength,
                 const CRYPT_HANDLE sigCheckKey,
                 const CRYPT_CONTEXT hashContext);
```

Parametr	Popis
`signature`	Adresa vyrovnávací paměti, která obsahuje podpis
`signatureLength`	Délka podepisovaných dat v bajtech
`sigCheckKey`	Kontext veřejného klíče nebo objekt certifikátu klíče obsahující veřejný klíč používaný k ověření podpisu
`hashContext`	Kontext obsahující hash dat

Funkce vrátí `CRYPT_OK`, pokud se podpis podařilo ověřit, nebo jeden z chybových kódů v případě chyby. Seznam všech kódů je uveden v [7].

3.2 libgcrypt

Libgcrypt je kryptografická knihovna pro všeobecné použití původně založená na kódu z GnuPG. Poskytuje funkce pro všechny kryptografické stavební bloky: symetrické šifrovací algoritmy a režimy, hashovací algoritmy, MAC, algoritmy veřejného klíče (RSA, Elgamal, DSA, ECDSA, EdDSA, ECDH), velké celočíselné funkce, náhodná čísla a mnoho podpůrných funkcí. Aktuální stabilní verze je 1.9.3, která byla vydána 19. 04. 2021. [8]

3.2.1 Licence

Libgcrypt je distribuována za podmínek GNU Lesser General Public License (LGPLv2.1 +). Pomocné programy i dokumentace jsou distribuovány v souladu s podmínkami GNU General Public License (GPLv2 +). [8]

3.2.2 Podepisování zpráv

Tato funkce vytváří digitální podpis pro data pomocí soukromého klíče. [9]

```
gcry_error_t gcry_pk_sign(gcry_sexp_t *r_sig,
                          gcry_sexp_t data,
                          gcry_sexp_t skey);
```

Parametr	Popis
`r_sig`	Ukazatel do paměti, kde bude uložen podpis
`data`	Data k podpisu
`skey`	Soukromý klíč k podepsání zprávy

Výsledkem je 0 v případě úspěchu, nebo chybový kód jinak. [9]

3.2.3 Ověření podpisu

Tato funkce slouží ke kontrole, zda digitální podpis odpovídá datům. [9]

```
gcry_error_t gcry_pk_verify(gcry_sexp_t sig,
                            gcry_sexp_t data,
                            gcry_sexp_t pkey);
```

Parametr	Popis
`sig`	Podpis k ověření
`data`	Data zprávy
`pkey`	Veřejný klíč k provedení ověření

Výsledkem je 0 pro úspěch (tj. data se shodují s podpisem) nebo chybový kód, kde nejčastější kód je `GCRY_ERR_BAD_SIGNATURE`, který označuje, že podpis neodpovídá poskytnutým datům. [9]

3.3 GNUTLS

GnuTLS je kryptografická knihovna implementující bezpečnostní protokoly SSL, TLS a DTLS. Poskytuje jednoduché rozhraní pro programování aplikací v jazyce C (API) pro přístup k zabezpečeným komunikačním protokolům a také API pro analýzu a zápis X.509, PKCS # 12 a dalších požadovaných struktur. Verze 3.7.2 byla vydaná 29. 5. 2021. [10]

3.3.1 Licence

Základní knihovna licencovaná pod GNU Lesser General Public License verze 2.1 (LGPLv2.1 +). Licence LGPL je kompatibilní s celou řadou bezplatných licencí a dokonce umožňuje používat GnuTLS v proprietárních programech. [10]

3.3.2 Podepisování zpráv

Tato funkce podepíše zadaná hashovaná data pomocí podpisového algoritmu podporovaného soukromým klíčem. Podpisové algoritmy se vždy používají společně s hashovacími funkcemi. Pro algoritmus RSA lze použít různé hashovací funkce, ale pouze hashe typu SHA-XXX pro klíče DSA. [11]

```
int gnutls_privkey_sign_hash
                (gnutls_privkey_t signer,
                 gnutls_digest_algorithm_t hash_algo,
                 unsigned int flags,
                 const gnutls_datum_t * hash_data,
                 gnutls_datum_t * signature);
```

Parametr	Popis
`signer`	Drží klíč podepisovatele
`hash_algo`	Použitý hashovací algoritmus
`flags`	0, nebo jeden z `gnutls_privkey_flags_t`
`hash_data`	Uchovává data, která mají být podepsána
`signature`	Bude obsahovat nově vytvořený podpis

Při úspěchu je vrácen `GNUTLS_E_SUCCESS` (0), jinak se vrátí záporná hodnota chyby. [11]

3.3.3 Ověření podpisu

Tato funkce ověří daný podpis pomocí veřejného klíče. [11]

```
int gnutls_pubkey_verify_hash2
                (gnutls_pubkey_t key,
```

```
                    gnutls_sign_algorithm_t algo,
                    unsigned int flags,
                    const gnutls_datum_t * hash,
                    const gnutls_datum_t * signature);
```

Parametr	Popis
`key`	Drží veřejný klíč
`algo`	Použitý podepisovací algoritmus
`flags`	0, nebo seznam `gnutls_certificate_verify_flags`, spojený pomocí operatoru „OR“
`hash`	Obsahuje hash pro ověření
`signature`	Obsahuje podpis

V případě selhání ověření se vrátí `GNUTLS_E_PK_SIG_VERIFY_FAILED` a nulový nebo kladný kód při úspěchu. Pokud je známo, že podpis s takovou délkou klíče není zabezpečený a není zadán flag `GNUTLS_VERIFY_ALLOW_BROKEN`, pak je vrácen `GNUTLS_E_INSUFFICIENT_SECURITY`. [11]

3.4 LibTomCrypt

LibTomCrypt je poměrně komplexní, modulární a přenosná kryptografická sada nástrojů, která poskytuje vývojářům širokou škálu kryptografických algoritmů. Poslední verze knihovny je 1.18.2 od 3. 7. 2018. [19]

3.4.1 Licence

LibTomCrypt je zdarma pro všechny účely pod veřejnou doménou. To zahrnuje komerční využití, redistribuci a dokonce i větvení. [19]

3.4.2 Podepisování zpráv

Funkce pro podepsání hashe zprávy má následující signaturu:

```
int ecc_sign_hash(const unsigned char *in,
                  unsigned long inlen,
                  unsigned char *out,
                  unsigned long *outlen,
                  prng_state *prng,
                  int wprng,
                  ecc_key *key);
```

Parametr	Popis
`in`	Ukazatel na vyrovnávací paměť obsahující hash zprávy k podpisu
`inlen`	Délka hashe k podpisu
`out`	Vyrovnávací paměť, do které se má vygenerovaný podpis uložit
`outlen`	Maximální délka výstupní vyrovnávací paměti
`prng`	Ukazatel na aktivní stav PRNG
`wprng`	Index PRNG, který se má použit
`key`	Ukazatel na soukromý klíč, pomocí kterého bude generován podpis

V případě úspěchu je vrácená `CRYPT_OK`. Jinak je vrácen jeden z chybových kódů. [20]

3.4.3 Ověření podpisu

Pro ověření digitálního podpisu se použije následující funkce:

```
int ecc_verify_hash(const unsigned char *sig,
                    unsigned long siglen,
                    const unsigned char *hash,
                    unsigned long hashlen,
                    int *stat,
                    ecc_key *key);
```

Parametr	Popis
`sig`	Ukazatel na vyrovnávací paměť obsahující podpis k ověření
`siglen`	Délka podpisu
`hash`	Ukazatel na vyrovnávací paměť obsahující hash zprávy
`hashlen`	Délka hashe
`stat`	Ukazatel na výsledek ověření. Nenulová hodnota označuje, že zpráva byla úspěšně ověřena
`key`	Ukazatel na klíč, který bude použit pro ověření

Funkce vrátí `CRYPT_OK`, nebo chybový kód pokud poskytnutý podpis má špatný formát. Poznámka: funkce nevrátí chybu, pokud je podpis neplatný. [20]

3.5 wolfCrypt

Kryptografický modul wolfCrypt je odlehčená kryptografická knihovna napsaná v ANSI C a zaměřená na vestavěná prostředí, prostředí RTOS a prostředí omezená zdroji – především kvůli své malé velikosti, rychlosti a sadě

funkcí. Běžně se používá i ve standardních operačních prostředích kvůli cenám bez licenčních poplatků a vynikající podpoře mezi platformami. Poslední verze knihovny 4.7.0 byla vydaná 15. 2. 2021. [12]

Vzhledem k tomu, že část API knihovny je velmi podobná API LibTomCrypt lze předpokládat, že tento kód byl odtud přebrán.

3.5.1 Licence

Software wolfSSL je k dispozici ve dvou odlišných licenčních modelech: open source a standardní komerční licence. wolfCrypt je ke stažení zdarma a může být upraven podle potřeb uživatele, pokud se uživatel bude řídit verzí 2 licence GPL. [12]

3.5.2 Podepisování zpráv

Tato funkce podepisuje hashovaná data pomocí objektu ecc_key. [13]

```
int wc_ecc_sign_hash(const byte * in, word32 inlen,
                     byte * out, word32 * outlen,
                     WC_RNG * rng, ecc_key * key);
```

Parametr	Popis
`in`	Ukazatel na vyrovnávací paměť obsahující hash zprávy k podpisu
`inlen`	Délka hashe k podpisu
`out`	Vyrovnávací paměť, do které se má vygenerovaný podpis uložit
`outlen`	Maximální délka výstupní vyrovnávací paměti
`rng`	Ukazatel na strukturu `RNG`, která se použije pro generování náhodných čísel
`key`	Ukazatel na soukromý klíč, pomocí kterého bude generován podpis

V případě úspěšného generování podpisu pro daný hash je vrácená 0. Jinak je vrácen jeden z chybových kódů. [13]

3.5.3 Ověření podpisu

Tato funkce ověří podpis hashovaných dat. [13]

```
int wc_ecc_verify_hash(const byte * sig,
                       word32 siglen,
                       const byte * hash,
                       word32 hashlen, int * stat,
                       ecc_key * key);
```

Parametr	Popis
`sig`	Ukazatel na vyrovnávací paměť obsahující podpis k ověření
`siglen`	Délka podpisu k ověření
`hash`	Ukazatel na vyrovnávací paměť obsahující hash ověřované zprávy
`hashlen`	Délka hashe
`stat`	Ukazatel na výsledek ověření. 1 označuje, že zpráva byla úspěšně ověřena
`key`	Ukazatel na veřejný klíč, kterým se ověřuje podpis

Po úspěšném provedení ověření podpisu je vrácená 0. Poznámka: to neznamená, že podpis je validní. Informace o správností podpisu se místo toho uloží do proměnné `stat`. [13]

3.6 OpenSSL

OpenSSL je robustní, komerční a plně funkční sada nástrojů pro protokoly TLS a SSL. Jedná se také o univerzální kryptografickou knihovnu. Poslední stabilní verze je 1.1.1k (25. 3. 2021). [14]

3.6.1 Licence

OpenSSL je licencován pod licencí ve stylu Apache, což znamená, že jej lze získat a používat pro komerční a nekomerční účely za určitých jednoduchých licenčních podmínek. [14]

3.6.2 Podepisování zpráv

Funkce vypočítá ECDSA podpis daného hashe pomocí soukromého klíče. [15]

```
int ECDSA_sign(int type, const unsigned char *dgst,
               int dgstlen, unsigned char *sig,
               unsigned int *siglen, EC_KEY *eckey);
```

Parametr	Popis
`type`	Tento parametr je ignorován
`dgst`	Ukazatel na hash k podpisu
`dgstlen`	Délka hashe
`sig`	Ukazatel na vyrovnávací paměť pro vytvořený podpis zakódovaný DER
`siglen`	Ukazatel na proměnnou, obsahující délku vráceného podpisu
`eckey`	Objekt `EC_KEY` obsahující soukromý klíč

Funkce vrátí 1 v případě úspěchu, nebo 0, pokud došlo k chybě. [15]

3.6.3 Ověření podpisu

Tato funkce ověřuje, že daný podpis je platný ECDSA podpis zadaného hashe pomocí veřejného klíče. [15]

```
int ECDSA_verify(int type, const unsigned char *dgst,
                 int dgstlen, const unsigned char *sig,
                 int siglen, EC_KEY *eckey);
```

Parametr	Popis
`type`	Tento parametr je ignorován
`dgst`	Ukazatel na hash
`dgstlen`	Délka hashe
`sig`	Ukazatel na podpis kódovaný DER
`siglen`	Délka podpisu
`eckey`	Objekt `EC_KEY` obsahující veřejný klíč

Funkce vrátí 1, pokud je podpis platný, 0 pro neplatný podpis a -1 pokud došlo k chybě. [15]

3.7 LibreSSL

LibreSSL je knihovna vytvořená z OpenSSL v roce 2014, jejímž cílem je modernizace kódu a zlepšení zabezpečení. Nejnovější stabilní verze je 3.3.3, vydaná 3. 5. 2021. [16]

3.7.1 Licence

Knihovna je uvolněna pod BSD licencí. [16]

3.7.2 Metody pro práce s podpisy

Vzhledem k tomu, že LibreSSL vychází z OpenSSL, knihovna má stejné rozhraní pro podepisování zpráv a ověřování podpisů.

3.8 Mbed TLS

Mbed TLS je knihovna, která implementuje kryptografická primitiva, manipulaci s certifikáty X.509 a protokoly SSL/TLS a DTLS. Díky malé stopě kódu je vhodný pro vestavěné systémy. Mbed TLS byla dříve známá jako PolarSSL. Od března 2020 je udržována pod otevřenou správou na TrustedFirmware. Před tím byla udržována společností Arm. Nejnovější verze knihovny je 2.26.0 vydaná 12. 03. 2021. [17]

3.8.1 Licence

Všechny aktuální verze knihovny Mbed TLS jsou distribuovány pod licencí Apache 2.0. Kromě toho existují zabalené verze knihovny Mbed TLS, které jsou distribuovány s GNU Public License verze 2.0 (GPL v2.0). [17]

3.8.2 Podepisování zpráv

Tato funkce počítá podpis ECDSA a zapisuje jej do vyrovnávací paměti serializované podle definice v *RFC-4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS).* [18]

```
int mbedtls_ecdsa_write_signature_det
        (mbedtls_ecdsa_context * ctx,
         const unsigned char * hash,
         size_t hlen, unsigned char * sig,
         size_t * slen, mbedtls_md_type_t md_alg);
```

Parametr	Popis
`ctx`	Kontext ECDSA, který se má použít. Musí být inicializován a mít k němu vázanou grupu a veřejný klíč
`hash`	Hash zprávy k podpisu. Musí to být vyrovnávací paměť délky `hlen` bajtů
`hlen`	Velikost hashe
`sig`	Vyrovnávací paměť, do které se má zapsat podpis. Musí to být vyrovnávací paměť délky alespoň dvakrát větší než je velikost použité křivky plus 9 bajtů
`slen`	Adresa, na kterou se má uložit skutečná délka zapsaného podpisu
`md_alg`	Použitá hashovací funkce

Funkce vrátí 0 v případě úspěchu, nebo jeden z chybových kódů. [18]

3.8.3 Ověření podpisu

Tato funkce načte a ověří ECDSA podpis. [18]

```
int mbedtls_ecdsa_read_signature
        (mbedtls_ecdsa_context * ctx,
         const unsigned char * hash,
         size_t hlen,
         const unsigned char * sig, size_t slen);
```

Parametr	Popis
`ctx`	Kontext ECDSA, který se má použít. Musí být inicializován a mít k němu vázanou grupu a veřejný klíč
`hash`	Podepsaný hash zprávy. Musí to být vyrovnávací paměť délky `hlen` bajtů
`hlen`	Velikost hashe
`sig`	Podpis ke čtení a ověření. Musí to být čitelná vyrovnávací paměť délky `slen` bajtů
`slen`	Velikost `sig` v bajtech

Funkce vrátí 0 pro úspěch, `MBEDTLS_ERR_ECP_BAD_INPUT_DATA`, pokud je podpis neplatný a `MBEDTLS_ERR_ECP_SIG_LEN_MISMATCH`, pokud je podpis platný, ale jeho délka je menší než `siglen`. [18]

3.9 Zvolené řešení

Pro naše účely budeme používat knihovnu OpenSSL, nebo LibreSSL, jako nejrozšířenější a nejvíc zdokumentované. Kvůli stejnému rozhraní těchto knihoven nemusíme přesně definovat, která z nich se musí použít. Použije se knihovna, která je zrovna přístupna na uživatelském systému.

Kapitola 4

Návrh řešení pro FITCOIN

Tato kapitola popisuje, jak lze dříve uvedené kryptografické mechanismy použít k zajištění bezpečnosti v technologii blockchain. Kapitola také obsahuje popis datových struktur, které byly v rámci této bakalářské práce změněné.

4.1 Teorie

Blockchain je druh distribuované decentralizované databáze a každý účastník sítě do ní může přidat záznam. Proto musíme zajistit, aby nebylo možné přidat takový záznam, který by utrácel cizí prostředky. Pro tyto účely se právě hodí digitální podpis.

4.1.1 Autorizace utrácení

V rámci své bakalářské práce navrhuji následující řešení: každá transakce bude obsahovat seznam vstupů a výstupů. Vstupy definují, odkud a kolik mincí by se mělo vybrat pro provedení této transakce. Výstupy definují, kam a kolik mincí by se mělo připsat po provedení této transakce. Každý vstup obsahuje následující položky: částka, číslo `nonce`, které zabrání replay útoku (viz 4.1.2), veřejný klíč odesilatele a ECDSA podpis daného vstupu. Adresa je potom SHA256 hash veřejného klíče a má délku 256 bitů = 32 bajtů.

Při vytvoření transakce se každý vstup musí podepsat. To se provede v následujících krocích:

1. Spočítá se SHA256 hash hlavičky vstupu (tzn. částky a `nonce`).
2. Pomocí soukromého klíče, příslušného veřejnému klíči, se tento hash podepíše.
3. Získaný podpis se vloží do vstupu.

Tyto podpisy ověřuje každý účastník, když se dozví o nové transakci. Ověření se provede následujícím způsobem:

1. Spočítá SHA256 hash hlavičky vstupu.

2. Tento hash se pomocí veřejného klíče ověří proti podpisu.

Pokud by někdo chtěl padělat vstup, aniž by znal soukromý klíč, pak není schopen vygenerovat správný podpis, a tím pádem se blok s takovou transakcí nemůže stát součástí blockchainu.

4.1.2 Replay útok

Vzhledem k otevřenosti a transparentnosti blockchainu, kde každý uživatel má přístup k datům všech bloků a transakcí, vzniká riziko přehrávání (replay).

Nechť máme dva uživatele FITCoinu – Alici a Boba. Všechny vstupy, které Alice utrácí, musí být podepsané soukromým klíčem Alice, který Bob nezná. Nemůže tedy vytvořit validní transakci, která by utrácela mince od Alice a ukradnout její mince.

Uvažujeme situaci, kde Alice už někdy posílala Bobovi mince. Vzhledem k tomu, že předchozí transakce byla vytvořená vlastníkem adresy a je validní, Bob ji muže vzít a znovu ji rozeslat všem účastníkům sítě. Transakce bude akceptovaná do dalšího bloku a Alice ztratí své prostředky podruhé.

4.1.3 Ochrana proti replay útoku

Pro ochranu proti replay útoku, přidáme do struktury vstupu další parametr `nonce`. Je to celé číslo, které ukazuje, kolik odchozích transakcí používalo danou adresu jako vstup. Toto číslo se zapisuje do vstupu v momentě vytvoření transakce a stává se její součástí. Kvůli tomu, že vstup je podepisován pomocí soukromého klíče odesílatele, není možné s tímto číslem manipulovat. Aby byla nová přijatá transakce validní, musí být číslo `nonce`, obsažené v každém vstupu stejné, nebo větší, než počet odchozích transakcí z dané adresy.

Pokud by útočník chtěl přehrát transakce s použitým vstupem, pak by `nonce`, obsažené ve vstupu bylo menší, než počet odchozích transakcí z dané adresy, a blockchain by jí odmítl.

4.2 Datové struktury

Tato sekce popisuje datové struktury blockchainu FITCoin, které byly v rámci této práce upravené, a význam jednotlivých položek.

4.2.1 Transakce

Transakce je struktura, která slouží pro změnu vlastnictví coinu. Transakce je definovaná v souboru `tx.h`.

Položka	Typ	Popis
`size`	uint16_t	Velikost struktury, včetně této proměnné
`numi`	uint8_t	Počet vstupů, které budou utracené v dané transakci
`numo`	uint8_t	Počet výstupů, které budou použité v dané transakci
`data`	unsigned char[]	`numi` vstupů a `numo` výstupů

4.2.2 Vstup

Vstup obsahuje všechny informace nutné pro útratu prostředků. Jeho struktura je definovaná v souboru `in.h`.

Proměnná	Typ	Popis
`amount`	uint32_t	Počet mincí, které budou z této adresy utraceny
`nonce`	uint32_t	Pořadové číslo transakce, používající danou adresu jako vstup
`slen`	uint32_t	Délka digitálního podpisu
`pkey`	char[131]	Veřejný klíč odesilatele
`signature`	unsigned char[72]	ECDSA digitální podpis

4.2.3 Výstup

Výstup definuje, kam budou prostředky převedeny po uplatnění transakce. Soubor `out.h` popisuje následující strukturu:

Proměnná	Typ	Popis
`addr`	unsigned char[32]	Adresa příjemce
`amount`	uint32_t	Počet mincí, které budou připsáné na danou adresu

4.2.4 Zdroj (src)

Struktura `src` reprezentuje všechny údaje o jedné adrese potřebné pro správné fungování soukromé peněženky. Je definovaná v souboru `wallet.c`.

Proměnná	Typ	Popis
`have`	struct in	Vstup, obsahující informace o adrese
`key`	EC_KEY *	Soukromý klíč příslušný k dané adrese
`prev`	struct src*	Další zdroj v seznamu
`next`	struct src*	Předchozí zdroj v seznamu

4.2.5 Peněženka

Struktura `wallet` je definovaná v souboru `wallet.h` a obsahuje všechny informace o existujících adresách uživatele, jejich zůstatcích a o celkovém zůstatku.

Proměnná	Typ	Popis
`dir`	char[128]	Cesta do složky wallet
`balance`	uint32_t	Počet mincí na všech adresách peněženky
`num`	uint32_t	Počet zdrojů
`head`	struct src*	První zdroj
`tail`	struct src*	Poslední zdroj

Kapitola 5

Implementace

Tato sekce popisuje metody, které byly přidané, nebo upravené v rámci této bakalářské práce. Významné časti kódu jsou okomentované.

5.1 Podepisování vstupu

Podepisování vstupu má za úkol funkce, která je umístěna v souboru `in.c` a má následující signaturu:

```
int signin (struct in * in, EC_KEY * key)
```

Na vstup funkce dostane ukazatel na strukturu `in`, kterou chceme podepsat, a klíč, který bude pro tento podpis použit. Na začátku metoda provede kontrolu, že daný klíč je validní, spočítá hash hlavičky vstupu (tzn. položek `amount` a `nonce`), spočítá digitální ECDSA podpis a uloží ho do struktury `in` spolu s jeho délkou. Výpočet samotného podpisu se provádí pomocí následujícího příkazu:

```
ECDSA_sign(0, hash, 32, signature, &size, key))
```

První parametr je ignorován, pak následuje hash hlavičky vstupu, délka hashe, signature – pole, kam se uloží podpis, size – ukazatel na proměnnou, kam se uloží délka podpisu, key – ukazatel na `EC_KEY` obsahující soukromý klíč podepisovatele.

Funkce `signin` vrátí 0 v případě úspěchu a -1, pokud během podepisování došlo k chybě.

5.1.1 Ověření podpisu

Ověření podpisu se provede pomocí funkce, která je definovaná v souboru `in.c`. Funkce má následující signaturu:

```
int verifyin(const struct in *in);
```

Funkce spočítá hash vstupu a ověří ho pomocí veřejného klíče proti podpisu, který je ve vstupu obsažen.

Ověření se provede zavoláním metody `ECDSA_verify` z knihovny OpenSSL (resp. LibreSSL):

```
ECDSA_verify(0, hash, 32,
             in->signature, in->slen, key);
```

První parametr je ignorován, pak následuje hash hlavičky vstupu, délka hashe, digitální podpis, délka podpisu a ukazatel na `EC_KEY`, obsahující veřejný klíč pro ověření.

Funkce `verifyin` dostane vstup a vrátí 1, pokud vstup obsahuje správný podpis, 0, pokud podpis není validní a -1, v případě, že během ověření došlo k chybě.

5.2 Ověření správnosti transakce

Ověření, jestli je transakce validní, se provede pomocí metody `int checktx (const struct tx* tx)` ze souboru `tx.c`.

V rámci smyčky se postupně iteruje podle vstupu a na každý z nich je volaná metoda `verifyin`, která kontroluje, jestli vstup má správný podpis.

```
for (i = 0, in = (struct in*)tx->data;
     i < tx->numi && size < tx->size; i++, in++) {
        if (1 != verifyin(in)) {
                return -1;
        }
    // ...
}
```

Pokud se podpis nepodaří ověřit, metoda zahlásí chybu a skončí s návratovou hodnotou -1.

5.3 Načítání klíčů

Funkce pro načítaní klíčů `getkey` je definovaná v souboru `key.c`. Funkce dostane na vstup jméno souboru, obsahující soukromý klíč (standardní umístění je `~/.ftc/wallet`), spočítá veřejný klíč a uloží pár klíčů do struktury `EC_KEY`.

```
EC_KEY * getkey(const char *name);
```

Soukromý klíč se načte ze souboru a následně je konvertován do struktury `BIGNUM` pomocí následujícího příkazu:

```
BN_hex2bn(&privkey, buf)
```

Pro výpočet veřejného klíče byla použita metoda `EC_POINT_mul`, která provéde operaci, popsanou v sekci 2.2.2. Následující příkaz vynásobí generátor grupy `group` soukromým klíčem `privkey` a uloží výsledek do proměnné `pubkey`.

```
EC_POINT_mul(group, pubkey, privkey, NULL, NULL, ctx))
```

V rámci metody se vytvoří nová struktura `EC_KEY`, přiřazení soukromého a veřejného klíče proběhne v následující části:

```
EC_KEY_set_private_key(key, privkey);
EC_KEY_set_public_key(key, pubkey);
```

5.4 Vytvoření transakce v peněžence

Za vytvoření nového příkazu o převodu peněz je zodpovědná funkce `mkpaytx` ze souboru `wallet.c`. Součástí kódu je smyčka, která iteruje přes všechny prostředky nutné pro provedení platby a v průběhu vyplňuje a podepisuje vstupy transakce.

```
for (i = 0, pay = funds->pay;
     i < numi && pay;
     i++, pay = pay->next) {

        in[i] = pay->src->have;
        signin(&in[i], pay->src->key);
}
```

5.5 Výběr ze vstupu

Do funkce `int withdraw(const struct in *in, const char* dir)` ze souboru `in.c` byla přidaná kontrola čísla `nonce`. Pokud číslo `nonce` obsažené ve vstupu je menší, než `nonce`, nacházející se v příslušném souboru ze složky `dir`, funkce zahlásí chybu a skončí.

5.6 Odrážení transakce v peněžence

Do funkce `int verify(const struct tx *tx, struct wallet *wallet)` ze souboru `wallet.c`, která zkontrolujte, zda je transakce legitimní z pohledu peněženky, byla také přidaná kontrola čísla `nonce`.

Funkce iteruje přes všechny vstupy a pokud je vstup náš, kontroluje, jestli `nonce`, obsažené ve vstupu je menší než číslo, které se nachází ve zdroji. Pokud je tato podmínka splněna, funkce zahlásí chybu a skončí.

5.7 Ověřování transakce v klientu

Projekt FITCoin se skládá ze dvou částí: klient `ftctl`, který dostává příkazy od uživatele a daemon `ftcd`, který na těchto příkazech poslouchá a provádí příslušné operace.

Pro možnost ověření správnosti transakce přes klienta byly do souboru `msg.h` přidané další typy zpráv: `MSG_CHECKTX_REQ` a `MSG_CHECKTX_RES` a do souboru `msg.c` byly přidané funkce pro jejích vytváření: `mkmsg_checktx_req` a `mkmsg_checktx_res` .

V souboru `ftctl.c` je definován příkaz `checktx`, po jehož zavolání klient pošle zprávu daemonu, dostane zpět výsledek a zobrazí ho. Za posílání zprávy odpovídá funkce `mkreq_checktx`, která vezme hash transakce, ověří jestli má správný formát a pošle zprávu. Funkce `prres_checktx` má za úkol zobrazit výsledek příkazu uživateli. Funkce dostane zprávu, získá z ní `status`. Pokud `status` je 0, pak je transakce validní.

Pro zpracovaní požadavku od `ftctl` v souboru `ftcd.c` je definován handler, který dostane hash transakce jako vstup, pomocí funkce `gettx` ji přečte ze složky `freshdir` a zavolá na ní funkci `checktx`. Kód handleru je uveden níže:

```
struct msg*
handle_checktx(const char* reqdata, uint32_t size)
{
        struct tx* tx;
        if (0 == size)
                return NULL;
        if (NULL == (tx = gettx((const char*) reqdata,
                                        freshdir))) {
                return mkmsg_fail(MSG_CHECKTX_RES);
        }

        uint32_t check = checktx(tx);
        return mkmsg_checktx_res(check);
}
```

Příklad použití příkazu `checktx`:

```
$ echo "checktx 6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49
c01e52ddb7875b4b" | ./ftctl
transaction is valid
```

Kapitola 6

Testování

Součástí zadání bylo i testování výsledného řešení. Testování jednotlivých komponent bylo prováděno pomocí unit testů.

6.1 Scénáře

V rámci testování byly otestované následující scénáře.

- Ověření podpisu u vstupu bez podpisu
- Ověření správného podpisu
- Zamítnutí podpisu chybné délky
- Zamítnutí chybného podpisu správné délky
- Zamítnutí transakce, obsahující chybný vstup
- Kontrola transakce, obsahující pouze správné vstupy
- Simulování replay útoku

6.2 Struktura testů

Pro účely testování se v souboru `Makefile` nachází speciální target `test`, který odpovídá za spouštění unit testů. Testy jsou uložené v souborech `test-*.c`. Test obsahuje podmínky, které ověří, jestli se testovaný objekt nachází v očekávaném stavu. Pokud tomu tak není, test skončí s chybovým kódem 1. Dále jsou uvedené názvy jednotlivých souborů a části projektu, které každý z nich testuje.

6.2.1 test-io.c

Soubor `test-io.c` má za účel otestovat všechnu funkcionalitu, která souvisí se vstupy a výstupy. V rámci unit testu se testuje připsání mincí/výběr z adresy, podepisování vstupu a následné ověření podpisu. Testuje se taky, jestli nepodepsané vstupy, podpisy se špatnou délkou a poškozené podpisy budou zamítnuté. Jeden z testů kontroluje ochranu proti replay útoku.

6.2.2 test-tx.c

V tomto souboru se nachází testy, které testují funkce související s transakcemi. V rámci testů jsou vytvářené správné a špatné transakce, které jsou potom ověřované pomocí funkce `checktx`.

6.2.3 test-wallet.c

V souboru `test-wallet.c` se nachází testy pro peněženku. Testuje se vytvoření nových adres, načítání klíčů a jejich dobití. Další testy ověří, jestli jsou adresy v peněžence správně seřazené a jestli peněženkou nově vytvořená transakce odpovídá očekávanému výsledku.

6.2.4 test-block.c

Soubor `test-block.c` testuje funkcionalitu související s bloky. Testuje se vytváření genesis a obyčejných bloků, kontroluje se, že struktura bloku, uložená na disku je správná.

6.3 Manuální testování

Kromě unit testů, byl zkompilovaný program spuštěn ručně a bylo vytvořeno několik testovacích transakcí. Po tom, co byly tyto transakce vytěžené do bloků, se mince objevily na příslušných adresách. Všechny struktury, vytvořené peněženkou byly prozkoumané pomocí nástroje hexdump a osvědčila se jejich správnost.

Závěr

Tato práce se většinou věnovala kryptografii a jejímu významu pro kryptoměny. V rámci bakalářské práce byla nastudována potřebná teorie týkající se asymetrické kryptografie a technologie blockchain obecně. Byla provedena rešerše kryptografických knihoven a zvolena knihovna OpenSSL/LibreSSL, jako nejvhodnější pro účely projektu FITCoin.

Získané znalosti umožnily úspěšně implementovat mechanismus digitálního podpisu a zabezpečit utrácení prostředků pouze vlastníkem soukromého klíče. Přidaná funkcionalita byla otestovaná v rámci unit testů a dá se považovat za funkční.

Dalším vylepšením může být použití algoritmu pro zotavení veřejného klíče přímo z digitálního podpisu, podobně jako u měny Ethereum, nebo oddělení digitálního podpisu od vstupu a použití analogu segregovaného svědka (SegWit) u Bitcoinu.

Literatura

[1] *Bitcoin hits $1 trillion market cap, surges to fresh all-time peak* [online]. Reuters. [vid. 10. 3. 2021]. Dostupné z: `https://www.reuters.com/article/us-crypto-currency-bitcoin-idUSKBN2AJ0GC`

[2] BASHIR, Imran. *Mastering Blockchain: Distributed ledger technology, decentralization, and smart contracts explained*. [2. vyd.]. Birmingham: Packt Publishing Ltd, 2018. ISBN 978-1-78883-904-4.

[3] FRANCO, Pedro. *Understanding Bitcoin: Cryptography, Engineering and Economics*. Chichester: John Wiley & Sons Ltd, 2014. ISBN 978-1-11901-916-9.

[4] OPPLIGER, Rolf. *Contemporary cryptography*. [2. vyd.]. Norwood: Artech house, 2011. Artech house computer security series. ISBN 978-1-60807-145-6.

[5] HOFFSTEIN, Jeffrey, Jill PIPHER and Joseph H. SILVERMAN. *An introduction to mathematical cryptography*. New York: Springer, 2008. ISBN 978-0-387-77993-5.

[6] *About cryptlib Security Software | Dr Peter Gutmann* [online]. cryptlib. [vid. 23. 4. 2021]. Dostupné z: `https://www.cryptlib.com/security-software/about-cryptlib`

[7] GUTMANN, Peter. *cryptlib Security Toolkit Version 3.4.5* [online]. [vid. 23. 04. 2021]. Dostupné z: `https://www.cryptlib.com/downloads/manual.pdf`

[8] *index* [online]. The GnuPG Project. [vid. 24. 4. 2021]. Dostupné z: `https://gnupg.org/software/libgcrypt/index.html`

[9] *Cryptographic Functions (The Libgcrypt Reference Manual)* [online]. The GnuPG Project. [vid. 24. 4. 2021]. Dostupné z: `https:`

//gnupg.org/documentation/manuals/gcrypt/Cryptographic-Functions.html#Cryptographic-Functions

[10] *GNUTLS* [online]. GNUTLS. [vid. 27. 4. 2021]. Dostupné z: https://www.gnutls.org/index.html

[11] MAVROGIANNOPOULOS, Nikos et al. *GnuTLS manual* [online]. Nikos Mavrogiannopoulos. ISBN 978-1-326-00266-4. [vid. 27. 04. 2021]. Dostupné z: https://www.gnutls.org/manual/gnutls.pdf

[12] *wolfCrypt Embedded Cryptography Engine | wolfSSL Products* [online]. wolfSSL Inc. [vid. 28. 4. 2021]. Dostupné z: https://www.wolfssl.com/products/wolfcrypt-2/

[13] *Algorithms - ECC* [online]. wolfSSL Inc. [vid. 28. 4. 2021]. Dostupné z: https://www.wolfssl.com/doxygen/group__ECC.html

[14] */index.html* [online]. OpenSSL Software Foundation. [vid. 1. 5. 2021]. Dostupné z: https://www.openssl.org/

[15] */docs/man1.1.0/man3/ECDSA_sign.html* [online]. OpenSSL Software Foundation. [vid. 1. 5. 2021]. Dostupné z: https://www.openssl.org/docs/man1.1.0/man3/ECDSA_sign.html

[16] *LibreSSL* [online]. OpenBSD Foundation. [vid. 1. 5. 2021]. Dostupné z: https://www.libressl.org/

[17] *Mbed TLS - Trusted Firmware* [online]. Linaro Limited. [vid. 1. 5. 2021]. Dostupné z: https://www.trustedfirmware.org/projects/mbed-tls/

[18] *ecdsa.h File Reference - API Documentation - mbed TLS (Previously PolarSSL)* [online]. ARM Limited. [vid. 1. 5. 2021]. Dostupné z: https://tls.mbed.org/api/ecdsa_8h.html

[19] *libtom* [online]. Team libtom. [vid. 2. 5. 2021]. Dostupné z: https://www.libtom.net/LibTomCrypt/

[20] *LibTomCrypt Developer Manual* [online]. LibTom Projects. [vid. 2. 5. 2021]. Dostupné z: https://github.com/libtom/libtomcrypt/releases/download/v1.18.2/crypt-1.18.2.pdf

Příloha A

Seznam použitých zkratek

EC Elliptic curve

ECC Elliptic curve cryptography

DSA Digital Signature Algorithm

ECDSA Elliptic Curve Digital Signature Algorithm

DLP Discrete Logarithm Problem

ECDLP Elliptic Curve Discrete Logarithm Problem

SHA Secure Hash Algorithms

SSL Secure Sockets Layer

TLS Transport Layer Security

DTLS Datagram Transport Layer Security

DER Distinguished Encoding Rules

API Application Programming Interface

PKCS Public Key Cryptography Standards

RSA Rivest–Shamir–Adleman

EdDSA Edwards-curve Digital Signature Algorithm

ECDH Elliptic-curve Diffie–Hellman

PRNG Pseudorandom Number Generator

LGPL The GNU Lesser General Public License

GPL The GNU General Public License

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `fitcoin.zip` ..... archív se zdrojovými kódy projektu
- `thesis` ..... adresář se zdrojovým kódem práce ve formátu LaTeX
- `thesis.pdf` ..... text práce ve formátu PDF