VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

SEMI-CENTRALIZOVANÁ KRYPTOMĚNA ZALOŽENÁ NA BLOCKCHAINU A TRUSTED COMPUTING

SEMI-CENTRALIZED CRYPTOCURRENCY BASED ON THE BLOCKCHAIN AND TRUSTED COMPUTING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JAKUB HANDZUŠ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. IVAN HOMOLIAK, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Handzuš Jakub, Bc.**
Program: Informační technologie
Obor: Bezpečnost informačních technologií
Název: **Semi-centralizovaná kryptoměna založená na blockchainu a trusted computing**
Semi-Centralized Cryptocurrency Based on the Blockchain and Trusted Computing
Kategorie: Bezpečnost
Zadání:

1. Study principles of blockchains and trusted computing.
2. Study approaches that combine blockchain with trusted computing and make a comparison of them.
3. Propose the model of a semi-centralized cryptocurrency that supports external interoperability.
4. Make the proof-of-concept implementation of the proposed model and evaluate its features.
5. Perform a security analysis of the approach.
6. Discuss challenges and options for real-world applications (such as a set of national banks).

Literatura:

- Allen, Sarah, et al. *Design Choices for Central Bank Digital Currency: Policy and Technical Considerations*. No. w27634. National Bureau of Economic Research, 2020.
- Homoliak, Ivan, and Pawel Szalachowski. "Aquareum: A Centralized Ledger Enhanced with Blockchain and Trusted Computing." *arXiv preprint arXiv:2005.13339* (2020).

Při obhajobě semestrální části projektu je požadováno:

- Items 1 to 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Homoliak Ivan, Ing., Ph.D.**
Konzultant: Perešíni Martin, Ing., UITS FIT VUT
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Cieľom práce je vytvoriť koncept semi-centralizovanej kryptomeny, ktorá podporuje externú interoperabilitu. Predpokladá sa, že práve semi-centralizovaná kryptomena je budúcnosť kryptomien v bankovom sektore, nakoľko aj za cenu čiastočnej centralizácie, prináša výhody konceptu decentralizovanej účtovej knihy. Keďže možno predpokladať súbežné nasadenie vlastných kryptomien rôznymi centrálnymi autoritami, ako napr. centrálne banky, je nutné vytvoriť komunikačný protokol pre medzibankové transakcie. Z toho dôvodu je práca zameraná na rozšírenie existujúceho riešenia Aquareum o interoperabilitový protokol.

Abstract

The aim of this thesis is to create a concept of semi-centralized cryptocurrency that supports external interoperability. It is assumed that semi-centralized cryptocurrency is the future of cryptocurrencies in the banking sector, because even at the cost of partial centralization, the concept brings the benefits of a decentralized ledger. Since the simultaneous deployment of their own cryptocurrencies by various central authorities, such as central bank, it is necessary to establish a communication protocol for interbank transactions. The work is thus focused on extending the existing Aquareum solution with an interoperability protocol.

Kľúčové slová

Trusted computing, trusted execution enviroment, enkláva, Intel SGX, blockchain, kryptomena, interoperabilita, Aquareum, smart kontrakty.

Keywords

Trusted computing, trusted execution enviroment, enclave, Intel SGX, blockchain, cryptocurrency, interoperability, Aquareum, smart contracts.

Citácia

HANDZUŠ, Jakub. *Semi-centralizovaná kryptoměna založená na blockchainu a trusted computing*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Ivan Homoliak, Ph.D.

Semi-centralizovaná kryptoměna založená na blockchainu a trusted computing

Prehlásenie

Prehlasujem, že som túto prácu vypracoval samostatne pod vedením pána Ing. Ivana Homoliaka, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Jakub Handzuš
14. mája 2021

Poďakovanie

Rád by som vyjadril úprimnú vďaku vedúcemu práce, pánovi Ing. Ivanovi Homoliakovi, Ph.D. za poskytnuté konzultácie, podporu a odborné vedenie práce. Zároveň by som sa rád poďakoval aj konzultantovi Ing. Martinovi Perešínimu, za poskytnutie veľmi užitočnej konzultácie.

Obsah

Kapitola 1

Úvod

Technológia blockchain zažila za poslednú dekádu dobu neuveriteľného záujmu. Táto technológia stojí za počiatkom veľkej zmeny vnímania platidiel, zavedením nového typu virtuálnej meny – tzv. kryptomeny. Kryptomeny sú oproti bežným platidlám založené na modernej kryptografii a decentralizovanosti, vďaka čomu prinášajú vlastnosti ako nezmeniteľnosť a nezvratnosť vykonaných transakcií. Práve z toho dôvodu, ponúkané vlastnosti prilákali množstvo technologických nadšencov a investorov, pričom čoraz viac sú kryptomeny využívané ako alternatíva bežných platidiel. Avšak momentálne, kvôli svojím obmedzeniam nie sú vhodné ako úplná náhrada aktuálneho spôsobu platidiel, v dôsledku čoho je táto technológia predmetom neustálych výskumov, ktoré sa zameriavajú na odstránenie nedostatkov, čo by umožnilo rozvinúť jej plný potenciál.

Jedným z možných postupov riešenia je upustenie od myšlienky decentarlizovanosti pri zachovaní všetkých ostatných vlastností. Takéto riešenie môže byť nasadené centralizovanými subjektami – bankami. Spomínané vlastnosti je možné v centralizovanom prostredí zabezpečiť technológiou trusted computing. Príkladom centralizovaného riešenia je projekt Aquareum [32], ktorý okrem technológie trusted computing využíva aj decentralizovanú technológiu blockchain.

Práca sa zameriava na rozšírenie spomínaného centralizovaného riešenia o možnosť externej interoperability. Nasadením viacerých inštancií tak vzniká semi-centralizovaná kryptomenová platforma.

Štruktúra dokumentu. Teoretická časť práce je členená do troch kapitol. V kapitole 2 je rozpracovaný úvod do problematiky dôveryhodného počítania (anglicky trusted computing), vrátane popisu vývoja technológie, jej prínosu ako aj prehľad aktuálnych prostredí určených pre zabezpečené vykonávanie kódu. V danej kapitole sa kladie dôraz najmä na technológiu Intel SGX. Kapitola 3 definuje technológiu blockchain – jej vznik, vlastnosti, štrukturálne rozdelenie, vrátane jej využitia v kryptomenách. Rovnako sú v príslušnej kapitole uvedené špecifikácie dvoch najznámejších kryptomien – Bitcoin a Ethereum. V kapitole 4 je priestor venovaný zhodnoteniu a porovnaniu prístupov využívajúcich kombináciu trusted computingu a blockchain technológie. Na základe nadobudnutých poznatkov je v kapitole 5 vytvorený návrh komunikačného protokolu zabezpečujúci externú interoperabilitu navrhovanej semicentralizovanej kryptomeny založenej na platforme Aquareum [32]. Kapitola 6 zobrazuje implementačné detaily jednotlivých častí navrhovaného systému. Následujúca kapitola 7 analyzuje bezpečnosť protokolu pri rôznych pozíciach útočníka.

Kapitola 2

Trusted Computing

Aplikácie čoraz častejšie pracujú so súkromnými informáciami, ako sú heslá, citlivé informácie užívateľov, čísla účtov, kryptografické kľúče, atď. Prístup k týmto dátam nesmie nadobudnúť nikto iný okrem príjemcu. Úlohou operačného systému je presadzovanie takých bezpečnostných politík, aby sa k týmto tajomstvám nemohol dostať neoprávnený subjekt. Operačný systém bráni prístup k súborom iného užívateľa (ak nebolo explicitne uvedené inak), jednej aplikácie v prístupe do pamäte druhej a neprivilegovanému používateľovi v prístupe k prostriedkom OS. Aby nedošlo k úniku informácií pri komunikácií aplikácie s úložiskom alebo inou entitou po sieti, dáta sú šifrované.

Avšak aj napriek týmto opatreniam nemá aplikácia záruku, že sa malvér nedokáže dostať k citlivým informáciám. Pretože každý program, ktorý získa oprávnenia správcu systému, získa zároveň prístup aj ku všetkým zdrojom vrátane dát. Preto je vhodné použiť na ochranu nové architektúry, ktoré zabezpečia dôveru aplikačných dát za akýchkoľvek podmienok. V kapitole sú uvedené stručné informácie o doposiaľ vyvíjaných technikách, pričom najväčšia pozornosť je upriamená na technológiu Intel SGX.

Anglické slovo *trust* – dôverovať, môže nadobúdať rôzny význam v závislosti na kontexte ako aj v odlišných vnímaniach rôznych ľudí. V spojení so slovom *computing* – počítanie, sa môže jednať o pojem, ktorý je bez ďalšieho dodatočného vysvetlenia bezvýznamný. Zjednodušene, v počítačovej bezpečnosti je koncept dôveryhodného počítania (*Trusted Computing* – **TC**) možné považovať za taký počítačový systém, ktorého entity majú určitú úroveň záruky takú, že jeho časť, prípadne celok počítačového systému, sa chová očakávaným spôsobom. Za entitou je možné považovať človeka počítačového systému, alebo program vykonávajúci na vzdialenom stroji, pričom stupeň záruky zabezpečenia môže pokrývať všetky aspekty systému, alebo iba jeho časť [43].

Ešte pred ďalším výkladom problematiky v tejto kapitole je potrebné uviesť a popísať bezpečnostné vlastnosti, ktoré väčšina architektúr TC nadobúda [40].

Izolácia. Označuje hardvérový mechanizmus, ktorý riadi prístup k softvéru a jeho pridruženým dátam. Ak je takýto softvér spolu s dátami umiestnený v špeciálne chránenom module, je odizolovaný od zbytku systému, čím znemožňuje neoprávnené čítanie/zápis dát a modifikáciu samotného kódu iným softvérom. Vďaka tejto vlastnosti je zaručená integrita ako aj kódu, tak aj údajov, ktoré sa nachádzajú v module. Z bezpečnostného hľadiska je vykonávanie takého kódu možné spustiť iba z jedného predom definovaného miesta. Chránené moduly sú používané na ukladanie dôverných informácií, ako napr. tajné kľúče.

Atestácia. Mechanizmus preukazovania identity a dôveryhodnosti cieľovej platformy tretej strane za účelom získania jej dôvery ešte pred samotnou interakciou. Existujú dva

druhy atestácie – lokálna, pri ktorej sa preukazujú jednotlivé moduly v rámci jednej platformy a vzdialená na preukazovanie mimo jednej platformy [38].

Šifrovanie dát *(anglicky sealing)*. Šifrovanie dôverných dát alebo kódu tak, že ich následné rozšifrovanie je možné vykonať iba za určitých podmienok (dáta sú rozšifrované na konkrétnom zariadení, v konkrétnej konfigurácií alebo stav zariadenia je rovnaký ako pri šifrovaní).

Odolnosť voči útokom na postranné kanály. Žiaden softvérový modul, vrátane privilegovaného operačného systému, nedokáže získať informácie z iných modulov inak, ako z ich vstupno-výstupného rozhrania. Architektúra by sa tak mala postarať o vymazanie vyrovnávajúcich pamätí, aby z nich nedochádzalo k únikom informácií pri prepínaní kontextu.

Ochrana pamäte. Ochrana integrity a autenticity dát ukladaných do externej pamäti alebo pri ich odosielaných cez systémové zbernice.

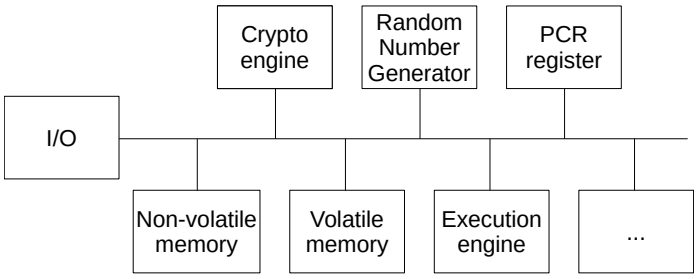
Už v 80-tych rokoch sú Ministerstvom obrany Spojených štátov amerických definované kritériá hodnotenia spoľahlivosti počítačových systémov (TCSEC), označované aj ako *Oranžová kniha*. Tento dokument definoval pojem *báza dôveryhodného počítania* (*Trusted Computing Base* – **TCB**) ako súhrn ochranných mechanizmov v rámci počítačového systému vrátane hardvéru, firmvéru a softvéru, ktorých kombinácie mechanizmov zodpovedajú za presadzovanie bezpečnostných politík systému. TCB sa môže skladať z jedného alebo viacerých komponentov spoločné vynucujúcich jednotnú bezpečnostnú politiku. Schopnosť dôveryhodnej výpočtovej základne správne vynútiť bezpečnostnú politiku závisí výlučne od mechanizmov v TCB a od správneho zadania parametrov administrátormi systému [6].

Ďalej sa TCSEC kritériá zameriavajú z väčšej časti na bezpečnosť operačných systémov. Avšak rovnako podstatnú časť zohráva aj hardvér, keďže izolácia pamäťových stránok je hlavnou myšlienkou zabezpečenia kontextu v režime jadra a aplikácie. Z hľadiska operačného systému je hardvér dôveryhodný, pretože neexituje alternatívny spôsob testovania a overovania správnosti hardvéru. Avšak to neznamená, že hardvér nemôže byť kompromitovaný čo naznačujú aj nedávne exploity ako Meltdown a Spectre[1] [30].

Hrozby zranitelnosti hardvéru motivovali počítačový priemysel na vytvorenie skupiny *Trusted Computing Group* (**TCG**), ktorá vzniká v roku 2003 a v roku 2006 prichádza s prvou generáciou modulu dôveryhodného počítania (*Trusted Platform Module* – **TPM**). Modul slúži na zaistenie dôveryhodnosti platformy, ktorej môžu dôverovať lokálni užívatelia aj vzdialené entity. Princíp budovania dôvery je založený na koncepte reťazca dôvery (*chain ot trust*). Koncept spočíva v postupnom overovaní jednotlivých komponentov (hardvérových aj softvérových) predošlým komponentom v rade, počínajúc od koreňa dôvery (*root of trust*), až po hardvérovú platformu, operačný systém a aplikácie. Koreň dôvery je systéme implementovaný ako mikroradič, ktorý je umiestnený na základnej doske. TMP poskytuje systému (1) možnosť generovať a ukladať kryptografické kľúče, a to aj s využitím pravého generátora náhodných čísel, (2) šifrovanie (*sealing*) a viazanie (*binding*), (3) vzdialenú atestáciu. Jeho zjednodušená architektúra je zobrazená na obrázku 2.1. Avšak je nutné podotknúť, že TPM poskytuje obmedzenú ochranu voči fyzickým útokom [49, 40, 34].

Hlavným nedostatkom TPM modulu je, že neponúka izolovane prostredie pre vykonávanie ľubovoľného kódu dostupného pre tretie strany, čím znižuje funkčnosť na predom definovanú množinu rozhraní. Riešením je vytvorenie obmedzeného prostredia, ktoré je odolné voči neoprávnenej manipulácií. Takéto prostredie sa nazýva ako prostredie dôveryhodného vykonávania (*Trusted Execution Environment* – **TEE**) a zaručuje:

[1] https://spectreattack.com/



Obr. 2.1: Zjednodušená architektúra TPM komponenty (prevzaté z [4]).

- autenticitu vykonávaného kódu
- integritu aktív (registre procesora, pamäť, vstupno-výstupné operácie)
- dôvernosť kódu, dát a aktív

Okrem toho musí byť schopné poskytnúť vzdialené osvedčenie, ktorým preukazuje jeho dôveryhodnosť tretím stranám. TEE musí odolávať všetkým známym softvérovým útokom ako aj externým útokom na hardvér, najmä na hlavnú pamäť systému [48].

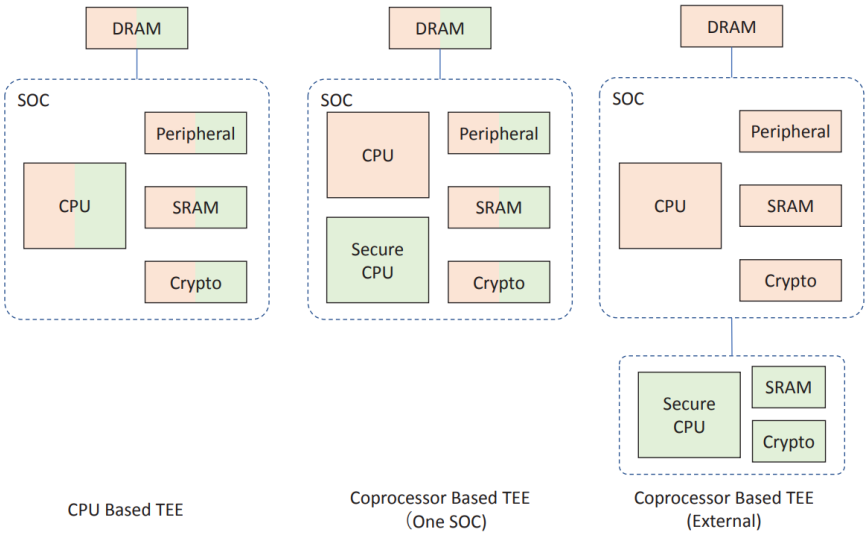
Všetky spomínané vlastnosti zaručené najmä vďaka separačnému jadru TEE, ktoré sa snaží ochrániť aktíva od zbytku prostredia systému REE.[2] Z pohľadu TEE sa jedná o nedôveryhodné prostredie, a preto je komunikácia s týmto prostredím realizovaná pomocou dôkladne kontrolovaného rozhrania. Rozhranie musí byť dostatočne zabezpečené aby nevznikali možnosti potencionálnych útokov, ako napr. útok preťažením správ, útok na kontrolu poškodenia správ, chyby pamäte spôsobené odstránením zdieľaných stránok. Aj preto sú na TEE kladené bezpečnostné požiadavky, ktoré pozostávajú z nasledujúcich bezpečnostných politík:

- Oddelenie dát – dáta v jednom oddieli nemôžu čítať ani upravovať dáta v inom oddieli.
- Časové oddelenie – zdielané zdroje nemôžu byť použité na únik informácií do iných oddielov.
- Kontrola toku – komunikácia medzi oddielmi môže prebiehať len vtedy, keď je explicitne povolené.
- Izolácia chyby – Porušenie bezpečnosti sa nesmie šíriť naprieč oddielami [48].

Zároveň TEE musi byť inštancované procesom bezpečného zavedenia, ktoré je izolované od REE. Tento proces pozostáva z viacerých fáz počas ktorých sa zakladá reťazec dôvery. Na základe toho nadobudne integritu a autenticitu ktorú si ponecháva počas celej svojej životnosti. TEE ďalej pristupuje k vstupno-výstupným portom pomocou dôveryhodnej cesty, čím chráni autenticitu a dôvernosť pri komunikácií s perifériami ako napr. klávesnicou, senzormi, atď.

Jednotka TEE sa môže v systéme implementovať troma rôznymi spôsoby v jej závislosti na lokácii v systéme. Tieto možnosti sú zobrazené na obrázku 2.2. Prvým spôsobom je začlenenie TEE priamo do hlavného procesora. Tento spôsob využívajú implementácie ako Intel SGX, ARM TrustZone alebo RISC-V Keystone. Druhou možnosťou je vytvorenie koprocesora vo vnútri jedného systému na čipe (SOC), ktorý zdieľa všetky časti SOC

[2] *Rich Execution Enviroment* (**REE**) – prostredie, ktoré poskytuje a riadi širší operačný systém [3].



Obr. 2.2: Možnosti lokácie TEE v systéme (prevzaté z [52]).

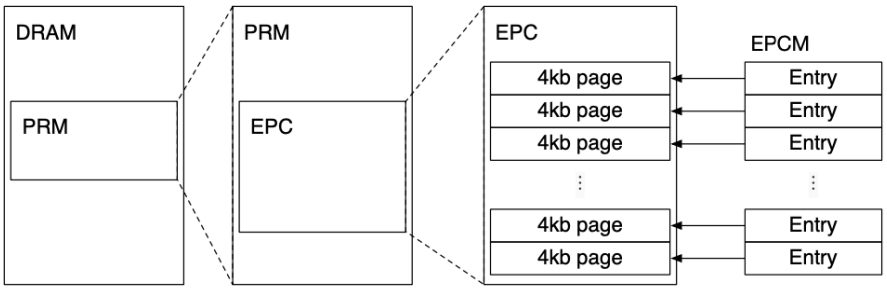
s hlavným procesorom. Predstaviteľ tohto spôsobu je Apple secure enclave processor (SEP), AMD Platform Secure Processor (PSP) alebo Intel Converged Security and Management Engine (CSME). Tretím spôsob je založený na samostatnom koprocesore, ktorý komunikuje zabezpečeným kanálom s hlavným procesorom, ale ďalej nezdieľa žiadne ďalšie časti systému [52].

V nasledujúcich podkapitolách sú rozobrané niektoré zo spomenutých implementácií TEE, pričom dôraz je kladený najmä na technológiu Intel SGX. Na konci kapitoly sú v tabuľke 2.1 porovnané rozoberané technológie.

2.1 Intel SGX

Technológia Intel Software Guard Extensions (SGX) prináša rozšírenie architektúry Intel procesorov o sadu nových inštrukcií, ktorých cieľom je poskytnúť záruku integrity a dôvernosti pre citlivé výpočty, a to aj v systéme, kde privilegovaný softvér (jadro, hypervízor, operačný systém) je potencionálne škodlivý. Systém využívajúci procesor s technológiou Intel SGX poskytuje TC na základe použitia dvoch princípov. (1) Nové inštrukcie poskytujú vývojárom aplikácií vytvoriť bezpečný a hardvérom izolovaný kontajner nazývaný *enkláva*, v ktorom sú vykonávané špecifické časti aplikácie operujúce nad citlivými dátami. Všetky dáta, s ktorými enkláva operuje, sú v šifrovanej podobe uložené v špeciálnej časti systémovej pamäti, ktorá je vďaka hardvérovým kontrolám dostupná len pre časti kódu vo vnútri enklávy. (2) Zároveň technológia Intel SGX implementuje atestáciu schémy softvéru, ktorá umožňuje, ako aj lokálnej, tak aj vzdialenej strane autentifikovať softvér, ktorý je vykonávaný vo vnútri enklávy (viac v sekcii 2.1.3) [40, 5, 17].

Intel prichádza na trh s SGX procesormi v roku 2015 pri uvedení 6. generácie Intel Core procesorov. Okrem notebookových a stolných procesorov prichádza podpora technológie aj pre radu *Xeon* využívanú v serveroch. Inštrukčná sada SGX prvej verzie implementuje 18 nových inštrukcií, z toho 13 je používaných privilegovaným softvérom a 5 užívateľským softvérom. Najdôležitejšie inštrukcie sú popísané v podkapitole 2.1.2. V roku 2016 bola predstavená druhá verzia SGX [42]. Inštrukčný set sa v tejto verzii rozrástol o ďalších



Obr. 2.3: Organizácia pamäti enklávy (prevzaté z [17]).

6 inštrukcií, ktoré prinášajú najmä dynamickú správu pamäti enklávy. Avšak do roku 2020 nebol uvedený na trh žiaden procesor, ktorý by túto verziu podporoval [50].

2.1.1 Pamäť enklávy

Kód a dáta používané enklávou sú uložené vo vyrovnávajúcej pamäti stránok enklávy (*Enclave Page Cache* – **EPC**), ktoré sa nachádzajú v špeciálnej súvislej časti DRAM pamäte rezervovanej pre procesor (*Processor Reserved Memory* – **PRM**). PRM pamäť o veľkosti 128MB je vytvorená už pri zavádzaní systému. Pamäť EPC je rozdelená do 4 kb stránok, ktoré sú prístupne len v rámci enklávy alebo cez inštrukčnú sadu SGX. Ani systém v privilegovanom režime, a ani softvér v užívateľskom režime nedokáže priamo pristúpiť k tejto pamäti s úmyslom jej čítania alebo zápisu. Rovnako je znemožnený priamy prístup do pamäte (Direct Memory Access – DMA) využívanými perifériami systému. Organizácia pamätí je znázornená na obrázku 2.3 [45].

O dôvernosť dát ukladaných do pamäti je zodpovedná hardvérová jednotka *Memory Encryption Engine* (MME). Jedná sa o rozšírenie pamäťového kontroléra, ktoré implementuje šifrovanie a dešifrovanie dát na hranici procesora so systémovou zbernicou. Odpočúvaním systémovej zbernice tak nie je možné zistiť pôvodné dáta. Kľúče využívané jednotkou MME sú generované už pri štarte systému a sú uložené v procesore [26, 27].

Rozdelenie EPC na stránky umožňuje priraďovať jednotlivé stránky rôznym enklávam, čím dovoľuje ich súčasný beh na jednom systéme. Avšak každá stránka môže patriť iba jednej enkláve, v dôsledku čoho je komunikácia medzi enklávami prostredníctvom týchto stránok pamäte zakázaná.

Správu EPC stránok zabezpečuje ten istý softvér (operačný systém alebo hypervízor), ktorý spravuje aj zbytok fyzickej pamäte. Ale keďže prístup do EPC pamäte nie je dostupný v žiadnom režime, každé priradenie, či odstránenie stránky je vykonané pomocou SGX inštrukcií. Avšak softvér, ktorý spravuje pamäť nie je dôveryhodný, procesory SGX si musia uchovávať informácie o stránkach pamäte, aby pri rozhodovaní o alokáciu nedošlo ku kompromitácií dát. To by mohlo nastať, ak by sa systémový softvér rozhodol alokovať už alokovanú stránku. V takomto prípade SGX inštrukcia použitá na vykonanie zlyhá.

Informácie o vyrovnávajúcich stránkach enklávy sú mapované do štruktúry **EPCM** (*Enclave Page Cache Map*). EPCM reprezentuje pole s jedným záznamov pre každú stránku, v ktorom sú uložené nasledujúce informácie:

- Záznam o aktuálnom stave platnosti stránky reprezentovaný jedným bitom. SGX inštrukcia pri alokovaní stánky skontroluje, či je tento bit nastavený na hodnotu logickej

nuly, inak operáciu odmietne vykonať. Ak predpoklad platí, a teda stránka nie je pridelená žiadnej enkláve, pri prideľovaní sa nastaví tento bit na hodnotu logickej "1".

- Typ stránky, definovaný na základne použitej inštrukcie pri alokácií. Stránky, ktoré uchovávajú kód a údaje sa považujú za bežné typy (`PT_REG`). Ak stránka obsahuje podporné dátové štruktúry SGX, je označená ako špeciálny typ (`PT_SECS, PT_TCS, PT_VA`) Podporné dátové typy sú rozoberané v nasledujúcich odsekov.

- Identifikácia enklávy, ktorá využíva danú stránku. Táto informácia je využívaná v mechanizmoch, ktoré zabraňujú v prístupe k dátam inej enklávy, čím zabezpečujú izolačné vlastnosti. V štruktúre sa jedná o uloženie virtuálnej adresy SECS (viď. ďalší odsek). [5]

Samotné meta-informácie sú pre každú enklávu samostatne ukladané v riadiacej štruktúre enklávy (*Enclave Control Structure* – **SECS**). Každý SECS je uložený na vyhradenej EPC stránke s typom stránky `PT_SECS`. Tieto stránky sú výhradne používané len SGX procesorom a nie sú určené na mapovanie do adresného priestoru. SECS záznam je možné považovať za identitu enklávy, keďže prvý krok vytvorenia enklávy vedie na pridelenie EPC, ktorá bude slúžiť enkláve ako SECS. Pri deštrukcií enklávy je naopak vymazanie tejto stránky ako posledný krok.

SGX inštrukcie prijímajú na vstupoch virtuálne adresy. Vzhľadom na to, že SGX inštrukcie používajú SECS adresy na identifikáciu enkláv, je nutné aby systémový softvér vo svojich tabuľkách stránok vytváral záznamy ukazujúce na SECS enkláv. K samotným SECS záznamom sa systémový softvér nedostane, pretože sa tieto údaje nachádzajú v PRM časti pamäti. SECS stránky nie sú určené na to, aby sa mapovali vo virtuálnych adresových priestoroch ich enkláv a procesory SGX vyslovene bránia kódu enklávy v prístupe na stránky SECS [2].

Každá enkláva určuje rozsah adries vo virtuálnom adresovom priestore, ktoré potom používa na mapovanie kódu a citlivých údajov uložených na EPC stránkach. Tento priestor sa označuje ako lineárny adresný priestor enklávy (*Enclave Linear Address Range* – **ELRANGE**). Virtuálny priestor mimo ELRANGE je mapovaný na rovnaké adresy ako hostiteľský proces enklávy, a teda sa jedná o mapovanie do nedôveryhodnej pamäte mimo PRM. ELRANGE je špecifikovaná pomocou bázovej adresy a veľkosti v SECS štruktúre enklávy, ktoré musia spĺňať isté obmedzenia. Veľkosť rozsahu musí byť mocninou čísla 2 a báza musí byť zarovnaná na adresu s takouto veľkosťou. Na základe toho je operácia overenia, či patrí daná adresa ELRANGE enkláve, implementovaná lacno v hardvéri aj softvéri.

Preklad virtuálnych adries na fyzické adresy pamäte DRAM sú v rámci systému s SGX vykonávané operačným systémom a hypervízorom, a teda aj tabuľky stránok aj rozšírené tabuľky stránok sú pod ich kontrolou. Kvôli minimalizovaniu zmien vyžadovaných na implementáciu podpory SGX existujúcim systémom, používa aj kód enklávy rovnaký proces prekladu adries. Tento proces je z pohľadu enklávy vykonávaný nedôveryhodným systémom a otvára potencionálne hrozby útokov na preklad adries.

Aby si SGX zachovalo svoje bezpečnostné vlastnosti, implementuje obranné mechanizmy zaisťujúce, že každú EPC stánku je možné mapovať len na konkrétnu virtuálnu adresu. Pri alokácií EPC stránky je zaznamenaná jej očakávaná virtuálna adresa do jej EPCM štruktúry. Potom, ak výsledkom prekladu adresy je fyzická adresa prislúchajúca EPC stránke, procesor sa uistí, že virtuálna adresa poskytnutá procesu sa zhodovala s očakávanou virtuálnou adresou uloženou v jej EPCM stránke. Procesor rovnako zabezpečí, že virtuálna pamäť vo vnútri ELRANGE je mapovaná výhradne na EPC stánky. V prípade, ak by táto sku-

točnosť nebola zabezpečená, útočník by mohol pozmeniť systémový softvér tak, že by celý virtuálny priestor enklávy mapoval na stránky DRAM mimo PRM časti pamäte. Keďže pre túto časť pamäte nie sú aplikované dodatočné kontroly, útočník by tak mal priamy prístup k citlivým dátam [17].

Technológia SGX umožňuje súčasné vykonávanie kódu viacerými vláknami v rámci jednej enklávy. Informácie o riadení každého logického procesora využívaného konkrétnou enklávou sa nachádzajú v samostatnej štruktúre (*Thread Control Structure* – **TCS**). Tieto dátové štruktúry musia byť zabezpečené vývojárom ešte pred samotným vytvorením enklávy, pričom ich minimálny počet musí odpovedať počtu súbežných vlákien, ktoré bude enkláva používať. Každá štruktúra je po vytvorení uložená na samostatnej EPC stránke typu `PT_TCS` a je ďalej nemenná. Podobne, ako pri SECS stránkach, ani do TCS stránok nie je možné pristúpiť priamo z enklávy, ktorej tieto stránky patria. Avšak pri použití ladiacich inštrukcií nastáva výnimka a enkláva schopná k tejto štruktúre pristúpiť. Medzi najdôležitejší údaj, ktorý obsahuje táto štruktúra, patrí informácia o vstupnom bode enklávy, ktorý je načítaný do hodnoty čítača inštrukcií pri spustení vykonávania kódu enklávy. [17, 26].

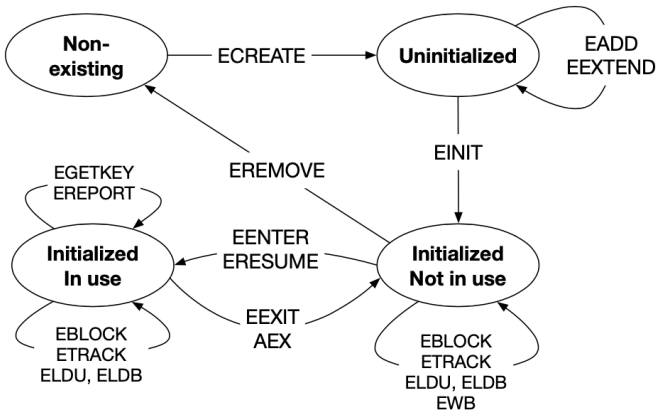
Každá štruktúra kontroly vlákien odkazuje na súvislú sekvenciu rámcov (*State Save Area* – **SSA**), ktorá je používaná na uloženie kontextu vykonávaného vlákna enklávy. Ukladanie nastáva pri prepínaní kontextu z dôvodu spracovania vzniknutej výnimky, alebo prerušenia. Pred opustením enklávy je jej stav zapísaný do sekvencie SSA. Pri návrate je tento uložený stav prečítaný, a kontext obnovený. Začiatok sekvencie spolu s počtom rámca je uložený v TCS. Každá rámec SSA sa skladá z niekoľkých EPC stránok, ktorých počet je definovaný v riadiacej štruktúre enklávy.

2.1.2 Životný cyklus enklávy

Enkláva môže počas svojho životného cyklu nadobúdať 4 základne stavy – (1) enkláva neexistuje, (2) existuje, ale je neinicializovaná, (3) inicializovaná, ale nepoužíva sa, (4) inicializovaná a používaná. Zmena stavov je prepojená so správou zdrojov, konkrétne s prideľovaním EPC stránok. Prideľovanie stránok je riadené (nedôveryhodným) systémovým softvérom, ktorý k týmto úkonom používa SGX inštrukcie. V nasledujúcej časti sú uvedené dôležité inštrukcie ovplyvňujúce stav enklávy, ktoré sú zároveň zakreslené v stavovom diagrame na obrázku 2.4.

Vytvorenie. Systémový softvér pomocou inštrukcie `ECREATE` vytvorí na voľnej EPC stránke novú identitu enklávy. Informácie zapísané na tejto stránke pochádzajú z nedôveryhodnej pamäti systému, a preto sú ešte pred zápisom validované. Keďže žiaden softvér nemá prístup k tejto stránke, ďalšie SGX inštrukcie, ktoré budú tento záznam využívať, ho nemusia ďalej validovať, a môžu sa spoľahnúť, že vďaka úvodnej validácií sú všetky dáta validné.

Načítanie. Po vytvorení je enkláva stále označovaná ako neinicializovaná. Pokiaľ je enkláva v tomto stave, systémový softvér pomocou volania inštrukcie `EADD` vytvára záznamy TCS, ako aj bežné záznamy, do ktorých sú následne načítané dáta a kód enklávy. Samotná inštrukcia zabezpečuje, že žiaden z vytvorených záznamov nebude priradený inej enkláve, a že virtuálna adresa stránky spadá do rozsahu enklávy. Po pridaní každého záznamu je možné pomocou inštrukcie `EEXTEND` definovať spôsob merania časti stránky, ktorý pri softvérovej atestácií slúži na dokázanie obsahu.



Obr. 2.4: Životný cyklus enklávy znázornený stavovým diagramom (prevzaté z [17]).

Inicializácia. Po načítaní obsahu enklávy je inštrukciou `EINIT`, s inicializačným tokenom, inicializovaná enkláva. Inicializačný token poskytuje privilegovaná enkláva, ktorá je podpísaná špeciálnym kľúčom, ktorého korešpondujúca verejná časť je na priamo zakódovaná do SGX implementácie. Tento proces inicializácie je nutný pre každú enklávu, ktorá nebola autorizovaná spoločnosťou Intel.

Po úspešnej inicializácií je napokon enkláva označená ako „inicializovaná“. To otvára možnosť aplikačnému softvéru spustiť vykonávanie kódu enklávy. Avšak v tomto stave už nie je možné vytvárať ďalšie stránky enklávy, a preto je vyžadované, aby všetky potrebné stránky boli priradené enkláve ešte pred samotným volaním inicializačnej inštrukcie.

Ukončenie. Na konci životného cyklu enklávy je systémovým softvérom vykonaná inštrukcia `EREMOVE`, ktorá ukončí enklávu spoločne so všetkými jej vytvorenými prostriedkami. V prvom kroku označí každú EPC stránku používanú enklávou za nevalidnú. Ešte pred ich uvoľnením inštrukcia overí, že žiaden z logických procesorov nevykonáva kód enklávy. Napokon, po dealokácií všetkých stránok, je uvoľnená SECS, čím zaniká aj samotná identita enklávy.

Synchrónny vstup. Od doby inicializácie enklávy, až po jej následné ukončenie, môže byť kód enklávy spúšťaný akýmkoľvek užívateľským procesom, ktorý má mapované EPC stránky enklávy vo svojom virtuálnom adresovom priestore. Systémový proces nemôže spúšťať kód enklávy.

Logický procesor je v režime enklávy, ak vykonáva jej kód. Kód vykonávaný logickým procesorom, je riadený štruktúrou TCS, ktorá zároveň zabezpečuje, aby žiadne dva procesory nepoužívali rovnaký TCS súčasne.

Zahájením vykonávania kódu predchádza inštrukcia `EENTER`, ktorá vykoná riadený skok na vopred preddefinované adresy. Preddefinované adresy zabezpečujú ochranu pred vynechaním bezpečnostných kontrol škodlivým softvérom. Pri vstupe do režimu enklávy sú niektoré registre uložené, aby mohli byť po vykonaní obnovené.

Synchrónny opustenie. Jedna z dvoch metód opustenia režimu enklávy. Na rozdiel od druhej metódy sa jedná o kontrolované opustenie režimu enklávy vykonávajúcim procesorom za použitia inštrukcie `EEXIT`. Inštrukcia deaktivuje režim enklávy a vyprázdni všetky

mapovania medzi virtuálnym a fyzickým adresovým priestorom, označí TCS ako neaktívny, obnoví obsah registrov uložených pri vstupe a prenesie riadenie do užívateľského procesu mimo enklávy na adresu, ktorá je špecifikovaná argumentom inštrukcie.

Asynchrónny opustenie. K asynchrónnemu opusteniu enklávy dochádza napr. pri vzniku hardvérovej výnimky alebo prerušeniu počas vykonávania kódu enklávy. V takomto prípade je, ešte pred vyvolaním obslužného programu, vykonaná inštrukcia `AEX`, pomocou ktorej je bezpečne uložený aktuálny stav enklávy do SSA rámca, a zároveň je tento stav nahradený pôvodným, ktorý bol uložený pri vstupe.

Pokračovanie vo výpočte. Potom, čo bola vykonaná obsluha udalosti, ktorá spôsobila opustenie enklávy, je kontext vrátený do obslužného programu nastaveného pri asynchrónnom opustení. Ten vykoná inštrukciu `ERESUME`, ktorá spôsobí, že sa logický procesor vráti späť do režimu enklávy a bude sa pokračovať vo výpočte, ktorý bol prerušený výnimkou [5, 17, 45].

2.1.3 Atestácia

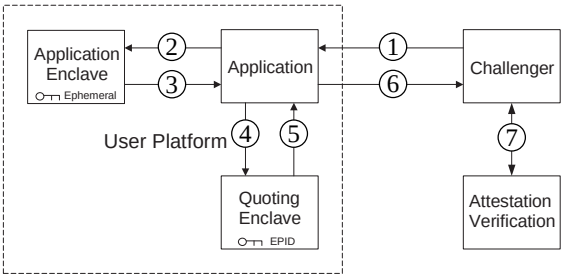
Zaistenie bezpečnosti systémov využívajúcich dôveryhodné procesory závisí aj na softvérovej atestácií. Jedná sa o mechanizmus, ktorý pre softvér bežiaci v izolovanom kontajneri vytvorenom dôveryhodným hardvérom, dokáže vytvoriť atestačný podpis na základe podpísania malej časti atestačných údajov. Atestácia zároveň zaručuje jednoznačnú identifikáciu softvéru bežiaceho v enkláve. Na základe týchto informácií je možné použiť atestačný podpis na presvedčenie overovateľa, že atestačné dáta boli vytvorené konkrétnym softvérom, ktorý je hostovaný vo vnútri kontajnera izolovaného dôveryhodným hardvérom [17].

Intel SGX architektúra poskytuje mechanizmus na vytvorenie atestačného podpisu medzi dvoma enklávami na jednej platforme. V takomto prípade hovoríme o lokálnej atestácii. Rozšírením tohto prístupu o poskytnutie podpisu tretím stranám vzniká vzdialená atestácia. Lokálna atestácia využíva systém symetrických kľúčov, pričom pre vzdialenú atestáciu je nutné použiť asymetrickú kryptografiu.

Vzdialená atestácia prebieha v nasledujúcich krokoch. Vyzývateľ kontaktuje aplikáciu s požiadavkou o preukázanie. Požiadavka spoločne s identitou *Quiting* enklávy je predaná enkláve aplikácie. Enkláva generuje manifest, ktorý obsahuje odpoveď na výzvu a verejný kľúč, ktorý použije vyzývateľ na komunikáciu s enklávou. Následne generuje hash manifestu a zahrnie ho spolu s užívateľskými dátami do inštrukcie `EREPORT`, ktorej výstupom je štruktúra `REPORT`, ktorá viaže manifest s enklávou. V ďalšiom kroku je štruktúra predaná do *Quiting* enklávy cez aplikáciu. Táto enkláva overí štruktúru `REPORT` a vytvorí podpísanú štruktúru `QUOTE`, ktorú následne vráti aplikácií. Aplikácia odošle vyzývateľovi štruktúru `QUOTE` a asociovaný manifest. V poslednom kroku vyzývateľ verifikuje nadobudnuté dáta a vytvorí verifikačný report. Na obrázku 2.5 je znázornený komunikačný diagram popisovanej vzdialenej atestácie [8].

2.2 ARM TrustZone

Z počiatku, technológia ARM TrustZone pozostávala z hardvérových bezpečnostných rozšírení zavedených do aplikačných procesorov Arm (*Cortex-A*) [7]. Od roku 2016 bola táto technológia upravená tak, aby podporovala aj radu Arm mikrokontrolérov (*Cortex-M*).



Obr. 2.5: Diagram komunikácie vzdialenej atestácie (prevzaté z [8]).

Dôležitým faktorom rastúceho záujmu o technológiu TrustZone bola podpora vývoja a výskumu od hlavných výrobcov hardvéru, spoločne s rastúcim počtom zariadení obsahujúcich ARM procesory. Tie sú využívané od najmä mobilných zariadeniach a vstavaných systémoch [46].

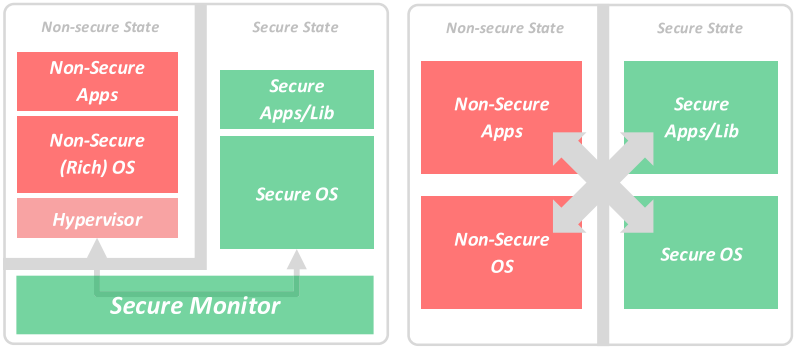
Koncept TrustZone je založený na rozdelení prostredia na dve asymetricky oddelené časti výpočtu, tzv. *svety*. Bezpečný svet, TEE, poskytujúci špecifické bezpečnostné vlastnosti a ochranu pre dôverný hardvér a softvérové zdroje. Normálny svet – REE, tvorí prostredie pre operačný systém a aplikácie. Bezpečnú časť systému je možné nakonfigurovať tak, aby mala prístup ku všetkým zdrojom, na druhú stranu, normálna časť nemôže za žiadnych okolností pristupovať k zdrojom bezpečnej časti. Izolácia zdrojov (hardvérových aj softvérových) je zabezpečená na základe hardvérovej kontroly prístupu. Procesor tak nemôže v jednej chvíli vykonávať kód oboch prostredí. To, v ktorom sa aktuálne nachádza, je definované pomocou bitu nezabezpečenia (*Non-Secure bit* – NS). Hodnota bitu je zapísaná v registri bezpečnej konfigurácie (*Secure Configuration Register* – SCR), a ďalej je propagovaná naprieč pamäťami a zbernicami [33].

Systémová zbernica, *AXI-bus*, je tak rozšírená o riadiaci signál, ktorý slúži pre propagovanie NS bitu pri prístupe k perifériám. Periférie na základe tohto signálu nesmú vykonať akciu, ktorá by mohla viesť k úniku citlivých dát. V prípade neoprávneného získania prístupu k zabezpečenej periférií, končí transakcia zlyhaním. [40]

Procesorové jadro v bezpečnej časti výpočtu generuje AXI transakcie s nulovou hodnotou NS bitu, ktoré vedú k získaniu zdrojov v oboch doménach bezpečnosti. Avšak procesorové jadro normálnej časti výpočtu je možné pristúpiť iba k zdrojom pochádzajúcej z rovnakej časti. Preto vyrovnávajúce pamäte ukladajú hodnotu NS bitu do adresového štítku pre každý riadok, čím vytvoria dva odlíšené pohľady na pamäťový priestor. Radiče vyrovnávajúcich pamätí zodpovedajú za to, aby len bezpečný proces vedel pristúpiť k bezpečnej časti. Rozšírením týchto vyrovnávajúcich pamätí odstraňuje potrebu vymazávania obsahu pri prepínaní kontextu.

Fyzické adresy v tabuľkách stránok sú taktiež rozšírené o hodnotu bitu NS, čo vytvára separáciu systémových pamätí na dve časti, každú pre jeden svet. Jadro procesora vynucuje nastavenie tohto bitu na nulu pri preklade adries normálneho sveta. [17].

Aby bolo možné vykonať prepnutie kontextu do druhého sveta, procesor musí najprv prejsť novým režimom nazývaným ako režim monitora. Tento režim slúži ako strážca pri prepínaní kontextu procesora medzi dvoma svetmi. Procesor je možné uviesť do tohto režimu pomocou novej privilegovanej inštrukcie, alebo vyvolaním hardvérovej výnimky či prerušenia. Procesor v režime monitora zaistí, že aktuálny stav sveta, z ktorého procesor odchádza, je uložený a stav sveta, do ktorého vstupuje, je obnovený. Tieto údaje zahrňujú všetky registre procesora a ďalšie informácie v závislosti od periférií. Procesory ARM



(a) TrustZone for Cortex-A (b) TrustZone for Cortex-M

Obr. 2.6: Separácia TrustZone hardvéru na dva svety (prevzaté z [46]).

Cortex-M nemajú žiaden režim monitora, a namiesto toho je spojenie medzi svetmi implementované do logiky jadra. Separácia svetov oboch spomínaných typov procesorov je znázornená na obrázku 2.6 [46].

Dokumentácia TrustZone neopisuje žiadnu implementáciu softvérovej atestácie, avšak načrtáva spôsob implementácie bezpečného zavádzania, pri ktorom je budovaný reťazec dôvery. Procesor po zapnutí štartuje do bezpečného sveta a začína overovať integritu jednotlivých častí zavádzajúcich systém. Firmvér prvej časti zavádzača, uložený v ROM pamäti, je považovaný za dôverný, a tak je ako prvá kontrolovaná integrita zavádzača druhej fázy uloženého vo flash pamäti. Po vykonaní úspešnej kontroly je zavádzač spustený. Posledné overenie integrity je vykonané na operačnom systéme pred jeho samotným spustením. Niektoré implementácie overujú aj integritu aplikácií pred ich spustením [40].

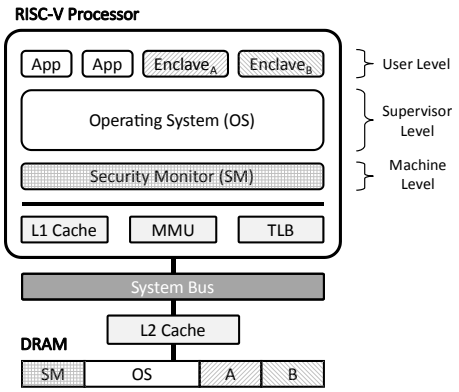
Oproti Intel SGX, Arm TrustZone nevynucuje šifrovanie obsahu pamätí. Preto, aj keď sú citlivé informácie uložené vo fyzickej pamäti chránenej TrustZone, útočník môže získať dáta prostredníctvom fyzických útokov na pamäte DRAM [55, 56].

2.3 Sanctum

V roku 2016 Costan a kol. [18] navrhli bezpečnostnú architektúru Sanctum, s cieľom vytvorenia dôveryhodného prostredia pre vykonávanie izolovaného výpočtu. Riešenie je postavené na architektúre RISC-V a zakladá na spoločnom návrhu hardvéru a softvéru pri minimálnej potrebe úpravy architektúry. Oproti predošlo-rozoberaným platformám TEE je Sanctum projektom s voľne šíriteľnými zdrojovými informáciami. Izolačná schéma je vo veľa ohľadoch podobná technológií Intel SGX a jej dizajn je znázornený na obrázku 2.7.

Rovanko ako SGX, Sanctum umožňuje spúšťať kód enklávy iba z užívateľského prostredia. Každá enkláva dostáva pridelenú samostatnú oblasť DRAM. Súčasne riadi a spravuje svoje vlastné tabuľky stránok a spracováva svoje chyby stránok, zatiaľ čo pri technológií SGX je riadenie stránok pod kontrolou operačného systému alebo supervízora.

Namiesto implementácie dôveryhodnej funkcionality do mikrokódu, používa Sanctum dôveryhodnú softvérovú komponentu nazývanú bezpečnostný monitor (SM). Ten poskytuje rozhranie pre správu enklávy ako napr. vytváranie a ukončenie enklávy. Zároveň spravuje prechody smerujúce do a z enklávy, čo znamená, že pre vstup, výstup je potrebné použiť špeciálne volania monitora. Monitor sa tiež stará o ukladanie aktuálneho stavu enklávy v prípade vzniku prerušenia. Avšak, pri obnove stavu enklávy, bezpečnostný monitor vstúpi



Obr. 2.7: Dizajn architektúry Sanctum (prevzaté z [22]).

do enklávy v jej vstupnom bode, a ďalšie obnovenie stavu prenecháva už na nej. Prostredie enklávy je tiež obmedzené, a preto je nutné pri systémových a vstupno-výstupných volaniach opustiť prostredie enklávy.

Dizajn Sanctum je zameraný hlavne na ochranu voči softvérovým útokom, vrátane softvérových útokov postrannými kanálami, ale neposkytuje žiadnu ochranu voči fyzickým útokom, keďže nemá k dispozícii žiaden modul šifrovania DRAM pamätí [17, 40].

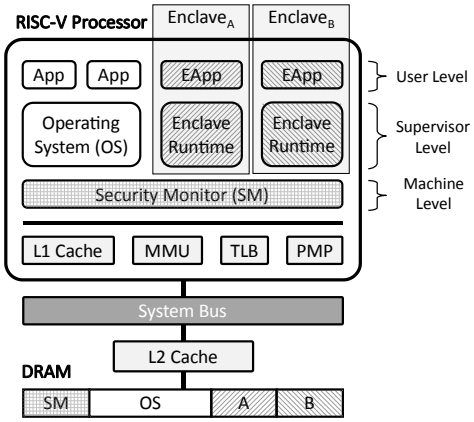
2.4 Keystone Enclave

V roku 2018 bol predstavený open-source projekt – Keystone Enclave, ktorý sa ako prvý zameriava na vytváranie prostredí pre dôveryhodné výpočty so zabezpečenými hardvérovými enklávami postavenými nad nemodifikovanou RISC-V architektúrou. Keystone prináša nové paradigma pre budovanie prispôsobiteľných TEE, kde poskytovatelia platforiem a vývojári enkláv prispôsobujú TEE tak, aby výsledná dôveryhodná výpočtová základňa bola čo najmenšia, ale zároveň nadobúdala vysokú mieru optimalizácie využitia zdrojov. To umožňuje uplatniť Keystone Enclave v rôznych aplikáciách od vstavaných systémov po strojové učenie [36].

Jedným z hlavných cieľov je dosiahnutie izolácie pamäte iba za použitia štandartných primitív architektúry RISC-V. Tým znižuje obmedzenia pre zavádzanie enkláv a umožňuje tak vyvíjať a spúšťať programy využívané túto technológiu bez potreby konkrétneho hardvéru. Taktiež je tým docielená možnosť spúšťania na virtuálnom stroji [44].

Konkrétne sa jedná o použitie ochrany fyzickej pamäte, ktorá umožňuje programovateľnému strojovému režimu prevádzkovanému na nižšej vrstve než operačný systém, špecifikovať ľubovoľné ochrany regiónov fyzickej pamäte. Strojový režim je využívaný na vykonávanie bezpečného monitora (SM) poskytujúceho bezpečné hranice.

Keystone dokáže spúšťať viacero enkláv. Každá enkláva pracuje vo svojej vlastnej izolovanej oblasti fyzickej pamäte a má svoju vlastnú komponentu (*runtime* – RT) v režime supervízora, na správu virtuálnej pamäte enklávy. Pomocou RT je možné implementovať akúkoľvek špecifickú funkcionalitu enklávy, zatiaľ čo SM spravuje záruky vynútené hardvérom. RT enklávy implementuje len požadovanú funkčnosť, komunikuje so SM a sprostredkováva komunikáciu s hostiteľom pomocou zdielanej pamäte a obsluhuje aplikáciu v enkláve. Popisované časti architektúry sú znázornené na obrázku 2.8 [52, 36].



Obr. 2.8: Dizajn architektúry Keystone (prevzaté z [22]).

Technológia	**Izolácia**	**Atestácia**	**Sealing**	**SCR**[3]	**MP**[4]	**Open-Source**	**Architektúra**
Intel SGX	✓	✓	✓	✗	✓	✗	x86_64
ARM TrustZone	✓	✗	✗	✗	✗	✗	ARM
Sanctum	✓	✓	✓	✓	✗	✓	RISC-V
Keystone Enclave	✓	✓	✓	✓	✓	✓	RISC-V

Tabuľka 2.1: Porovnanie popisovaných technológií [40, 46, 36].

[3]Ochrana voči softvérovým útokom postrannými kanálmi zameranými na prístup do pamäte.
[4]Ochrana pamäte voči fyzickým útokom.

Kapitola 3

Blockchain

Pod technológiou blockchainu rozumieme istý typ dátovej štruktúry – nazývaný tiež ako *digitálna účtová kniha*, ktorá je odolná proti neoprávnenej manipulácií a je implementovaná distributívnym spôsobom, zvyčajne bez centralizovanej entity. Zároveň je postavená na využití modernej kryptografie. Účtovú knihu si môžu používatelia prezerať a zároveň môžu vytvárať nové záznamy – transakcie. Po publikovaní transakcie do tejto knihy neexistuje možnosť následného odobrania.

Prvé úvahy vedúce na neskorý vznik technológie blockchain vznikali už na prelome 80-tych a 90-tych rokov minulého storočia. Tieto publikácie popisovali konsenzný model pre dosiahnutie dohody o výsledku v počítačovej sieti, kde jednotlivé uzly, alebo sieť samotná, môžu byť nespoľahlivé. [51]. Na základe týchto úvah vznikla už v roku 1991 prvá účtová kniha uchovávajúca digitálne podpísané dokumenty za účelom jednoduchého preukázania ich nemennosti [28].

Predošlé koncepty viedli skupinu, označovanú pseudonymom Satoshi Nakamoto, k definovaniu elektronickej hotovosti – kryptomeny nazývanej *Bitcoin* (podrobnejšie popísanú v sekcii 3.2.1). Publikácia bola zverejnená v až roku 2008, pričom samotná blockchainová sieť bola zavedená začiatkom nasledujúceho roku. Týmto Nakamoto zaviedol princípy, ktorým sa s miernymi úpravami riadi väčšina moderných kryptomien [51].

Technológia blockchain nie je výhradou len kryptomien, ale jej uplatnenie je možné nájsť v iných odvetviach ako napr. zdravotníctvo, potravinárstvo, volebné systémy atď. [41, 35, 31]. Avšak v niektorých aplikáciach nie je problém ich samotná implementácia, ale legislatívne zmeny s tým spojené [16]. V súčastnosti je technológia blockchainu využívaná hlavne v oblasti kryptomien (viac v sekcii 3.2) a decentralizovaných aplikácií.

Technológia blockchain je postavená na nasledujúcich princípoch [57]:

- Decentralizovanosť: Oproti bežným centralizovaným transakčným systémom, ktoré overujú transakcie pomocou centrálnej dôveryhodnej agentúry (napr. centrálnej banky), nie je v systémoch založených na technológii blockchain potrebné žiadnej ďalšej tretej strany, čím odpadávajú náklady spojené s udržiavaním centrálnych uzlov.

- Trvácnosť: Transakciu zapísanú do blockchainu je takmer nemožné vymazať alebo vrátiť späť. Pred každým zápisom sú transakcie kontrolované a neplatné sú okamžite zamietnuté.

- Anonymita, resp. pseudoanonymita: Používatelia v systéme vystupujú pod svojou vygenerovanou adresou, čím nie je odhalená ich skutočná identita. Avšak úplná ano-

nymita nie je zaručená, pretože existujú metódy, ktoré za istých okolností vedia odhaliť užívateľovu identitu.

- Auditabilita: Transakčná história vrátane aktuálneho stavu účtov jednotlivých používateľov je jednoducho dohľadateľná a vypočítateľná

Existujú dve všeobecné kategórie prístupov k účtovej knihe – bez oprávnenia a s oprávnením. V blockchainovej sieti bez oprávnenia môže ktokoľvek, a to aj bez autorizácie, pristúpiť k uloženým transakciám, a taktiež pridávať nové. A keďže sa väčšej časti jedná o voľne dostupný open-source softvér, nájdu sa aj taký používatelia, ktorí sa pokúšajú so zlým úmyslom zverejniť bloky podkopávajúce systém. Preto sú súčasťou blockchain siete mechanizmy na budovanie konsenzu (viac v sekcii 3.1.3). Na druhú stranu, siete s oprávnením vymedzujú vykonanie určitých akcií (prístup, vytváranie nových blokov) len pre istú autorizovanú skupinu používateľov. Autorizácia prebieha s centralizovanou alebo decentralizovanou autoritou. Aj tieto siete používajú modely pre budovanie konsenzu, avšak nevyžadujú rovnaké náklady na údržbu zdrojov, pretože užívateľom je pri porušení pravidiel odobratý prístup [23, 51].

3.1 Štruktúra

Na prvý pohľad sa môže zdať technológia blockchain komplikovaná, avšak táto problematika môže byť dekompovaná na samostatné jednoducho vysvetliteľné komponenty. V nasledujúcej časti sú preto vysvetlené jednotlivé dôležité komponenty vyskytujúce sa v každej blockchain sieti.

3.1.1 Blok

Ako už vyplýva zo samotného názvu technológie, jednou z najdôležitejších súčastí systému tvorí blok. Samotný blockchain je tak tvorený reťazcom týchto blokov.

Obsah blokov sa môže líšiť od samotnej implementácie, avšak z princípu musí každý blok obsahovať hlavičku a dáta. V hlavičke sa nachádzajú informácie o reťazení blokov (číslo bloku, časová pečiatka, hash hlavičky predošlého bloku) a údaje dokazujúce legitimitu bloku (hash dátovej časti bloku). Dáta nesú zoznam transakcií, prípadne iný typ dát. Väčšina účtových kníh obmedzuje veľkosť bloku, čím definuje maximálnu veľkosť dátovej časti.

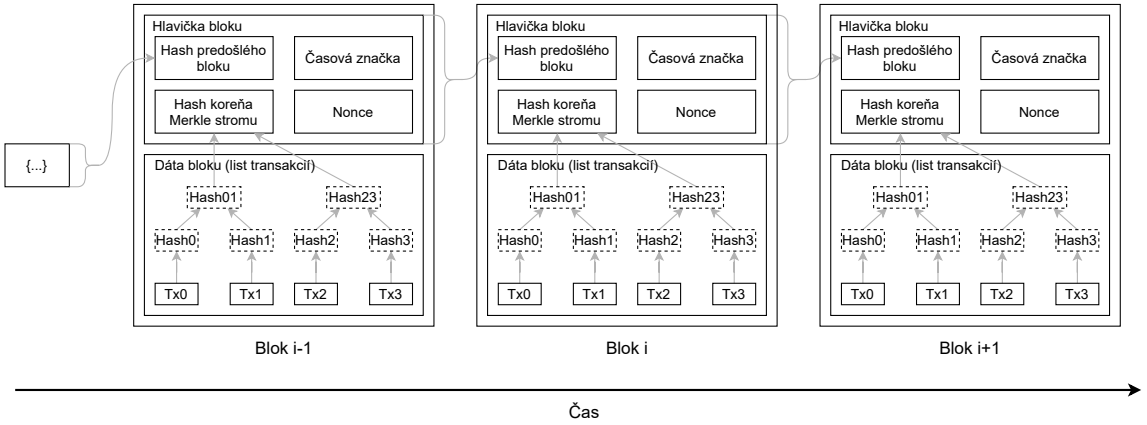
Bloky sú v reťazci usporiadané podľa času vytvorenia, pričom každý nový blok priradený do štruktúry musí ukazovať na posledný blok v reťazci. Výnimku tvorí prvý blok, označovaný ako *blok genézy*, ktorý nemá predchodcu. Takýto reťazec blokov je znázornený na obrázku 3.1.

Keďže posledný blok obsahuje hash hodnotu hlavičky predošlého bloku, zmena ľubovoľného predchádzajúceho bloku by tak viedla k reťazovej zmene ostatných blokov [51].

3.1.2 Uzol

Podstata decentralizovanej siete spočíva vo vzájomnej komunikácií uzlov medzi sebou (*peer-to-peer*). Uzol siete rozumieme ako individuálny systém, ktorý môže byť kategorizovaný na základe vykonávajúcej funkcie a obsahu uchovávaných dát.

Prvou kategóriou uzlov sú zjednodušené uzly, nazývané ako *lightweight nodes*. Tieto uzly dokážu vytvárať nové transakcie, ale keďže neuchovávajú a ani nespravujú celý reťazec blokov, vytvorené transakcie musia ďalej distribuovať po sieti do uzlov s kompletnou históriou.



Obr. 3.1: Ukážka jednoduchého reťazca blokov [51].

Uzly s kompletnou históriou siete, nazývané *full nodes*, uchovávajú celý reťazec blokov od samého počiatku vrátane obsahu všetkých dát blokov. Keďže tieto uzly obsahujú všetky informácie o uskutočnených transakciách, môžu vykonávať kontrolu ich validity. Istá podskupina týchto uzlov zároveň vytvára a zverejňuje nové bloky.

Vznik nového bloku závisí od konkrétnej implementácie. Najčastejšie však tento úkon nastáva pri získaní dostatočného počtu transakcií, alebo v pravidelne definovaných časových intervaloch. Po vytvorení nového bloku jedným z uzlov nastáva jeho distribúcia sieťou, pri ktorej musí dojsť k vytvoreniu konsenzu naprieč uzlami siete [51].

3.1.3 Mechanizmy na vytvorenie konsenzu

Vo blockchain sieťach sú používatelia siete motivovaní vytvárať nové bloky. Majú šancu získať odmenu v podobe časti hodnoty z vykonávanej transakcie. Táto skutočnosť vedie na časté súbežné publikovanie nových, navzájom odlišných blokov v jednej chvíli.

Na to, aby nevznikali konflikty, pri ktorých uzly ďalej propagujú rôzne vetvy, siete blockchain využívajú rôzne modely na vytvorenie konsenzu, čím zaručia spoluprácu aj vzájomne nedôverčivých uzlov.

Proof-of-work

Pred publikovaním nového bloku je nutné vyriešiť výpočtovo náročnú úlohu, ktorej riešenie slúži ako dôkaz o vykonanej práci (*proof-of-work* - PoW). Úloha je koncipovaná tak, aby jej riešenie bolo výpočtovo náročné, ale následná kontrola platnosti prebehla jednoducho. Každý novo pridaný blok je overený každým uzlom v sieti a pokiaľ nespĺňa kritériá platnosti, je zamietnutý.

Zvyčajne sa jedná o výpočet hash funkcie z obsahu hlavičky bloku, pričom výsledok tejto funkcie musí odpovedať určitej podmienke (napr. hodnota výsledku musí byť menšia ako dohodnutá hodnota). Blockchain založený na tejto metóde obsahuje v hlavičke bloku číselnú hodnotu nazývanú *nonce*, ktorú je možné ľubovoľne meniť. Publikujúce uzly (tiež označované ako ťažiari) tak hľadajú takú hodnotu čísla *nonce*, ktorá vedie na výpočet hashu spĺňajúceho podmienku výpočtu. Rýchlosť nájdenia riešenia (vyťaženia) závisí od počtu a výkonnosti publikujúcich uzlov v sieti. Preto je možné podmienku časom meniť, čo ovplyvňuje náročnosť výpočtu, a tým aj rýchlosť zverejňovania blokov.

Dôležitým aspektom tohto modelu je fakt, že práca investovaná do úlohy nemá vplyv na pravdepodobnosť nájdenia riešenia súčasnej alebo nasledujúcej úlohy. Čo znamená, ak niekto nájde riešenie úlohy a daný blok publikuje, každý môže svoje aktuálnu prácu zahodiť a začať stavať na novo prijatom bloku.

Preto najväčšiu šancu získať odmenu z vyťaženia majú ťažiari zoskupovaní do určitých skupín (anglicky *mining pools*). Ak sa podarí niekomu v skupine nájsť nový blok, odmena je spravodlivo rozdelená medzi všetkých členov. Avšak vytvárania takýchto skupín podkopáva základné princípy technológie blockchain, a pri získaní nadpolovičnej väčšiny môže takáto skupina začať potvrdzovať falošné transakcie [51].

Proof-of-stake

Proof-of-stake vznikol ako riešenie pre odstránenie záporov *proof-of-work* konceptu, akým je náročnosť na výpočtové zdroje, vrátane času, prevádzkových nákladov na spotrebovanú elektrinu a obstarávacie náklady systémov na ťažbu.

Koncept je založený na skupine validátorov. Pred vytvorením nasledujúceho bloku je na základe určitého algoritmu zvolený jeden z nich, ktorý následne vykoná validáciu následného bloku a získa odmenu. Používateľ je zaradený do skupiny validátorov len po vložení vkladu. Tento vklad je používateľovi zablokovaný na dobu vykonávania tejto činnosti. Algoritmus voľby uprednostňuje validátorov na základe ich hodnoty a staroby vkladu, pretože čím viac prostriedkov konkrétny používateľ vložil do systému, tým menšia pravdepodobnosť, že by ho chcel kompromitovať. V prípade, ak by validátor vyprodukoval nevalidný blok, prišiel by o časť vkladu [51].

Koncept sa spolieha na to, že získanie 51% hodnoty vkladu je príliš nákladné. Zároveň útočník, ktorý by vykonal takýto útok riskuje stratu hodnoty, čím by v podstate devaluoval vlastný majetok. Preto je takýto deštruktívny útok pre útočníka vysoko neprofitový, a tým nezaujímavý. Avšak takáto úvaha môže byť mylná [37].

Ak dostatočný počet ľudí nadobudne dôveru v systém tým, že vloží doňho prostriedky, systém sa tým stane lepšie zabezpečeným. Avšak z počiatku je nutné použiť iný koncept vytvorenia konsenzu a akonáhle je komunita pevne založená, je môžné prejsť na *proof-of-stake* koncept [16].

3.1.4 Transakcie

Zainteresované strany interagujú medzi sebou pomocou transakcií. Transakcie môžu niesť rôzne dáta v závislosti od uplatnenia technológie blockchain. Napr. pod transakciou v rámci kryptomien rozumieme prevod kryptomeny medzi používateľmi tejto siete.

Transakcie sú zapisované do dátovej časti bloku, pričom každý blok môže obsahovať 0 a viac transakcií. Z bezpečnostného hľadiska je pre niektoré implementácie blockchainu dôležité neustále zverejňovanie nových blokov, čím potencionálnym útočníkom bránia k vytvoreniu dlhšieho pozmeneného reťazca.

Údaje, ktoré transakcia obsahuje sa líšia v závislosti na implementácií, avšak mechanizmus transakcie zostáva z veľkej časti totožný. Vytvorená transakcia najčastejšie obsahuje informácie ako identifikátor odosielateľa (väčšinou sa jedná o adresu), verejný kľúč odosielateľa, digitálny podpis, transakčné vstupy a výstupy.

Vo väčšine kryptomien pozostáva každá transakcia z transakčných vstupov a výstupov. Vstupom rozumieme zoznam digitálnych aktív, ktoré budú modifikované a prevádzané na výstupy. Digitálne aktíva sú reprezentované odkazom na svoj zdroj. To môže byť predchádzajúca transakcia, alebo v prípade novej aktívy sa jedná o pôvodnú udalosť. Odosielateľ

musí byť schopný preukázať prístup k odkazovaným aktívam obvykle pomocou podpisu. Z tejto podstaty vyplýva, že žiadna transakcia nemôže do systému zaviesť nové aktíva, alebo zrušiť pôvodné. Transakcia tak môže len rozdeliť jedno, prípadne viac aktív do menších novo vzniknutých aktív, prípadne naopak zjednotiť viac aktív do celku. Rozdelenie alebo zjednotenie je určené podľa výstupu transakcie.

Výstupom transakcie sú adresy účtov spoločne s množstvom aktív, na ktoré budú dané prostriedky prevedené po zapísaní a zverejnení transakcie do blockchainu. Taktiež je nutné špecifikovať podmienky, za akých bude môcť nový vlastník využiť pridelené aktíva. V niektorých implementáciách je nutné vyčleniť istú časť aktív tomu, kto danú transakciu zapíše do bloku a následne zverejní do siete. Keďže počet transakcií je v bloku obmedzený, môže sa stať, že nie všetky transakcie sú zaradené do nasledujúceho bloku. Transakcie, ktoré vyčlenia vyššiu odmenu za zápis, sú pre publikujúce uzly zaujímavejšie, a tým sú zapísané prednostnejšie.

Pred samotným zápisom je nutné skontrolovať platnosť a autenticitu transakcie. Transakcia je považovaná za platnú, ak spĺňa všetky požiadavky protokolu. Autenticitu nadobúda transakcia vtedy, ak odosielateľ preukáže prístup ku všetkým vstupným aktívam. Toto je zvyčajne docielené za použitia asymetrickej kryptografie spojenej s kryptografickými hashovacími funkciami. Ešte pred prijatím aktív musí príjemca poskytnúť svoju adresu odosielateľovi. Adresa je väčšinou tvorená hashom verejnej zložky asymetrického páru kľúčov, ktoré si príjemca sám vytvorí. Odosielateľ tak v transakcii uvedie danú adresu ako adresu prijímateľa. Prijímateľ môže ďalej operovať s aktívami len za preukázania odpovedajúceho privátneho kľúča.

Smart kontrakty

Okrem implementovania jednoduchých transakcií na prenos aktív, môže určité siete blockchaniu podporovať prenos dát pomocou takzvaných *smart kontrakov*. Smart kontrakty sú počítačové programy, ktoré sú schopné plniť podmienky dohody medzi jednotlivými stranami bez potreby ľudskej koordinácie alebo zásahov [12]. Tieto dohody možno zaznamenávať a overovať v blockchaine. Po zverejnení smart kontraktu je na všetkých uzloch zahájené vykonávanie kódu, ktorý zvyčajne pozostáva zo sledu podmienených udalostí: „ak sa niečo stane, potom sa vykonajú určité transakcie“. Týmto spôsobom je možné vykonať dôveryhodnú transakciu medzi dvoma alebo viac stranami bez potreby sprostredkovateľov (napr. sprístupnenie hotelovej izby na určitý čas po zaplatení poplatku) [15].

Smart kontrakty sa skladajú z časti kódu – stavové premenné, a metódy. Po zverejnení kontraktu na blockchaine je ďalej identifikovateľný pomocou pridelenej adresy. Oprávnený užívatelia môžu následne volať jeho metódy, a tým upravovať jeho stavové premenné. Existuje viacero programovacích jazykov, ktoré sú vytvorené na vytváranie smart kontraktov (napr. *Solidity*, *Vyper*). Podporujúca blockchain technológia musí implementovať virtuálny stroj, v ktorom sú vykonávané jednotlivé inštrukcie kódu. Pretože kód je nezávisle vykonávaný na rôznych uzloch a jeho výsledok je porovnávaný s výsledkom vyťaženého bloku, kód musí byť deterministický a vždy musí produkovať rovnaký výstup pre konkrétne zadaný vstup. Vstupné dáta pochádzajúce vrámci systému pracujú s kontraktom podľa očakávaní, avšak dáta pochádzajúce zo zdrojov reálneho sveta (označované ako *Orákulum*), môžu spôsobiť problémy [51].

3.2 Kryptomeny

Kryptomena je digitálna, resp. virtuálna mena, ktorá je chránená modernou kryptografiou. Vďaka tomu sú prakticky nemožné situácie, pri ktorých by mohol potencionálny útočník vytvoriť falzifikát alebo vykonať viacnásobnú útratu, čím by zmenil počet platných jednotiek meny v obehu. Väčšina kryptomien je založených na decentralizovanom prístupe s využitím technológie blockchain, čo prináša teoretickú imúnnosť voči vládnym zásahom alebo manipulácií. Na druhú stranu, informácie o všetkých vykonaných transakciách sú verejne dostupné, avšak adresy, s ktorými používatelia disponujú, neprezrádzajú ich skutočnú identitu [24].

Technológie kryptomien sú v neustálom vývoji. Avšak zavedenie nových funkcií, alebo zmena aktuálnych vyžaduje presvedčiť a následne upraviť kódy väčšiny uzlov v sieti, čo predstavuje veľký problém. Pri zmene protokolu blockchain sieti sa vytvárajú tzv. vetvy (anglicky *forks*). Existujú dve kategórie vetvenia – spätne kompatibilné (*soft forks*) a spätne nekompatibilné (*hard forks*). Spätne kompatibilné vetvenie je taká zmena blockchainu, ktorá je spätne kompatibilná s doposiaľ neupravenými uzlami siete. Na druhú stranu bloky, ktoré nasledujú nekompatibilné zmeny, sú odmietané neupravenými uzlami. Pre zavedenie takýchto zmien sa určí konkrétny bod v čase (zvyčajne sa jedná o špecifické číslo bloku), pri ktorých všetky uzly zmenia svoj protokol. Neupravené uzly, ktoré zostanú publikovať bloky starého formátu, simultánne vytvárajú ďalšiu vetvu blockchainu. Okrem úmyselného vytvárania ďalších vetiev môžu vznikať nové vetvy aj neúmyselne, napr. pri softvérovej chybe. Pri vetvení dochádza k vzniku dvoch rôznych nezávislých a navzájom nekompatibilných sietí, čo má za následok zdvojnásobenie počet aktív. Predpokladá sa, že ďalej bude pokračovať len jedna vetva, čím stará vetva stratí na hodnote a časom zanikne [51].

Koncom roka 2020 existujú tisícky rôznych kryptomien.[1] Najväčšiu tržnú kapitalizáciu si s prehľadom udržiava prvo-vzniknutá kryptomena – Bitcoin. Druhou najpopulárnejšou kryptomenou je Ethereum. Do top 5 sa ďalej dostávajú kryptomeny XRP, Litecoin a Bitcoin Cash. Dve najrozšírenejšie kryptomeny sú popísané v nasledujúcich podkapitolách.

3.2.1 Bitcoin

Často je v neodbornej literatúre spájaný pojem blockchain s kryptomenou Bitcoin, keďže práve táto kryptomena rozšírila povedomie tejto technológie. Ako už bolo v úvode kapitoly spomínané, jedná sa o prvú a zároveň najrozšírenejšiu kryptomenu s najväčším tržným podielom. Jeden Bitcoin, označovaný symbolom 1 BTC, je možné rozdeliť až na 10^8 častí nazývaných *Satoshi* (SAT).

Tvorba bloku

Jednotlivé bloky Bitcoin blockchainu sú veľkostne obmedzené na 1 MB. Avšak pri použití *SegWit* transakcií je možné mierne zvýšiť efektívnu veľkosť bloku, a tým doňho uložiť viac transakcií.

Pre získanie hashu dátovej časti je využitá dátová štruktúra Merklového stromu. Výhodou tejto štruktúry je overenie integrity v logaritmickom čase. Jednotlivé transakcie tvoria listové uzly stromu, pričom ostatné uzly reprezentujú výsledok hash funkcie aplikovanej na dáta potomkov daného uzla. Výsledný hash, koreň stromu, je uložený v hlavičke bloku.

[1] https://coinmarketcap.com/

Bitcoin využíva PoW model na vytvorenie konsenzu, čo znamená, že nové bloky sú v sieti vytvárané ťažiarmi. Ťažiari, po zahrnutí čakajúcich transakcií do dátovej časti bloku, upravujú hodnotu *nonce* v hlavičke tak, aby výsledný hash bloku začínal určitým počtom núl. Hash je počítaný pomocou algoritmu `SHA-256`. Počet potrebných núl je upravovaný každých 2016 blokov tak, aby priemerná doba výpočtu bloku bola 10 minút.

S každým novo vytvoreným blokom vznikajú aj nové „*mince*“. Ich počet je znižovaný na polovicu po každých 210 000 vytvorených blokov. Táto udalosť sa nazýva polenie (*halving*). Z počiatku bolo možné získať vytvorením každého nového bloku až 50 BTC. Ostatné (štvrté) polenie nastalo 11. Mája 2020, a aktuálne je táto hodnota 6,25 BTC. Na základe toho je definovaný maximálny počet BTC, ktorý môže byť v obehu – 21 miliónov. Novo vzniknuté mince sú pripísané ako odmena ťažiarovi, ktorý publikuje nový blok. Okrem toho sú ťažiari motivovaní ťažením aj pripisovaním si poplatkov z transakcií zahrnutých v bloku [51, 1, 25].

Transakcie

Každá Bitcoin transakcia je popísaná v základnom skriptovacom jazyku *Bitcoin Script* zásobníkového typu. Skripty vyžadujú len minimálne spracovanie, a preto sú zámerne navrhnuté ako Turingovo neúplné, keďže im chýba podpora cyklov. Dôvodom ich nezahrnutia je následné uľahčenie a zabezpečenie procesu overenia transakcie.

Existuje niekoľko typov šablón transakcií, ktoré sú označené ako štandardné. Predvolená, a zároveň najpoužívanejšia, je platba na hash verejnej adresy prijímateľa. Existuje aj varianta bez použitia hash funkcie. Ďalej je možné tvoriť transakcie s viacerými vstupmi a výstupmi. Bitcoin podporuje aj vytváranie vlastných skriptov, ktoré môžu byť použité napr. pri multi-podpise viacerých strán. Tvorca skriptu rozpošle jeho hash používateľom, ktorý následne vykonávajú platbu na daný hash skriptu. Je v záujme tvorcu, aby bol vytvorený skript bezchybný a prijímateľ tak dostal prostriedky. Pre šetrenie miesta, a tým zvýšenia priepustnosti transakcií, boli zavedené *SegWit* transakcie, ktoré skracujú bežné platby na hash verejného kľúča alebo platby na hash skriptu. Okrem platieb je možné do Bitcoin blockchainu uložiť aj dáta. Transakcia je prijatá sieťou, ak spĺňa test overujúci príslušnosť do jednej z typov šablón [11].

Nedostatky

Aj napriek najväčšej obľúbenosti z pomedzi všetkých kryptomien, Bitcoin v súčastnosti nie je preferovaná mena na bežné platobné transakcie. Hlavným nedostatkom je škálovateľnosť. Sieť používa čoraz väčší počet ľudí, avšak veľkosť bloku zostáva rovnaká. Tým vzniká obmedzenie na počet transakcií v bloku a používatelia, ktorí chcú zapísať transakciu v najkratšej dobe, sú nútení zvyšovať odmeny ťažiarom, čím sa priemerná výška poplatkov neustále zvyšuje. Ďalším problémom je vysoká volatilnosť hodnoty Bitcoinu, čoho dôsledkami sú denné výkyvy hodnôt v rozmedzí až desiatok percent.

Vývojári sú motivovaní odstrániť spomínané nedostatky, čím vznikajú nové alternatívne kryptomeny označované aj ako *Altcoin*.

3.2.2 Ethereum

Najznámejšia alternatívna kryptomena Ethereum, označovaná symbolom ETH, posúva koncept kryptomien na ďalšiu úroveň. Prináša myšlienku Turingovo-komplentého blockchainu. Ethereum pracuje s dátovou štruktúrou podobne ako Bitcoin s rozdielom, že Ethereum má

vstavaný programovací jazyk – Solidity, Viper. Kód v týchto jazykoch tvorí základ *smart kontraktu*. Kódy sú vykonávané vo virtuálnych strojoch Etherea (EVM) v každom uzle. Výpočet stojí určité prostriedky (čas a priestor), a preto je za zverejnenie a vykonávanie kódu kontraktov nutné zaplatiť určitú sumu – poplatok (nazývaný ako *gas*). Tento poplatok určuje, akú veľkú časť výpočtu je možné vykonať a koľko stavových premenných je možné alokovať. Cenu poplatku je možné zistiť podľa aktuálnych cien inštrukcií použitých EVM v danej transakcii. Ak by náhodou tvorca transakcie poskytol menej *gasu*, než je potrebné, výpočet by bol ukončený akonáhle by došiel vložený *gas* a všetky zmeny vykonané transakciou by boli odvolané. Tým je zabezpečené, že nedôjde k situácií, kedy by kód spôsobil nekonečné vykonávanie. Každý blok má určený limit *gasu*, ktorý môže spotrebovať. Transakcie vkladané do bloku nesmú v súčte prekročiť tento limit.

Spôsobov, ako je možné využiť smart kontrakty v sieti Ethereum je nespočetne veľa. Neustále vznikajú nové aplikácie a služby. Jedným zo spôsobov využitia je vytvorenie digitálnych tokenov, ktoré môžu reprezentovať rôzne digitálne aktíva (napr. poukážky), ale aj objekty reálneho sveta. Jeden z najrozšírenejších tokenov je ERC-20, pomocou ktorého vznikli aj kryptomeny ako *Wrapped Bitcoin*.[2]

Význam Ethereum meny tak nie je len v uchovávaní hodnoty (ako to je u Bitcoinu), ale je možné ju považovať ako prostriedok na vykonávanie decentralizovaných aplikácií a služieb [20].

Ethereum 2.0

Rovnako ako Bitcoin, je konsenzus postavený na modeli PoW, avšak za použitia iného hashovacieho algoritmu – `Keccak-256`. Ten je navrhnutý tak, aby svojou pamäťovou náročnosťou bol ťažko implementovaný špeciálnymi ASIC obvodmi [20]. PoW má svoje nedostatky ako vysoké nároky na elektrickú energiu, postupná centralizácia ťažiarov. Z toho dôvodu je momentálny vývoj venovaný prechodu na iný konsenzuálny model – PoS, čím sa urýchli zverejňovanie nových transakcií a zníži cena poplatkov vynaložených k zrealizovaniu transakcie [47].

[2] https://wbtc.network/

Kapitola 4

Blockchain s využitím Trusted Computing

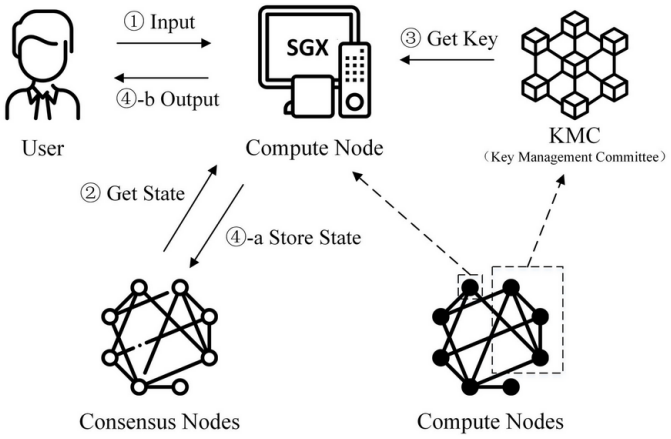
Ako bolo uvedené v kapitole 3, technológia blockchain má svoje obmedzenia. Jednou z nich je slabá škálovateľnosť, čo spôsobuje nedostatočnú priepustnosť transakcií (vrátane smart kontraktov). Zväčšenie veľkosti blokov nie je ideálnym riešením, pretože to by viedlo k rýchlejšie narastajúcim potrebám úložiskovej kapacity a tým pomalšej propagácií sieťou. Ďalšou nevýhodou je verejnosť transakcií, čím je možné sledovať zostatky každej adresy v sieti. Nájdenie prepojenia medzi adresou a identitou používateľa je tak otázkou analýzy.

Systémy založené na technológií TC ponúkajú vykonávanie kódu a prácu s citlivými dátami v bezpečnom kontajneri s využitím dôveryhodného hardvéru. Kontajner chráni integritu a dôvernosť dát, zatiaľ čo vykonáva výpočet. Ten sú používatelia schopní vzdialene overiť, a tak osvedčiť prácu TC ako spoľahlivú.

Na základe týchto charakteristík, spoločne s ďalšími popísanými v kapitole 2, je možné poskytnúť silnú podporu v riešení spomínaných problémov blockchainu, čomu sa poslednú dobu zameriavali rôzne výskumy. V tejto kapitole sú stručne opísané niektoré z publikovaných riešení. V tabuľke 4.1 sú následné zhrnuté popísané riešenia [9].

4.1 Ekiden

Projekt Ekiden [14] sa zameriava na riešenie dôvernosti a efektívnosti vykonávania smart kontraktov v blockchain sieťach. Návrh je založený na rozdelení uzlov siete do dvoch skupín. Prvá, zložená z uzlov podieľajúcich sa na tvorení konsenzu, udržiavaní účtovej knihy a aktualizovaní stavu smart kontraktov. Druhá skupina uzlov sú výpočtové uzly, ktoré musia podporovať jednu z TEE prostredí (napr. Intel SGX). Tieto uzly zodpovedajú za vykonávanie smart kontraktov v enkláve. Akýkoľvek uzol, ktorý podporuje TEE môže byť zahrnutý do skupiny výpočtových uzlov, čo zabezpečuje škálovateľnosť. Kvórum výpočtových uzlov tvorí komisiu pre správu kľúčov (KMC). Funkciou komisie je generovanie a správa kľúčov výpočtových uzlov, pomocou distribuovaných protokolov. Pre každý kontrakt je vygenerovaný súkromný a verejný kľúč. Užívateľ ho pred odoslaním do výpočtových uzlov zašifruje verejným kľúčom. Odpoveď sa vracia užívateľovi zabezpečeným kanálom a zároveň je konsenznými uzlami aktualizovaný stav kontraktu na blockchaine. Konsenzné uzly overujú validitu aktualizácií stavov pomocou vzdialenej atestácie voči výpočtovým uzlom. Prehľad spomínanej architektúry je znázornený na obrázku 4.1 [14, 9].



Obr. 4.1: Ekiden architektúra (prevzaté z [9]).

4.2 FastKitten

FastKitten [21] umožňuje vykonávať ľubovoľne komplexné smart kontrakty pri nízkych nákladoch nad decentralizovanými kryptomenami, ktoré podporujú iba jednoduché transakcie. Riešenie je založené na dvoch podmienkach. Systém blockchain musí podporovať časované transakcie a uchovávanie dát, a zároveň je nutné zaviesť operátora s prístupom do TEE prostredia, ktorý vykonáva kontrakty.

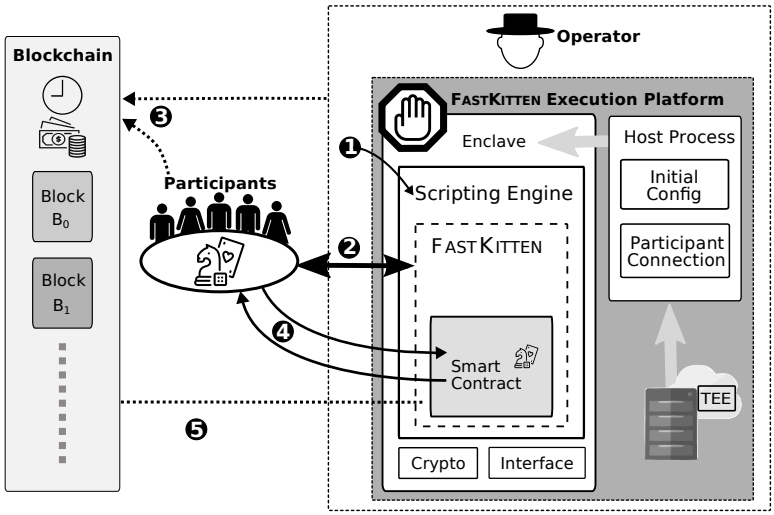
Vykonávanie kontraktu je zložené z troch fáz. V prvej je operátorovi odoslaný kontrakt s informáciami o účastníkoch pre inicializáciu enklávy. Operátor inicializuje enklávu a vloží vklad určitej veľkosti na časovanú transakciu. Následne si účastníci overia správnosti kontraktu v enkláve pomocou atestácie a vložia vklad zapísaním do blockchainu. Veľkosť vkladu operátora sa musí rovnať súčtu vkladov všetkých účastníkov, aby nemohol zneužiť svoje postavenie a získať tak vklady všetkých účastníkov. V takomto prípade by po uplynutí časovača prišiel o svoj vklad v prospech účastníkov.

V druhej fáze prebieha komunikácia medzi účastníkmi a enklávou zabezpečeným kanálom, preto nie je možné, aby ju operátor odpočúval. Operátor preposiela vstupy užívateľov do enklávy a výstupy enklávy užívateľom, pričom žiadna informácia z tejto komunikácie nie je zapisovaná do blockchainu.

V poslednej fáze je vytvorená enklávou finálna transakcia, ktorá udáva rozdelenie prostriedkov užívateľov a zároveň transakciu, ktorou si bude môcť operátor vyzdvihnúť svoj vklad. Preto je v záujme operátora, aby transakciu zverejnil do blockchain siete.

V prípade podozrenia škodlivého chovania jedného z užívateľov, operátor zverejní do blockchainu výzvu – transakciu pripisujúcu malú čiastku podozrivému účastníkovi. Ak sa účastník nepreukáže odoslaním rovnakej čiastky naspäť operátorovi, je dokázané, že daný účastník nespolupracuje a bude potrestaný. Ak sa však účastník preukáže, výpočet pokračuje ďalej. Rovnaký mechanizmus je možné použiť aj obrátene – preukázať škodlivosť operátora.

Avšak tento mechanizmus nie je dokonalý a je nutné podotknúť niekoľko obmedzení. Požitie časovaču obmedzuje dobu vykonávania druhej fázy. Predstavený mechanizmus nedokáže tolerovať zlyhania enklávy. Architektúra riešenia je znázornená na obrázku 4.2, pričom prerušované šípky označujú interakciu s blockchainom a neprerušované šípky znázorňujú komunikáciu medzi stranami [21, 9].



Obr. 4.2: FastKitten architektúra (prevzaté z [21]).

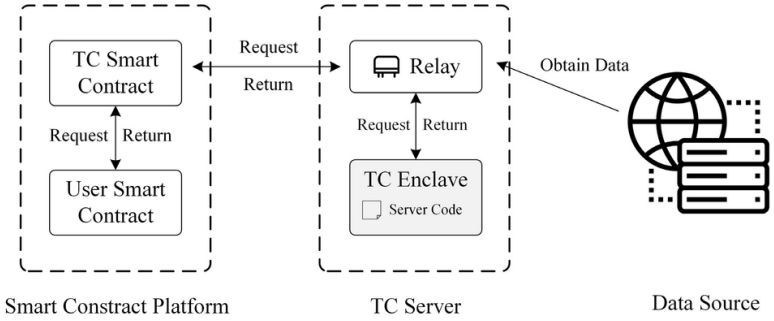
4.3 Town Crier

Smart kontrakty získavajú dáta z reálneho sveta väčšinou z dôveryhodných webových zdrojov pomocou informačných kanálov kontraktov (orákul). Avšak žiadosti k týmto zdrojom sú verejné a takto získaným údajom nie je možné plne dôverovať, keďže daný zdroj môže byť nespoľahlivý alebo môže byť obeťou *man-in-the-middle* útoku.

Town Crier (TC) [54] predstavuje systém poskytovania autentizovaných dát do siete blockchain. Jedná sa tak o *orákulum* postavené nad TEE prostredím, ktoré vytvára spojenie medzi HTTPS webstránkou a Ethereum blockchainom. Systém sa skladá z troch častí. Prvou je smart kontrakt, ktorý poskytuje rozhranie pre ostatné kontrakty – vytvára žiadosti a získava odpovedajúce dáta. Ďalšiu časť tvorí TC server obsahujúci enklávu. Enkláva zodpovedá za analýzu požiadavky, ktorú následne odosiela na cieľovú adresu. Keďže enkláva nemôže priamo komunikovať na sieťovom rozhraní, o komunikáciu sa stará relé (anglicky *relay*) skript. Architektúra systému je znázornená na obrázku 4.3.

V prípade, že užívateľ vyžaduje dáta mimo blockchain, je zavolaný TC smart kontrakt, ktorý upraví svoj stav. Relé tento stav sleduje, a v prípade jeho zmeny odošle požiadavku do TC enklávy. Enkláva za pomoci relé kontaktuje zdroj s využitím zabezpečeného protokolu HTTPS a získa potrebné dáta. Výsledné dáta sú vrátené TC smart kontraktu pomocou správy zapísanej do blockchainu. V poslednom kroku sú finálne dáta poskytnuté užívateľskému kontraktu.

Town Crier tak poskytuje overené dáta smart kontraktom a to aj bez dôveryhodného operátora. Ochranou pred kompromitovaním jednej inštancie, systém navrhuje použitie viacerých TC serverov, ktoré zároveň poskytnú užívateľovi väčšiu spoľahlivosť dát. TC systém umožňuje používateľovi definovať viacero zdrojov dát, aby sa dosiahla odolnosť voči chybám niektorého zo zdrojov. Avšak v tomto prípade TC nemôže zaručiť konzistentnosť dát, ku ktorým pristupujú rôzne uzly [54, 9].



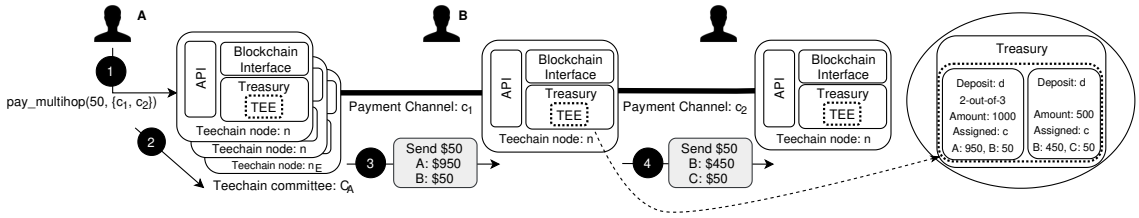
Obr. 4.3: Town Crier architektúra (prevzaté z [9]).

4.4 Teechain

Teechain [39] popisuje novú platobnú sieť, ktorá podporuje bezpečné platby s vysokou priepustnosťou na existujúcich blockchain štruktúrach. Sieť Teechain pozostáva z tzv. pokladníc (anglicky *treasuries*), ktoré sú chránené TEE prostredím. Toto prostredie zabezpečuje udržiavanie vkladov vo fondoch a vykonávanie nového efektívneho platobného protokolu mimo blockchain. Pokladnice sú prevádzkované rôznymi stranami a medzi sebou komunikujú pomocou P2P siete.

Komunikácia dvoch strán vyžaduje zavedenie platobného kanálu. Ešte pred jeho vytvorením, pokladnica vygeneruje svoju adresu pomocou páru kľúčov, pričom súkromný kľúč je bezpečne uložený v enkláve. Následne používateľ zakladajúci kanál zašle finančné prostriedky na danú adresu vo forme zálohy a transakciu publikuje do blockchainu, čím vytvorí kanál. Na základe správ zasielajúcich medzi účastníkmi mimo blockchainu sú aktualizované vklady. Každá strana môže kedykoľvek uzavrieť platobný kanál zaslaním transakcie do blockchainu. Teechain podporuje transakcie medzi dvoma stranami, ktoré medzi sebou nemajú zavedený platobný kanál. V takom prípade sa využíva viac-skoková transakcia, ktorá je znázornená spoločne s dizajnovou architektúrou teechainu na obrázku 4.4.

Z dôvodu možného zlyhania alebo kompromitovania uzlov vyžaduje každá transakcia v sieti Teechain aspoň n podpisov z m pokladníc (pričom $m > n$, kde m značí komisiu). Zúčtovacia transakcia tak nezávisí od jedného uzla. Pokladnice spravujú informácie o vkladoch pomocou replikačného protokolu. Uzly z komisie nemusia používať rovnakú TEE technológiu, čím odpadávajú problémy pri kompromitácií konkrétnej technológie [39, 9].



Obr. 4.4: Prehľad architektúry Teechain (prevzaté z [39]).

4.5 Tesseract

V súčasnosti mnoho centralizovaných zmenární ponúka používateľom výmenu aktív medzi viacerými kryptomenami. Pri realizácií takejto výmeny musí používateľ plne dôverovať danej zmenárni, pretože pre dokončenie výmeny musí zmenáreň najprv obdržať používateľove krypto tokeny. Decentralizované zmenárne založené na smart kontraktoch riešia riziká spojené s centralizovanosťou, avšak nemôžu poskytovať výmenu v reálnom čase.

Ako riešenie problému vznikol projekt s názvom Tesseract [10] – bezpečná kryptozmenáreň pracujúca v reálnom čase. Pre vykonanie výmeny kryptomeny je nutné najprv vykonať registráciu, a následne odoslať prostriedky na Tesseract adresu. Táto adresa je odlišná pre každú kryptomenu. Adresa ako aj stav účtov registrovaných používateľov spravuje enkláva zmenárne.

Používateľ vytvorí a odošle ponuky zámeny do enklávy zmenárne zabezpečeným kanálom. Po prijatí ponuky enklávou je anonymne rozposlaná každému užívateľovi. Ak užívateľ ponuku prijme, enkláva upraví aktuálny stav účtu obchodujúcich užívateľov a zapíše transakciu vyrovnania do blockchain siete. Avšak, keďže sa jedná o rozdielne kryptomeny, vzniknú takto až dve transakcie, ktoré musia mať atomické vlastnosti. Pre zabezpečenie týchto vlastností Tesseract implementuje protokol spravodlivého vyrovnania [10, 9].

4.6 ShadowEth

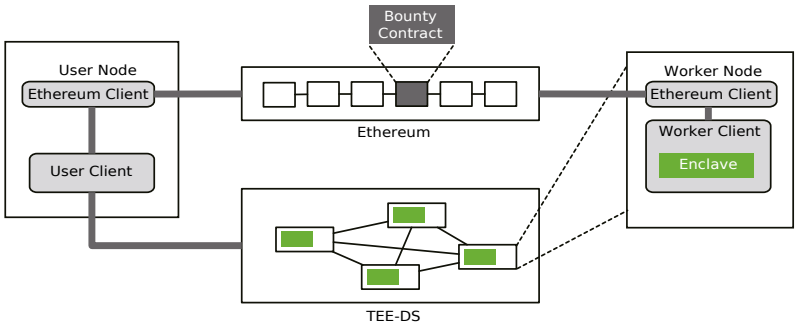
Jednou zo spomínaných nevýhod verejného blockchainu postaveného na smart kontraktoch je práve súkromie, keďže všetky informácie o vykonaných transakciách sú verejne dostupné. Platforma ShadowEth rieši tento problém pomocou hardvérovej enklávy, ktorá zabezpečuje dôvernosť smart kontraktov pri zachovaní integrity a dostupnosti. Riešenie vytvára dôveryhodné výpočtové prostredie mimo verejný blockchain a umožňuje tak ukladanie a vykonávanie súkromných smart kontraktov. Verejný blockchain je používaný len pre proces nasadenia, verifikácie a ukladania meta-informácií o privátnych kontraktoch pomocou tzv. *bounty kontraktu*.

Zároveň ShadowEth implementuje distribuované úložisko (TEE-DS) pre uloženie kódu a stavu súkromných kontraktov. Toto úložisko je implementované mimo verejný blockchain a je chránené pomocou enkláv pracovníkov. Z podstaty TEE je tak nemožné, aby tieto dáta unikli alebo boli pozmenené.

Vykonanie súkromného kontraktu začína klient zaslaním kontraktu do distribuovaného úložiska pomocou zabezpečeného kanálu a súčasným zverejnením identifikačných informácií kontraktu do bounty kontraktu na verejný blockchain. Následne užívateľ odošle žiadosť o vyvolanie kontraktu s príslušnými argumentami. Pracovný uzol zistí danú požiadavku a načíta tak kontrakt z TEE-DS do enklávy a vykoná ho na základe aktuálneho stavu. Okrem podpisu pracovníka po vykonaní súkromného kontraktu sa do verejného blockchainu ukladá aj hash spustiteľného kódu, verejného kľúča a hash stavu spoločne so zašifrovanými vstupno-výstupnými dátami. Komunikácia je znázornená na obrázku 4.5 [53, 9].

4.7 Aquareum

V reálnom svete stále prevládajú centralizované riešenia účtových kníh oproti decentralizovaným. Avšak pre ich centralizovanosť trpia istými obmedzeniami ako nedostatočná efektívnosť overenia, vyššie riziko cenzúry a nejasností.



Obr. 4.5: ShadowEth architektúra (prevzaté z [53]).

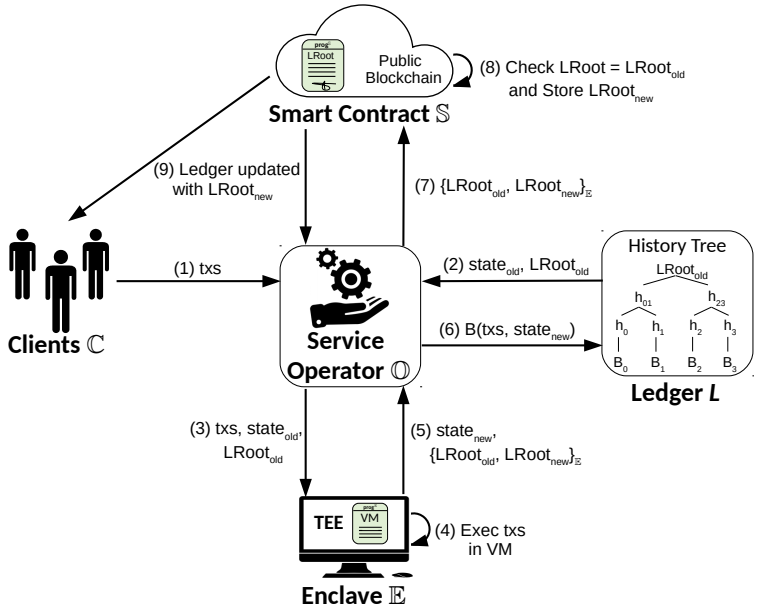
Tieto problémy odstraňuje framework Aquareum [32], ktorý prináša výhody oboch prístupov. Aquareum využíva TEE prostredie a verejnú smart kontraktovú platformu pre dosiahnutie overiteľnosti, nezvrátiteľnosti a potlačenie cenzúry.

Centralizovaným prvkom systému je operátor, ktorý vykonáva operácie nad účtovou knihou $\mathbb{L}$, na základe prijatých požiadaviek od klientov. Všetky zmeny $\mathbb{L}$ vedú na vytvorenie nového bloku, ktorý obsahuje hlavičku, zoznam vykonaných transakcií a ich potvrdení a je uložený v dátovej štruktúre *History Tree* [19] odolnej voči modifikácií. Tieto zmeny sú výlučne vykonávané v enkláve, ktorá dohliada nad jej konzistenciou a správnosťou. Pridanie bloku spôsobí zmenu hodnoty koreňa stromu (*LRoot*), ktorý agreguje všetky bloky pomocou hash funkcie, čím vytvorí nový stav $\mathbb{L}$. Prechod do nového stavu je zverejnený do existujúceho verejného blockchainu, a slúži ako dôkaz zaručujúci, že aktuálny stav $\mathbb{L}$ je overený a neexistujú žiadne alternatívne stavy. Postupnosť krokov je znázornená na obrázku 4.6.

Aby si bol klient istý, že jeho transakcia bola vykonaná, a teda zahrnutá v $\mathbb{L}$, platforma Aquareum poskytuje 3 typy dôkazov na vyžiadanie.

- Inkrementálny dôkaz (π^{inc}) medzi dvoma stavmi $\mathbb{L}_i$ a $\mathbb{L}_j$ dokazujúci, že $\mathbb{L}_j$ je rozšírením $\mathbb{L}_i$ ak $i \leq j$. Vďaka dátovému typu *Historic Tree* je možné tento dôkaz vyprodukovať v logaritmickej zložitosti.
- Dôkaz o členstve (*membership proof* – π^{mem}_{hdr}) bloku i v stave $\mathbb{L}_j$, pričom platí $i \leq j$. *Historic Tree* je špeciálny typ Merklového stromu, pri ktorom taktiež platí logaritmická zložitosť výpočtu ako pri Merklovom strome.
- Merklov dôkaz ($\pi^{mk}_{rcp_i}$), ktorý svedčí, že potvrdenie rcp_i transakcie tx_i je zahrnuté v bloku b.

Ak má klient podozrenie o cenzúrovaní jeho transakcií, požiada o jej vyšetrenie prostredníctvom verejnej smart kontraktovej platformy. Operátor účtovej knihy je povinný vybaviť všetky takto zadané požiadavky a preukázať sa verejným dôkazom o ich vyriešení. V prípade, že by operátor nevyriešil klientov dotaz, je verejne preukázateľné, že sa dopustil cenzúry, na základe čoho môže klient vykonať následné právne úkony (napr. obžalovať operátora) [32].



Obr. 4.6: Spracovanie klientských transakcií do Aquareum účtovej knihy (prevzaté z [32]).

Názov	**Cieľový problém**	**TEE**	**Riešenie**
Ekiden	Otázka dôvernosti a efektívnosti vykonávania smart kontraktov.	Výpočtové uzly	Presun vykonávania smart kontraktov do výpočtových uzlov s podporou TEE, ktoré zaručujú dôvernosť a minimalizujú zápis dát na blockchain, čím sa zvýši efektivita.
FastKitten	Chýbajúca podpora smart kontraktov v sieti Bitcoin.	Operátor	Vykonávanie smart kontraktov operátorom mimo Bitcoin blockchain.
Town Crier	Otázka dôveryhodnosti a bezpečnosti zdrojov dát smart kontraktov.	Orákulum	Vytváranie bezpečného spojenia medzi dôveryhodnými internetovými zdrojmi a enklávou Orákul.
Teechain	Existujúce schémy platobných kanálov nie sú bezpečné.	Komisia pokladníc	Vytvorenie zabezpečeného platobného kanálu medzi enklávami pokladníc, ktoré implementujú platobný protokol mimo blockchain.
Tesseract	Transakcie naprieč rôznymi blockchainami sú vykonávané nezabezpečenou autoritou alebo sú neefektívne.	Zmenáreň	Vytvorenie centrálnej zmenárne založenej na TEE, ktorá presúva obchodovanie do enklávy s následnými periodickými synchronizáciami s blockchainom.
ShadowEth	Otázka ochrany súkromia smart kontraktov v existujúcich verejných blockchainoch.	Pracovné uzly	Vykonávanie smart kontraktov v uzloch mimo blockchain s využitím TEE na ochranu súkromia.
Aquareum	Centralizované účtové knihy trpia nedostatočnou overiteľnosťou a rizikom cenzúry.	Operátor	Zavedenie operátora, ktorý môže operovať nad účtovou knihou len pomocou preddefinovaných operácií v enkláve, pričom dôkaz o každej zmene je zverejnený na verejný blockchain.

Tabuľka 4.1: Zhrnutie popisovaných riešení (vychádza z prieskumu [9]).

Kapitola 5

Návrh

Návrh semicentralizovanej kryptomeny podporujúcej externú interoperabilitu je možné rozdeliť do dvoch častí. Prvá časť návrhu je zameraná na realizáciu semicentralizovanej kryptomeny (bližšie v sekcii 5.1). Druhá časť (sekcia 5.2) rieši externú interoperabilitu medzi viacerými inštanciami navrhnutej kryptomeny.

5.1 Semicentralizovaná kryptomena

Decentralizované kryptomeny majú svoje krypto tokeny chránené pomocou kryptografických dôkazov uložených v blockchain sieťach. Centralizované kryptomeny zabezpečujú ochranu kryptografických dôkazov pomocou TEE. Určité vlastnosti, ako napr. minimalizácia rizika cenzúry, môžu byť zaručené pomocou verejného blockchainu. Týmto spôsobom báza dôveryhodného výpočtu (TCB) centralizovaných kryptomien môže zahŕňať obe technológie – TEE a blockchain. Na rovnakom princípe je založený aj framework Aquareum [32].

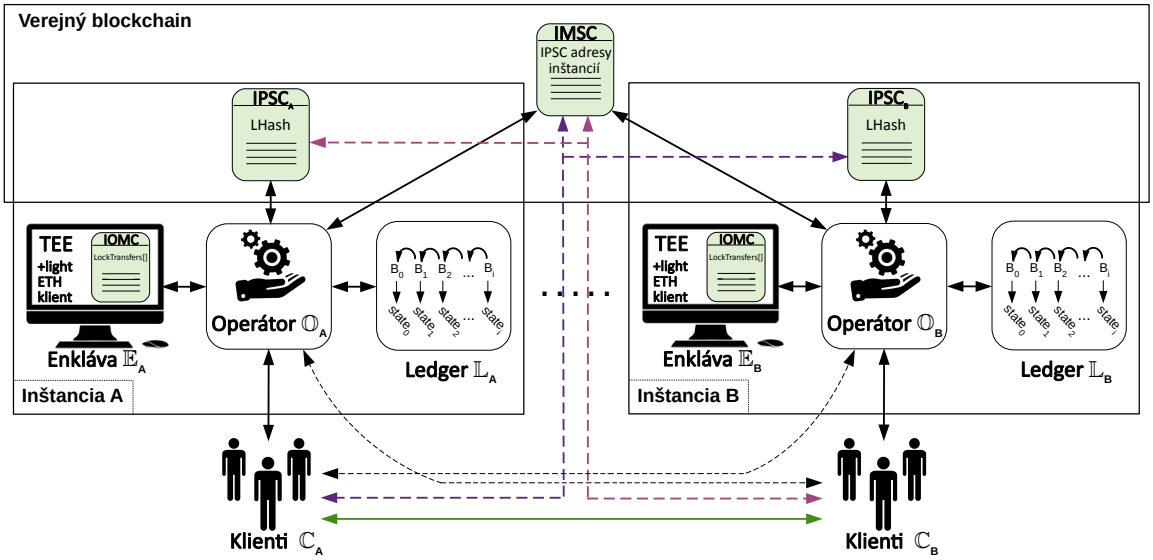
Navrhovaná semicentralizovaná kryptomena sa líši oproti centralizovanej kryptomene v tom, že na rozdiel od jednej centrálnej entity existuje viacero centrálnych entít komunikujúcich navzájom, pričom každá z nich obsluhuje vlastnú účtovú knihu. Na to, aby sa z centralizovanej kryptomeny stala semicentralizovaná, je nevyhnutné, aby podporovala komunikáciu s ostatnými entitami. Táto komunikácia naprieč entitami je riešená pomocou protokolu externej interoperability, ktorú tak musí každá entita podporovať (viac o protokole v sekcii 5.2).

Navrhovaná semicentralizovaná kryptomena vychádza z práce Aquareum [32] (popísaná v sekcii 4.7), ktorá je doplnená o interkomunikačný protokol uvedený v nasledujúcej sekcii.

5.2 Interoperabilita

Externú interoperabilitu nadobúda kryptomena vtedy, ak jej inštancia môže komunikovať s ďalšími inštanciami kryptomien a vykonávať tak určité úkony definované pravidlami protokolu, ktorý musia obe zároveň podporovať. Tento protokol je navrhovaný s cieľom eliminácie následkov jednej zo strán pri podvodnom chovaní druhej strany. Protokol by tak mal predchádzať duplikácii, alebo prípadnej strate tokenov pri transakciách medzi inštanciami. Bezpečnosti navrhovaného interoperabilitového protokolu je venovaná kapitola 7.

V rámci navrhovaného protokolu budeme uvažovať o dvoch možných scenároch. V prvom sú prenášané natívne krypto tokeny z jednej inštancie do druhej. Druhý scenár zahrňuje



Obr. 5.1: Návrh architektúry.

externé invokovanie funkcií mikro kontraktov[1] inštancie. Tieto funkcie môžu modifikovať stav účtovej knihy volanej strany alebo zároveň upravujú aj stav účtovej knihy volajúceho strany. Práca je primárne zameraná na prvý scénar zo spomínaných možností – prenášanie z natívnych krypto tokenov.

Všetky spomínané spôsoby komunikácie vychádzajú z komunikácie enkláv operátorov konkrétnych inštancií. Avšak enkláva neumožňuje priamu komunikáciu s okolitým svetom, a preto je nevyhnutné využiť prostredníka. Jedným z riešení je možnosť využiť operátora enklávy, avšak ideálnejším riešením je využitie klientov ako prostredníkov. Klienti majú dostatočné právomoci na vykonanie všetkých potrebných úkonov, pričom je v ich záujme úspešne dokončenie interoperabilitového protokolu, čím zároveň aj zjednodušia prácu operátora.

Konceptuálny model architektonického návrhu je znázornený na obrázku 5.1 a pozostáva z viacerých inštancií smart kontraktovej platformy Aquareum a verejného blockchainu. Základnou entitou každej inštancie Aquarea je operátor – $\mathbb{O}$. Ten zodpovedá za vytvorenie a udržiavanie účtovej knihy (anglicky *ledger* – $\mathbb{L}$), inicializuje prostredie enklávy – $\mathbb{E}$ a zároveň komunikuje s klientmi – $\mathbb{C}$, ktorí mu posielajú transakcie a ďalšie požiadavky, ako napr. žiadosť o registráciu klienta, potvrdenia o vykonaných transakciách, atď.. Prijaté transakcie operátor validuje a odošle spolu s aktuálnou verziou účtovej knihy do enklávy. Enkláva na základe transakcie upraví stav $\mathbb{L}$, čím vytvorí novú verziu $\mathbb{L}$. Operátor priebežne synchronizuje koreň aktuálnej verzie $\mathbb{L}$ do smart kontraktu pre zachovanie integrity (*integrity preserving smart contract* – **IPSC**) zverejneného na verejnom blockchaine. Klienti tento smart kontrakt využívajú aj pre rezolúciu cenzurovaných transakcií a dotazov. Predchádzajúce špecifikované časti a procesy boli súčasťou pôvodného návrhu jednotlivých inštancií Aquarea.

Rozšírenie o interoperabilitový protokol vyžaduje vytvorenie komunikačného kanálu medzi externými klientmi[2] a operátorom (na obrázku 5.1 je znázornená prerušovanou čiarou

[1] Pre lepšiu rozlíšiteľnosť, mikro kontraktom označuje smart kontrakt vykonávaný enklávou centralizovanej platformy a smart kontrakt označuje štandartný smart kontrakt nasadený vo verejnom blockchaine.

[2] Klient inej inštancie.

čiernej farby). Externí klienti tento kanál využívajú len na získanie inkrementálných dôkazov. Avšak môže nastať situácia, kedy by operátor cenzuroval takéto dotazy, a preto externý klient musí byť schopný rovnako použiť také prostriedky pre rezolúciu cenzúry ako interní klienti. Pre vyžiadanie rezolúcie cenzurovaného dotazu musí externý klient disponovať prideleným prístupovým lístkom k IPSC kontraktu. Tento lístok vydáva enkláva pre konkrétnu adresu na dobu určitú. Jedná sa o ochranný mechanizmus operátora v prípade DoS útokov klientov, keďže každá požiadavka o rezolúcia stojí operátora prostriedky a je jeho povinnosťou ich vyriešiť. Protokol je navrhnutý tak, aby nepokračoval v prípade, ak by klient nedisponoval týmto lístkom.

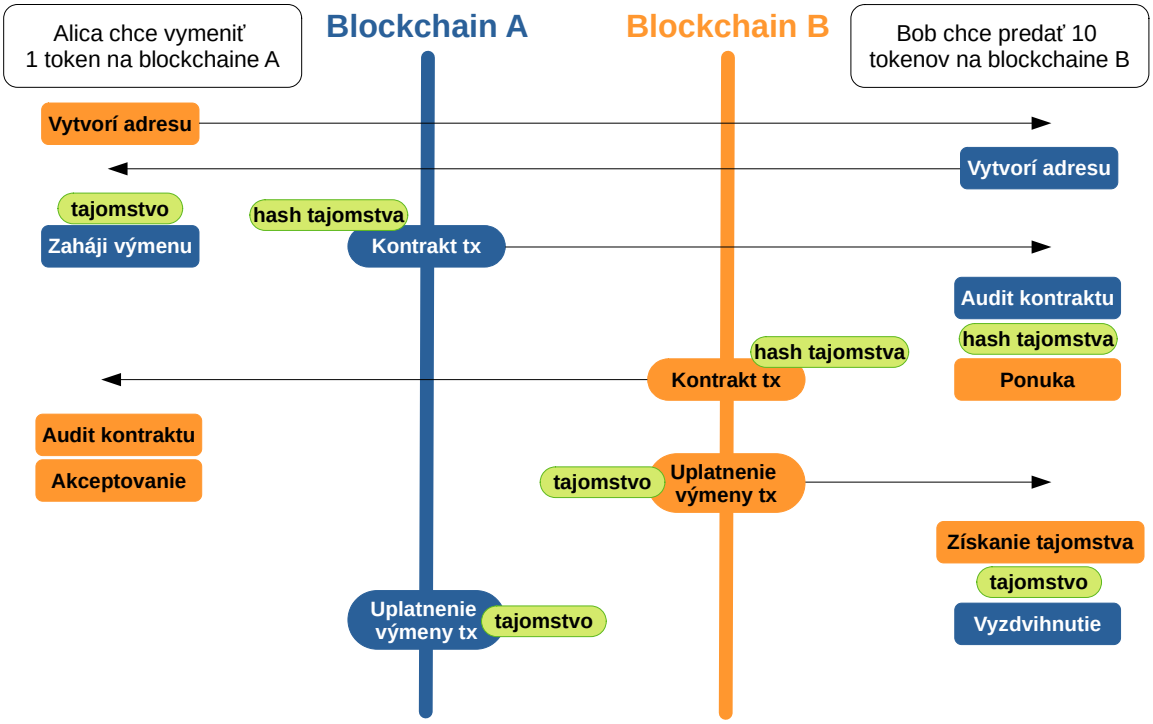
Pre jednoznačné identifikovanie klienta naprieč inštanciami je nevyhnutné špecifikovať identifikátor inštancie, v ktorej má klient vytvorený účet. Inštancie sú jednoznačne identifikovateľné pomocou adresy IPSC kontraktu vo verejnom blockchaine. V tomto smart kontrakte sa ďalej okrem spomínaných metód a aktuálneho stavu $\mathbb{L}$ nachádza aj verejný kľúč operátora a dva kľúče enklávy. Jeden pre interakciu s verejným blockchainom a druhý pre vzdialenú atestáciu TEE. Z toho dôvodu, pri použití tejto adresy ako identifikátora nie je nutné v protokole zasielať spomínané kľúče. V prípade potreby je ich možné získať jednoducho pomocou light klienta.

Pred zahájením interoperabilitového protokolu s externým klientom je potrebné overiť, či daná inštancia podporuje tento protokol. Pre jednoduché overenie je vytvorený ďalší smart kontrakt vo verejnom blockchaine, ktorý spravuje identity podporovaných inštancií (*identity managment smart contract* – **IMSC**). Identita inštancií je reprezentovaná rovnako ako pri jednoznačnej identifikácií klienta, a to IPSC adresou inštancie. Adresa IMSC je tak pre všetky inštancie rovnaká, a preto môže byť napevno uložená v enklávach všetkých podporujúcich inštancií. Za účelom zamedzenia tvorby jednej centrálnej entity spravujúcej tento zoznam, prihlásenie novej inštancie do IMSC musí schváliť nadpolovičná väčšina už prihlásených inštancií.

Samotná interoperabilitová logika je zabezpečovaná interoperabilitovým mikro kontraktom (*interoperability micro contract* – **IOMC**) a enklávou, ktorá s využitím light klienta vykonáva overovanie kryptografických dôkazov. Pre lepšiu prehľadnosť je IOMC kontrakt rozdelený na dva – jeden určený na posielanie, druhý na prijímanie prostriedkov. Predpokladá sa, že obidva kontrakty sú operátorom nasadené ihneď po vytvorení inštancie. Pred volaním vybraných funkcií transakciou, je nutné vykonať špeciálne úkony, ktoré sú detailnejšie rozpísané v sekcii 5.2.1.

5.2.1 Protokol pre prevod natívnych krypto tokenov

Protokol zasielania natívnych krypto tokenov medzi jednotlivými inštanciami kryptomien musí spĺňať atomické vlastnosti – buď sa vykoná celý, alebo ani jedna jeho časť. Inšpiráciou navrhovaného protokolu je protokol *Atomic Swap* [29], ktorý umožňuje výmenu krypto tokenov dvoch navzájom nedôveryhodných strán medzi rozličnými blockchain sieťami bez použitia tretej dôveryhodnej strany. Pri takejto výmene je vyžadované, aby obe strany deklarovali podmienené odoslanie druhej strane v stanovenom časovom okne. Podmienené odoslanie je založené na kryptografickej hash funkcii. Iniciátor výmeny vytvorí tajomstvo (*preimage*), na ktoré následne aplikuje hash funkciu. V ďalšom kroku vytvorí podmienenú transakciu v blockchain sieti A, ktorá sa vykoná až po zverejnení tajomstva. Keď druhá strana registruje vzniknutú podmienenú transakciu, vytvorí rovnakú transakciu v druhej blockchain sieti B s rozdielom, že nevytvára nové tajomstvo, ale použije hash vytvorený iniciátorom. Ak iniciátor detekuje zverejnenú transakciu v sieti B, zverejnením tajomstva



Obr. 5.2: Postupnosť krokov protokolu *Atomic Swap*.

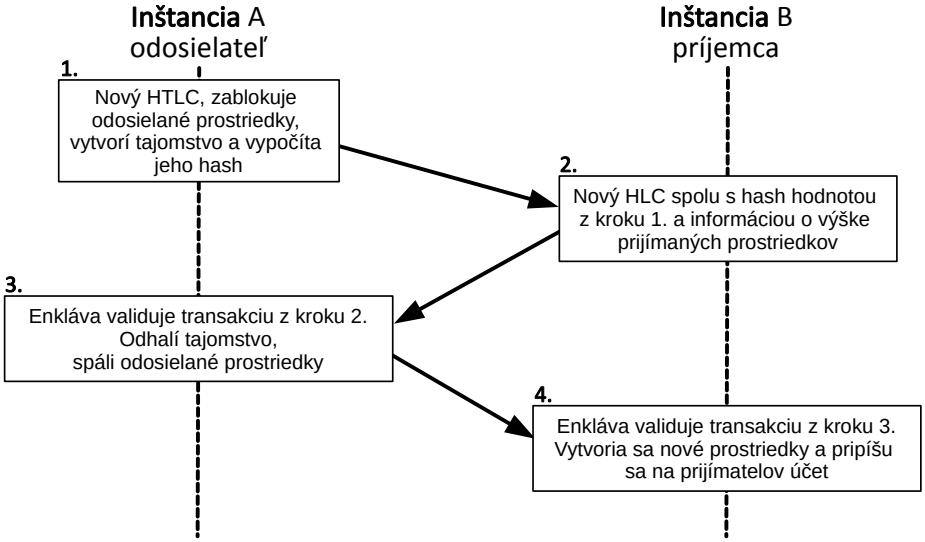
akceptuje prijímané tokeny. Druhá strana použije zverejnené tajomstvo k dokončeniu transakcie vytvorenej iniciátorom v sieti A, a tým zároveň taktiež akceptuje prijímané tokeny, čím protokol úspešne skončí. Na obrázku 5.2 je znázornená postupnosť krokov vedúcich k úspešnému záveru.

Keďže vytvorením podmienenej transakcie sú odosielané tokeny klientovi zablokované, protokol používa tzv. HTLC (*Hashed Timelock Contract*) kontrakty, ktoré v prípade nepreukázania tajomstva do určitej doby strácajú platnosť, pričom zablokované tokeny si môže odosielateľ znovu pripísať, čím sa transakcia anuluje.

Navrhovaný protokol. Oproti protokolu *Atomic Swap* sa v interoperabilitovom protokole pri prenášaní krypto tokenov zasielajú prostriedky iba od jednej strany k druhej, čím nedochádza k žiadnej výmene. Z tohto dôvodu nie je potrebné uvažovať o dvoch HTLC kontraktoch, ale iba o jednom a to na strane odosielateľa. Odosielateľ v prvom kroku vytvorí tajomstvo, na ktoré následne aplikuje hash funkciu. Výsledok odovzdá ako argument do vzniknutého HTLC kontraktu, ktorý zablokuje odosielané prostriedky. Ak nebudú prostriedky odoslané zverejnením tajomstva do určitej doby, odosielateľ si ich môže pripísať späť.

V ďalšom kroku odosielateľ oboznámi príjemcu o detailoch prevodu. Príjemca použije tieto informácie pri vytváraní podmieneného kontraktu HLC (*Hashed Lock Contract*)[3] v inštancii B a ďalej čaká na zverejnenie tajomstva. Príjemca informuje odosielateľa o registrovaní jeho požiadavky spoločne s dôkazom o vytvorení HLC kontraktu s korektnými informáciami. V treťom kroku odosielateľ overí získaný dôkaz pomocou enklávy. Ak je dôkaz potvrdený, odosielateľ odhalí tajomstvo a odosielané prostriedky sú v inštancii A odstrá-

[3]Kontrakt podobný HTLC, avšak HLC nemá žiaden termín, po ktorom by prestal platiť.



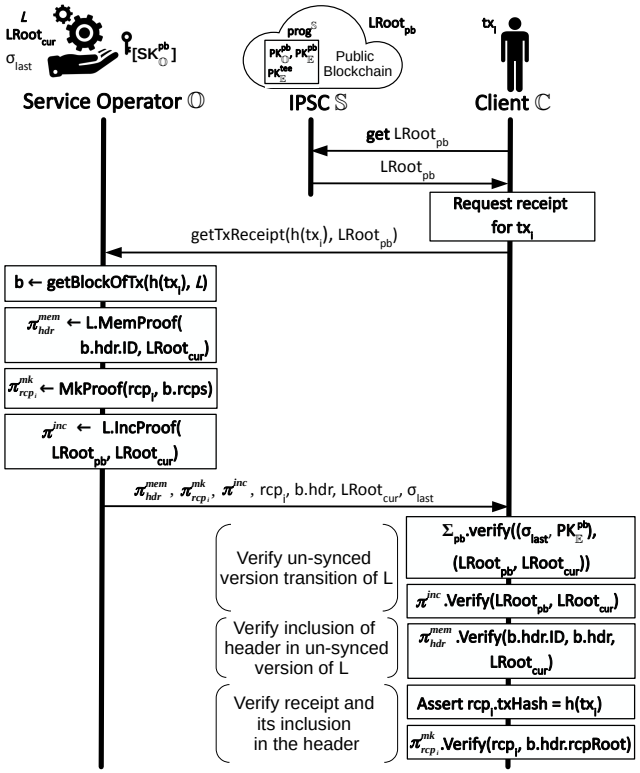
Obr. 5.3: Zjednodušená logika inteoperabilitového protokolu.

nené. Odosielateľ odošle príjemcovi dôkaz o dokončení prevodu spoločne s tajomstvom. V poslednom, štvrtom kroku príjemca pomocou enklávy overí dôkaz a dosadí tajomstvo do transakcie. Ak je všetko v poriadku, vytvoria sa nové krypto tokeny, ktoré sa pripíšu na účet príjemcu v inštancii B. Diagram takto popísaného protokolu je znázornený na obrázku 5.3. Avšak jedná sa o príliš zjednodušený a abstraktný model navrhovaného protokolu, ktorý neoddeľuje jednotlivé entity v inštanciách, a preto nad ním môžme uvažovať ako o zjednodušenom pohľade na základnú logiku protokolu.

Ako vyplýva aj zo zjednodušeného diagramu, protokol pozostáva zo štyroch fáz, pričom každá pozostáva z viacerých krokov. V nasledujúcom texte je detailne rozobraná postupnosť krokov každej z fázy. Na konci kapitoly je na obrázku 5.9 zobrazený kompletný navrhnutý protokol pre prevod natívnych krypto tokenov medzi klientom $\mathbb{C}_A$, z inštancie A, a klientom $\mathbb{C}_B$, z inštancie B, zahrňujúci všetky potrebné entity.

Fáza 1 – Inicializácia protokolu klientom $\mathbb{C}_A$. Klient $\mathbb{C}_A$ (odosielateľ) vytvorí transakciu s hodnotou odosielanej sumy, ktorá invokuje metódu `sendInit()` v interoperabilitovom mikro kontrakte určenom pre odosielanie. Argumentami tejto transakcie je adresa externého klienta B (príjemca), adresa $IPSC$ kontraktu inštancie, v ktorej je klient B registrovaný a hash vopred vytvoreného tajomstva. Takto vytvorenú transakciu podpíše vlastným privátnym kľúčom a odošle operátorovi. Operátor transakciu prijme a prepošle enkláve. Enkláva pred samotným vykonaním transakcie zabezpečí, že externý príjemca ma pridelený prístupový lístok pre uverejňovanie požiadavkov na vyriešenie cenzurovaných transakcií do $IPSC_A$, ktorý platí minimálne po celú dobu definovanú podla HTLC kontraktu. Ak enkláva zistí, že externému klientovi ešte nebol vytvorený prístupový lístok alebo mu skončila platnosť, vytvorí nový. Lístok sa skladá z dvojice – adresa klienta B[4] a doba vypršania platnosti reprezentovaná časovou známkou. Následne je táto dvojica podpísaná súkromným kľúčom enklávy pre podpisovaciu schému verejného blockchainu (PK_E^{pb}). V ďalšom kroku je vykonaná transakcia, ktorá vytvorí v mikro kontrakte nový záznam o prevode so zadanými vstupnými informáciami. Odosielané prostriedky odosielateľa sú prevedené na adresu mikro

[4]Predpokladá sa, že externý klient disponuje rovnakou adresou aj vo verejnom blockchaine.

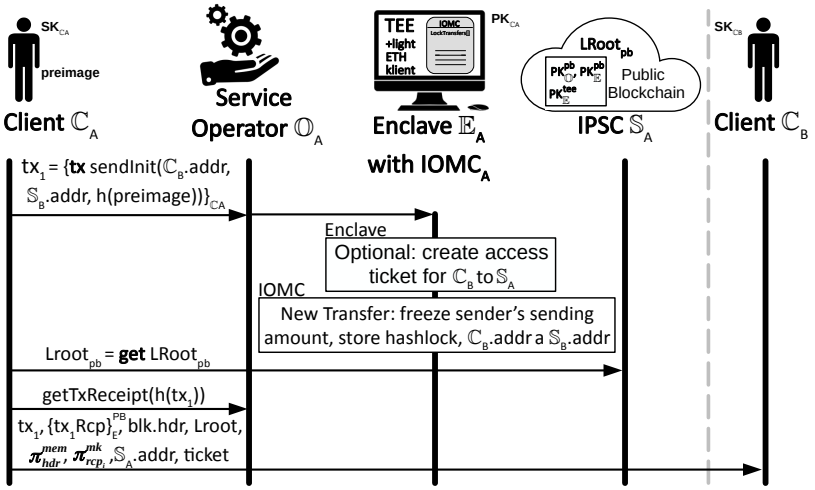


Obr. 5.4: Protokol pre získanie potvrdenia o vykonaní transakcie tx_i (prevzaté z [32]).

kontraktu, čím stráca nad nimi plnú kontrolu a ich pohyb je ďalej definovaný metódami kontraktu.

Už z pôvodného návrhu Aquarea [32] vyplýva, že klienti po odoslaní transakcie nedostanú automatickú odpoveď s potvrdením o jej vykonaní, v ktorom sa nachádza hash transakcie, návratová hodnota virtuálneho stroja po vykonaní transakcie ako aj log emitovaných udalostí, a celé potvrdenie je podpísané PK_E^{pb}. Je tak v záujme klienta si tieto informácie vyžiadať. Klient vytvorí žiadosť na základe hash hodnoty transakcie a aktuálne získaného koreňa $\mathbb{L}$ z verejného blockchainu. Okrem potvrdenia o vykonaní transakcie dostane aj sadu kryptografických dôkazov (π_{hdr}^{mem}, π_{rcp}^{mk}, π^{inc}), hlavičku bloku, v ktorej je zapísaná transakcia ako aj aktuálnu hodnotu koreňa účtovej knihy $LRoot_{cur}$. Spomínané dôkazy slúžia pre overenie klientom, že daná transakcia bola reálne vykonaná. Operátor pri vytváraní dôkazov najprv nájde blok b, ktorý obsahuje vyžadovanú transakciu a vypočíta dôkaz o členstve hlavičky bloku b v aktuálnej verzii účtovej knihy $\mathbb{L}$, čím vznikne π_{hdr}^{mem}. Druhým vypočítaným dôkazom je Merklov dôkaz svedčiaci o tom, že potvrdenie o vykonaní transakcie rcp_i je zhrnuté v bloku b. Nakoniec operátor vypočíta inkrementálny dôkaz π^{inc} prechodu $\langle LRoot_{pb}, LRoot_{cur}\rangle$ – od verzie vo verejnom blockchaine k najnovšej verzii, ktorá vznikla po vykonaní transakcie. Grafická podoba protokolu pre získanie potvrdenia o vykonaní, ktorá je prevzatá z pôvodnej publikácie Aquarea [32], sa nachádza na obrázku 5.4.

Po získaní potvrdenia spolu s ostatnými dôkazmi, klient $\mathbb{C}_A$ odosiela priamym komunikačným kanálom klientovi $\mathbb{C}_B$ vykonanú transakciu a jej potvrdenie, kryptografické dôkazy π_{hdr}^{mem}, π_{rcp}^{mk} spoločne s hlavičkou bloku, v ktorej je transakcia zahrnutá, hodnotou koreňa $\mathbb{L}$ po vykonaní transakcie, adresu $IPSC_A$ a validným prístupovým lístkom pre klienta $\mathbb{C}_B$.



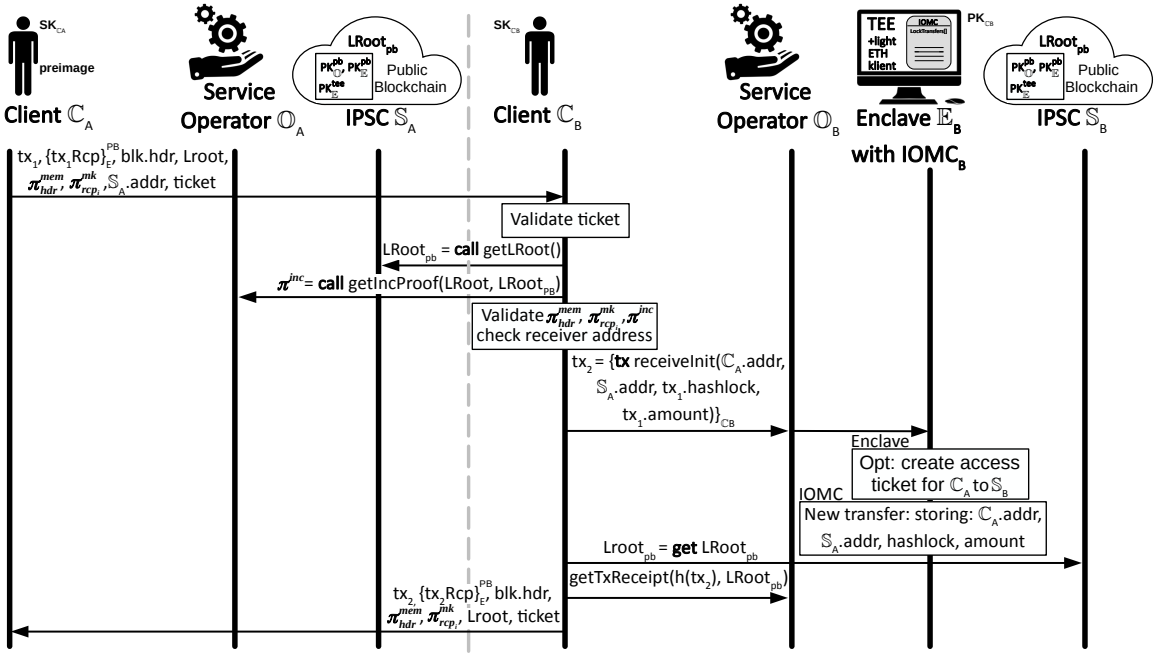
Obr. 5.5: Prvá fáza interoperabilitového protokolu.

Následne sa protokol presúva do druhej fázy. Komunikačný diagram tejto fázy je znázorený na obrázku 5.5.

Fáza 2 – zaznamenanie transakcie o prijímaní klientom $\mathbb{C}_B$. Fáza začína prijatím správy od klienta $\mathbb{C}_A$. V prvom kroku, klient $\mathbb{C}_B$ validuje prístupový lístok k $IPSC_A$ pomocou verejného kľúča enklávy prístupného v tomto kontrakte. Bez validného lístka by protokol ďalej nepokračoval, z dôvodu rizika cenzúry. V ďalšom kroku, získa z verejného blockchainu koreň $\mathbb{L}$. Tento krok je vykonávaný za účelom uistenia sa klientom B, že stav, ktorý je zverejnený v $IPSC_A$, obsahuje prijímanú transakciu, a tým pádom je nevrátiteľný späť v čase. Po získaní koreňa z verejného blockchainu ho odošle spoločne so získaným koreňom od odosielateľa operátorovi na vytvorenie inkrementálneho dôkazu $\langle LRoot_{cur}, LRoot_{pb} \rangle$. V prípade cenzúrovania tohoto dotazu sa môže obrátiť na $IPSC_A$ a verejne požiadať o rezolúciu dotazu.

Po získaní dôkazu a následnej validácií môže pristúpiť k validácií zvyšných dôkazov zaslaných klientom $\mathbb{C}_A$ spoločne s overením prijímacej adresy. Klient postupuje do ďalšieho kroku až po úspešnom overení zvyšných dôkazov, pričom až v tomto momente si môže byť istý, že klient $\mathbb{C}_A$ zadal platbu na jeho účet. Ďalší krok pozostáva z vytvorenia transakcie tx_2 invokujúcej metódu `receiveInit()` s argumentami: adresa klienta $\mathbb{C}_A$ získaná z transakcie tx_1, adresa $IPSC_A$ inštancie, v ktorej ma klient $\mathbb{C}_A$ účet, hash hodnota tajomstva a počet posielaných krypto tokenov získaných taktiež z tx_1. Vytvorenú transakciu odošle operátorovi svojej inštancie, ktorý ju predá enkláve na spracovanie.

Enkláva, rovnako ako v prvej fáze, zistí, či externý klient, v tomto prípade klient $\mathbb{C}_A$, má vytvorený prístupový lístok s dostatočne dlhou dobou platnosti. V prípade, že daný stav tomu nezodpovedá, enkláva zabezpečí jeho vytvorenie. V ďalšom kroku enkláva vykoná prijatú transakciu, čím vytvorí nový záznam v IOMC kontrakte. V zázname sú uložené všetky vstupné argumenty transakcie. Vzniknutý identifikátor záznamu je zapísaný do udalosti emitovanej transakcie, čím je tento krok považovaný za dokončený. Klient $\mathbb{C}_B$, v ďalšom kroku, získa koreň $\mathbb{L}$ z verejného blockchainu a vyžiada od operátora $\mathbb{O}_A$ potvrdenie o vykonaní transakcie (podobne, ako je tomu vo fáze č. 1). V závere druhej fázy odošle klient $\mathbb{C}_B$ klientovi $\mathbb{C}_A$ správu s vykonanou transakciou tx_2, jej potvrdenie, kryptografické dôkazy π^{mem}_{hdr}, π^{mk}_{rcp}, spoločne s hlavičkou bloku, v ktorej je transakcia tx_2 zahrnutá, hodnotou ko-



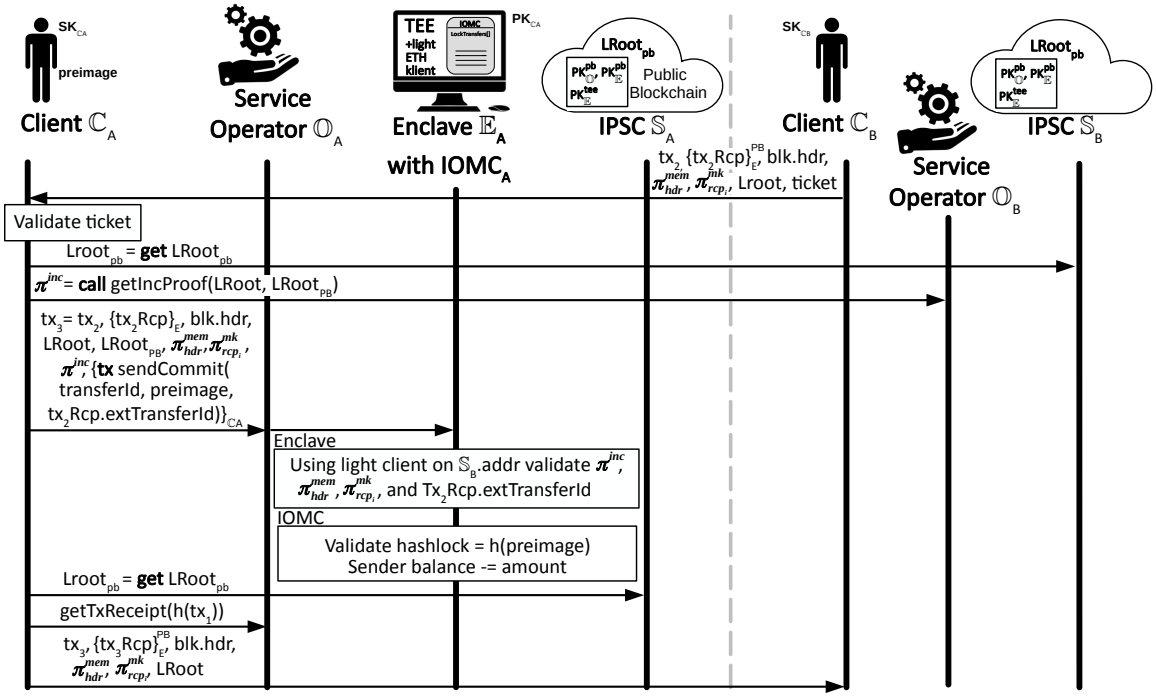
Obr. 5.6: Druhá fáza interoperabilitového protokolu.

reňa $\mathbb{L}$ po vykonaní transakcie a validným prístupovým lístkom klienta $\mathbb{C}_A$. Komunikačný diagram fázy č. 2 je znázorený na obrázku 5.6.

Fáza 3 – potvrdenie prevodu klientom $\mathbb{C}_A$. Fáza začína prijatím správy od klienta $\mathbb{C}_B$. V prvom kroku vykoná klient validáciu prijatého prístupového lístka k $IPSC_B$, ktorá musí pre ďalšie pokračovanie protokolu skončiť úspešne. V nasledujúcom kroku získa z $IPSC_B$ koreň $\mathbb{L}$. Rovnako ako v predošlej fáze je nevyhnutné, aby stav, ktorý je zverejnený v $LRoot_{pb}$ v sebe zahrňoval transakciu tx_2, čo by znamenalo, že je aktuálnejší ako koreň prijatý od klienta. Po získaní koreňa sú oba korene zaslané externému operátorovi, ktorý vyprodukuje dôkaz overujúci daný predpoklad. Ak dôkaz nie je možné overiť, klient počká na aktualizáciu koreňa vo verejnom smart kontrakte a dotaz opakuje. Získaný platný dôkaz spoločne s ostatnými prijatými dôkazmi musia byť overené aj enklávou ešte pred vykonaním transakcie. Do enklávy sú zahrnuté ako argumenty transakcie, ktoré však nie sú podpísané klientom $\mathbb{C}_A$, keďže sú spracované ešte skôr ako transakcia a do EVM sa už nepredávajú. Z toho dôvodu už nie je nevyhnutné ich posielať ďalej klientovi $\mathbb{C}_B$ v transakcii tx_3 na konci fázy.

Vytvorená transakcia tx_3 pozostáva z invokovania metódy `sendCommit()` spolu so zverejneným tajomstvom a identifikátorom záznamu externého prijímajúceho prevodu. Transakcia je spoločne s ostatnými prijatými argumentami odoslaná operátorovi (viď. obrázok 5.7). Operátor predá prijatú transakciu do enklávy, ktorá ešte pred samotnou realizáciou transakcie v EVM vykoná nasledujúce overenia:

- pomocou light klienta overí správnosť $LRoot_{pb}$. Táto kontrola prebieha z toho dôvodu, aby si operátor overil, že prijatý koreň je naozaj skutočný – t.z. že ho nevytvoril klient.
- postupne validuje π^{inc}, π^{mem}_{hdr}, π^{mk}_{rcp}



Obr. 5.7: Tretia fáza interoperabilitového protokolu.

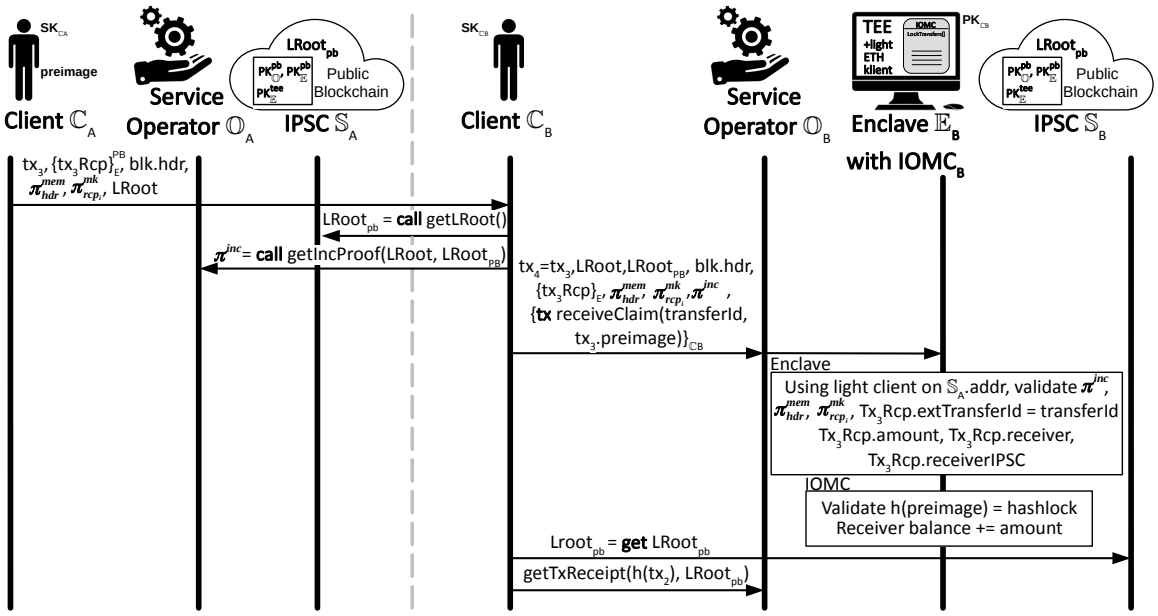
- identifikátor záznamu externého prijímajúceho prevodu sa zhoduje s identifikátorom z udalosti v potvrdení o vykonaní – t.z. klient ho nepozmenil

Po úspešnej validácií dôkazov je vykonaná transakcia, ktorá v prvom rade validuje správnosť tajomstva a následne definitívne zlikviduje zasielané krypto tokeny. Po vykonaní transakcie si klient vyžiada od operátora jej potvrdenie, ktoré spolu s vykonanou transakciou odošle komunikačným kanálom klientovi $\mathbb{C}_B$.

Fáza 4 – prijatie krypto tokenov klientom $\mathbb{C}_B$. Po prijatí správy od klienta $\mathbb{C}_A$, klient $\mathbb{C}_B$ získa $LRoot_{pb}$ z $IPSC_A$ a zároveň si vyžiada od operátora inštancie A inkrementálny dôkaz získaného koreňa. Ak dôkaz nie je možné overiť, klient počká na aktualizáciu koreňa vo verejnom smart kontrakte a dotaz opakuje. Následne vytvorí transakciu invokujúcu funkciu `receiveClaim()` s identifikátorom prevodu a zverejneným tajomstvom odosielajúceho klienta. Spomínané dva argumenty sú podpísane klientom $\mathbb{C}_B$, pričom transakcia zároveň obsahuje aj všetky zvyšné prijaté dáta od klienta $\mathbb{C}_A$, ktoré však už nie sú podpísané.

Vytvorená transakcia tx_4 je odoslaná operátorovi, a ten ju ďalej posúva enkláve. Pred samotným vykonaním transakcie v EVM enklávy, sú vykonané rovnaké prvé dve overenia ako vo fáze 3. Tretie overenie pozostáva z kontroly argumentov emitovaných udalosťou v tretej fázy. Tá sa nachádza v odpovedajúcom potvrdení o vykonaní podpísanom enklávou $\mathbb{E}_A$ a obsahuje identifikátor prevodu, adresu príjemcu spoločne s adresou inštancie a počet odosielaných krypto tokenov, pričom platí:

- Identifikátor prevodu musí byť zhodný s identifikátorom zaslaným v transakcii tx_4.
- Adresa príjemcu musí byť zhodná s adresou, ktorá vytvárala transakciu v druhej fáze.
- Zasielaná adresa inštancie je totožná s pravou adresou.



Obr. 5.8: Štvrtá fáza interoperabilitového protokolu.

- Počet prijímaných tokenov odpovedá informácii uloženej v prijímateľovom IOMC kontrakte pod daným identifikátorom.

Nasleduje vykonanie transakcie tx_4, ktorá overí pravosť tajomstva, vytvorí a pripíše krypto tokeny na účet klienta $\mathbb{C}_B$. Následným úspešným overením potvrdení o vykonaní transakcie vyžiadaných od operátora, protokol úspešne končí. Komunikačný diagram poslednej štvrtej fázy je zobrazený na obrázku 5.8.

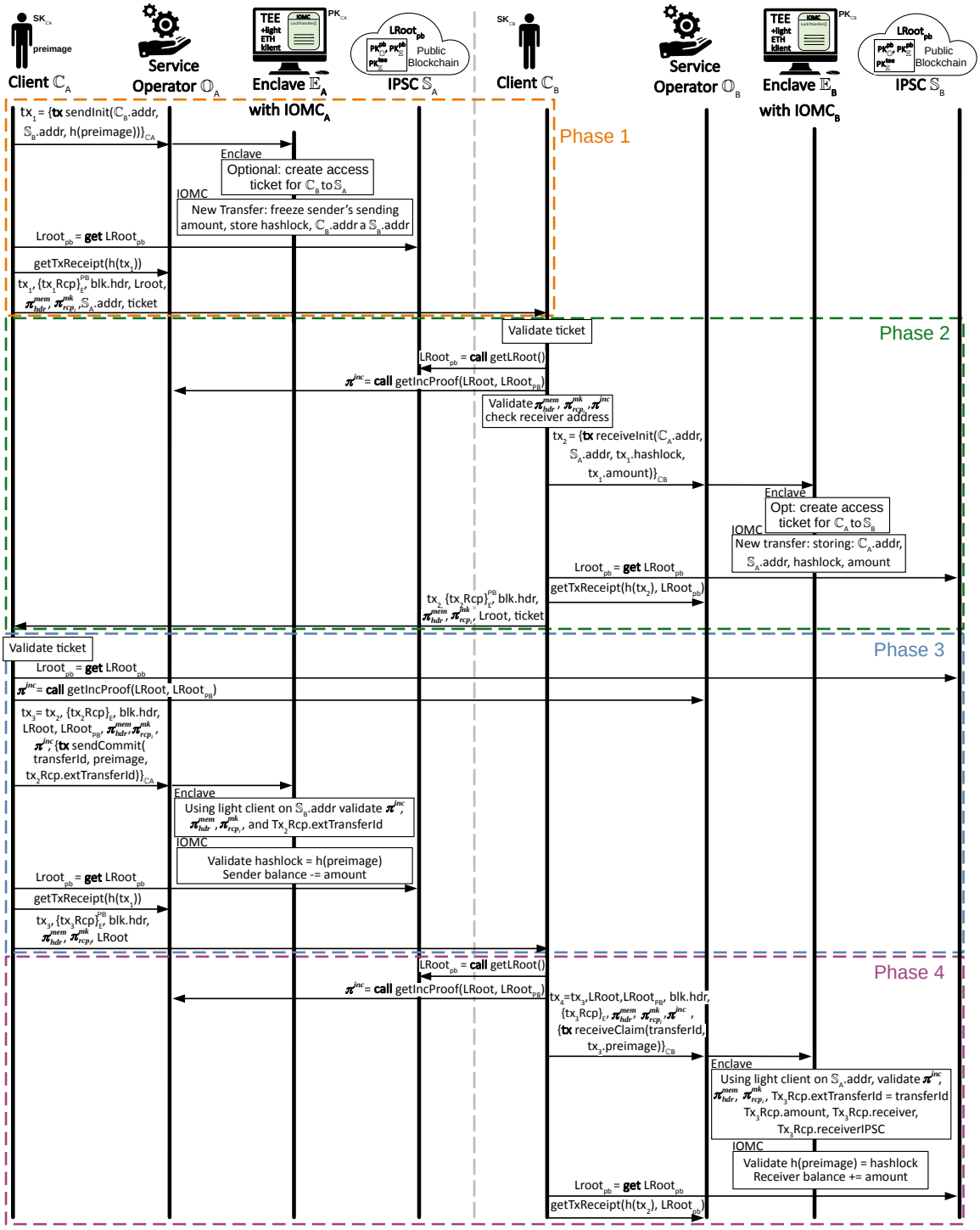
Zrušenie prevodu odosielateľom. Odosielateľovi je umožnené do začatia fázy č. 3 prevod zrušiť. Toto rozhodnutie môže vykonať napr., ak zadal nesprávne údaje v prvej fáze, ak druhá strana nespolupracuje, ak transakcia vytvorená prijímateľom v druhej fáze neodpovedá zasielanými parametrom alebo sa môže bezdôvodne rozhodnúť ďalej nepokračovať. V takomto prípade musí počkať na uplynutie doby definovanej v IOMC kontrakte od vytvorenia prevodu. Po tomto termíne je možné zablokované prostriedky naspäť odosielateľovi uvolniť pomocou transakcie inovokujúcej funkciu `sendRevert()` s identifikátorom prevodu. Táto funkcia mikro kontraktu je navrhnutá tak, aby po úspešnom vykonaní tretej fázy konkrétneho prevodu nebolo možné zrušiť prevod týmto postupom.

5.2.2 Externé invokovanie mikro kontraktov

Existujú dve varianty:

- Externé invokovanie mikro kontraktov so zmenou oboch účtových kníh (internej aj externej) – viacfázový protokol založený na podobnom princípe.
- Externé invokovanie mikro kontraktov so zmenou len jednej účtovej knihy, a to externej – jednofázový protokol invokujúci externý mikro kontrakt.

Avšak táto práca je zameraná výhradne na prevod natívnych krypto tokenov, a preto nebol vytvorený kompletný návrh žiadneho zo spomínaných variantov externých invokácií mikro kontraktov.



Obr. 5.9: Protokol pre prevod natívnych krypto tokenov.

Kapitola 6

Implementácia

Práca je založená na základnej *proof-of-concept* implementácii decentralizovanej smart kontraktovej platformy Aquareum v jazyku C++ a technológie Intel SGX pre inštanciáciu enklávy. IPSC kontrakt vo verejnom blockchaine je postavený na jazyku Solidity a je pripravený na nasadenie na Ethereum blockchain. Enkláva využíva vývojový nástroj OpenEnclave SDK,[1] ktorého jednou z najdôležitejších výhod je univerzálnosť. Tento nástroj je možné používať s viacerými technológiami TEE a nad rôznymi operačnými systémami. Aquareum integruje v enkláve virtuálny stroj Etherea – EVM, v jeho minimalistickej verzii eEVM.[2]

Cieľom implementácie práce bolo tak rozšíriť aktuálny stav Aquarea o navrhované časti protokolu. Avšak je nutné poznamenať, že niektoré potrebné časti programu z pôvodného návrhu neboli riadne implementované, a preto ich bolo nutné ešte pred samotnou implementáciu interoperabilitového protokolu doprogramovať. Jedná sa najmä o klientsky program komunikujúci s operátorom pomocou TCP/IP protokolu. V nasledujúcich sekciách sú detailnejšie rozpracované implementačné detaily tých častí systému, ktoré boli v rámci tejto práce vytvorené, prípadne pozmenené.

6.1 Klientsky program

V rámci implementácie bol zhotovený program v jazyku C++ umožňujúci klientovi pristúpiť k operátorovi a interagovať tak s účtovou knihou. Pri prvom štarte program vygeneruje súkromný a verejný kľúč, ktorý je zároveň uložený v binárnej podobe do súboru. Veľkosť súkromného kľúča je 256 bitov a veľkosť verejného kľúča je 512 bitov. Následne môže používateľ požiadať operátora o registráciu pomocou vytvoreného verejného kľúča, čím sa nasledne stane klientom danej inštancie. Momentálne je registrácia potvrdená každému používateľovi, ktorý o to zažiada. Je zrejmé, že pri finálnom riešení bude nutné zároveň odosielať aj identitu používateľa, ktorú operátor overí. Pri opakovanom spustení klientského programu je nežiadúce vytváranie nových kľúčov, a preto sú zo súboru načítané predom vytvorené kľúče.

Program umožňuje klientovi vytvárať transakcie troch typov:

- Platobnú transakciu pre prenos natívnych krypto tokenov na adresu klienta v rámci danej inštancie. V takomto prípade klient zadá príkaz `pay a b`, kde `a` znamená počet zasielaných krypto tokenov a `b` je adresa príjemcu.

[1] https://openenclave.io/sdk/

[2] Enclave EVM vyvíjaný spoločnosťou Microsoft https://github.com/microsoft/eEVM

- Invokovanie funkcie už nasadeného mikro kontraktu. V takomto prípade je nutné poznať adresu mikro kontraktu spoločne s 32-bitovým hashom invokovanej funkcie.
- Platobnú transakciu na adresu externého klienta pomocou špeciálnych predpripravených „iomc“ príkazov. Príkazy podľa aktuálnej vykonávanej fázy vytvoria transakciu invokujúcu konkrétnu funkciu v prislúchajúcom mikro kontrakte. Je nutné podotknúť, že pred použitím týchto príkazov je nutné zistiť od operátora adresy IOMC kontraktov pomocou príkazu `iomc addr`.

6.2 Aquareum

V pôvodnej proof-of-concept implementácií mohol vytvárať transakcie jedine operátor pomocou terminálového zadávania príkazov. V rámci tejto práce bol tento vstup od operátora ponechaný a zároveň bola doplnená serverová časť, ktorá slúži na obsluhovanie klientov (viac v podsekcii 6.2.1). Klienti môžu rovnako ako operátor vytvárať transakcie, ktoré sa po zaslaní na server vykonajú. Týmto spôsobom vznikajú viaceré miesta v programe, kde by boli vykonávané transakcie po jednom, čo nepridáva na efektívnosti a prehľadnosti programu. Pre vyriešenie týchto problémov bol zavedený a implementovaný dispečer transakcií. Jedná sa tak o jediné miesto v programe, ktoré spravuje a vykonáva prijaté transakcie od klientov, ale aj od operátora. Implementácii dispečera je venovaná podsekcia 6.2.2. Hlavná interoperabilitová logika z návrhu je založená na IOMC kontraktoch vykonávaných v enkláve (viď sekcia 6.2.3).

6.2.1 Server

Serverová časť je implementovaná ako konkurentný neblokujúci server vykonávaný v samostatnom vlákne. Po nadviazaní TCP komunikácie s klientom, server vytvorí ďalšie vlákno obsluhujúce prijatého klienta. Komunikácia je koncipovaná tak, aby sa po každej odoslanej transakcii spojenie medzi klientom a serverom uzatvorilo. Na obrázku 6.1 je zobrazená komunikácia jednotlivých častí systému vrátane klient-server komunikácie.

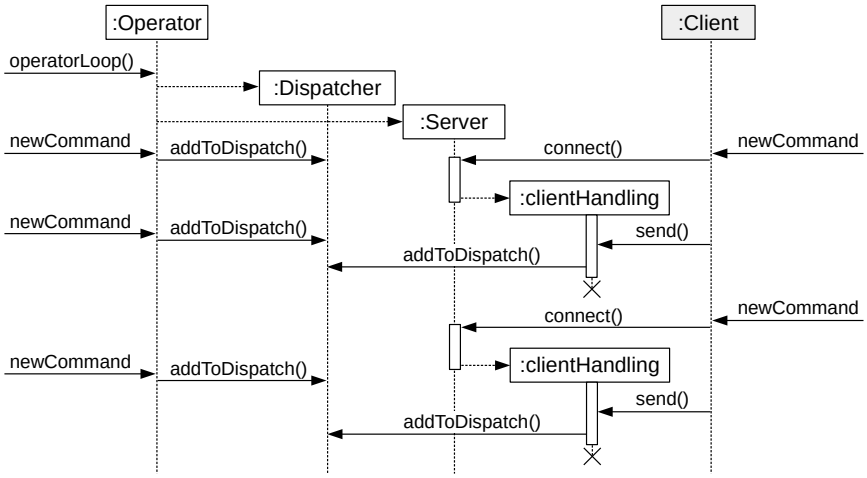
Správy zasielané klientom majú danú štruktúru. Prvý bajt správy definuje jej typ. V rámci práce sú implementované následujúce 3 typy správ:

- vykonanie transakcie,
- registrácia klienta,
- dotaz na adresy IOMC kontraktov.

Prvý typ správy v sebe zahrňuje transakciu, ktorej veľkosť sa odvíja od jej typu a počtu argumentov. Takto prijatá transakcia je zaradená do poľa transakcií čakajúcich na vykonanie. Prijatím registračnej správy spolu s 512-bitovým verejným kľúčom server vytvorí registračnú transakciu, a taktiež ju zaradí do spomínaného poľa. Dotaz na adresy IOMC kontraktu nenesie žiadne ďalšie argumenty. Pri tomto type správy sa zároveň klientovi odošlú dotazované adresy.

6.2.2 Dispečer transakcií

Vykonávanie transakcií vo vlákne, ktoré vytvorí, resp. príjme transakciu, nie je efektívne, pretože v prípade veľkého množstva prijatých transakcií by sa každá vykonávala samos-



Obr. 6.1: Komunikácia medzi klientmi a jednotlivými časťami Aquarea. Každý z objektov reprezentuje samostatné vlákno, pričom klient reprezentuje samostatný program.

tatne a procesor by tak musel obsluhovať veľké množstvo prepínaní kontextu medzi enklávou a užívateľským procesom, čo je časovo náročné. Efektívnejšie je vykonávať prijaté transakcie vo väčších množstvách. Z tohto dôvodu je do programu zavedený dispečer, ktorý riadi vykonávanie všetkých transakcií. Jedná sa o samostatné vlákno a dátovú štruktúru `std::vector` so synchronizačným mechanizmom v podobe zámku. Pre zápis a čítanie štruktúry je nutné k nej získať výlučný prístup. Ostatné vlákna pridávajú prijaté transakcie do dátovej štruktúry pomocou definovanej metódy. Akonáhle štruktúra nie je prázdna, dispečer získa výlučný prístup, v dôsledku čoho vymení vektor za nový – prázdny, odomkne zámok a vykoná všetky zaradené transakcie v pôvodnom vektore pri jednom volaní enklávy. Po ich vykonaní overí aktuálny stav vektora. Ak sa v ňom nachádzajú transakcie, dispečer proces opakuje. Ak je prázdny, vlákno pasívne čaká na zobudenie, ktoré prichádza po vložení novej transakcie.

6.2.3 Interoperabilita

Základom interoperabilitového protokolu každej inštancie je:

- Interoperabilitový mikro kontrakt (IOMC) vykonávaný v EVM enklávy.
- Overenia dôkazov vyprodukovaných enklávou inštancie externého klienta pred určitými funkciami IOMC kontraktu.

Pre podporu interoperability je potrebné podporovať oba spomínané body. Nasadenie IOMC kontraktov do pôvodnej inštancie Aquarea bez potrebných overení je nedostatočné, a takto upravenú inštanciu nie je možné považovať za podporovanú.

Interoperabilitový mikro kontrakt – IOMC je implemntovaný ako dva samostatné kontrakty nasadené v účtovej knihe každej z inštancií. Jeden zameraný na odosielanie krypto tokenov externému klientovi (viď. algoritmus 1) a druhý na ich prijímanie (viď. algoritmus 2). Nasadenie oboch kontraktov vykonáva operátor ihneď po vytvorení inštancie. Operátor aj enkláva si zaznamená vytvorené adresy kontraktov. Operátor adresy publikuje klientom. Enkláva pred každým vykonaním transakcie porovná jej cieľovú adresu s adresami

Algoritmus 1: Program odosielajúceho IOMC kontraktu

```
▷ Deklarácia typov a premenných:
  struct LockTransfer {
      address sender,
      address receiver,
      address receiverIPSC,        ▷ Adresa IPSC kontraktu prijímateľovej inštancie.
      uint256 amount,              ▷ Počet posielaných krypto tokenov.
      uint256 hashlock,            ▷ Hash tajomstva.
      uint256 timelock,            ▷ Časová značka vymedzujúca platnosť prevodu.
      bool used,                   ▷ Udáva, či bol prevod dokončený.
      bool reverted},              ▷ Udáva, či bol prevod stornovaný.
  LockTransfer[] transfers
▷ Deklarácia funkcií:
function sendInitialize(receiver, receiverIPSC, hashlock) public payable
    assert msg.value > 0;
    timelock ← block.timestamp + 24h;        ▷ Nastavenie časového zámku na 24 hodín.
    transferId ← transfers[].length;
    t ← LockTransfers(msg.sender, receiver, receiverIPSC, msg.value,
                      hashlock, timelock, false, false);   ▷ Vytvorenie nového prevodu.
    transfers[].add(t);
    emit sendInitialized(transferId);

function sendCommit(transferId, preimage, externalTransferId) public
    assert transfers[transferId].exists();   ▷ Existuje prevod s daným id.
    t ← transfers[transferId];
    assert t.hashlock = keccak256(preimage);   ▷ Hash tajomstva je zhodný.
    assert t.used = false;                     ▷ Prevod je ešte neukončený.
    t.used ← true;
    address(0).transfer(t.amount);             ▷ Zničenie krypto tokenov.
    emit sendCommited(transferId, externalTransferId, t.receiver, t.receiverIPSC,
      t.amount);

function sendRevert(transferId) public
    assert transfers[transferId].exists();   ▷ Existuje prevod s daným id.
    t ← transfers[transferId];
    assert t.used = false;                   ▷ Prevod je ešte neukončený.
    assert t.reverted = false;               ▷ Prevod nie je stornovaný.
    assert t.timelock <= block.timestamp;    ▷ Uplynula doba platnosti prevodu.
    t.sender.transfer(t.amount);             ▷ Vrátenie tokenov naspäť odosielateľovi.
    t.reverted ← true;
    emit sendReverted(transferId);
```

IOMC kontraktov. Ak dôjde k zhode, na základe invokovanej funkcie sú vykonané príslušné kroky.

Mikro kontrakty sú implementované v jazyku Solidity s použitím kompilátora vo verzii *0.4.23*. Táto verzia bola zvolená z dôvodu overenej kompatibility s integrovanou minimalistickou verziou virtuálneho stroja Etherea.

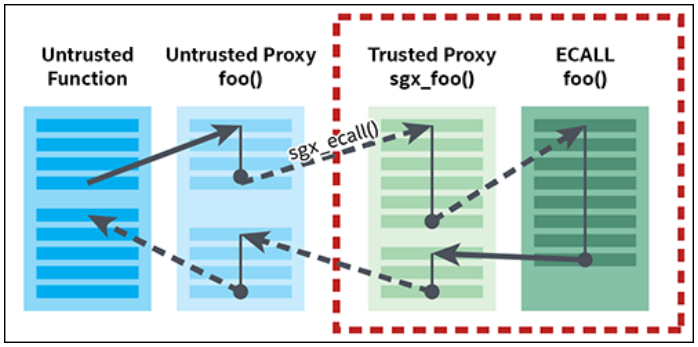
Odosielajúci IOMC vychádza z konceptu HTLC kontraktu (viac v sekcii 5.2.1). Pri takomto type kontraktu je nevyhnutné nastaviť správnu dĺžku implementovaného časového zámku. Príliš krátky zámok by znamenal nemožnosť dokončenia protokolu. Potom, čo je vytvorený nový prevod klientom $\mathbb{C}_A$ (fáza 1), je nutné počkať na synchronizáciu koreňa účtovej knihy do verejného blockchainu. Táto synchronizácia prebieha v určitých intervaloch i_A nastaveným operátorom $\mathbb{O}_A$. Zápis tejto synchronizačnej transakcie do verejného blockchainu, v tomto prípade do Etherea, trvá určitú dobu a je nepriamoúmerný poskytnutej odmene. Pre zjednodušenie môžme uvažovať v rádoch jednotkách minút. Keďže decentrali-

Algoritmus 2: Program prijímajúceho IOMC kontraktu

```
▷ Deklarácia typov a premenných:
  struct LockTransfer {
      address sender,
      address senderIPSC,                    ▷ Adresa IPSC kontraktu odosielateľovej inštancie.
      address receiver,
      uint256 amount,                        ▷ Počet prijímaných krypto tokenov.
      uint256 hashlock,                      ▷ Hash tajomstva vytvoreného odosielateľom.
      bool used},                            ▷ Udáva, či bol prevod dokončený.
  LockTransfer[] transfers
  address operator: operátor inštancie

▷ Deklarácia funkcií:
function constructor() public
    operator ← msg.sender;

function receiveInitialize(sender, senderIPSC, hashlock, amount) public
    assert amount > 0;
    transferId ← transfers[].length;
    t ← LockTransfers(sender, senderIPSC, msg.sender, amount,
                hashlock, false);            ▷ Vytvorenie prijímajúceho prevodu.
    transfers[].add(t);
    emit receiveInitialized(transferId);

function receiveClaim(transferId, preimage) public
    assert transfers[transferId].exists();   ▷ Existuje prevod s daným id.
    t ← transfers[transferId];
    assert t.hashlock = keccak256(preimage); ▷ Hash tajomstva je zhodný.
    assert t.used = false;                   ▷ Prevod je ešte neprijatý.
    if address(this).balance < t.amount then
        emit notEnoughReserve(transferId);   ▷ Účet kontraktu je nedostatočný.
    else
        t.used ← true;
        t.receiver.transfer(t.amount);       ▷ Pripísanie tokenov príjemcovi.
        emit successfulyClaimed(transferId);
```

zované blockchain siete akceptujú najdlhšiu reťaz blokov, nie je vylúčené, že ihneď po zápise danej synchronizačnej transakcie môže útočník prísť s dlhšou postupnosťou blokov a danú transakciu tak anulovať. Pravdepodobnosť, že blok zostane nemenne zapísaný sa zvyšuje každým novším zapísaným blokom a po určitom počte blokov je pravdepodobnosť prakticky zanedbateľná. Blok zapísaný v Ethereum blockchaine nadobúda finálnosť po ďalších 12-tich blokoch, čo predstavuje približne 3 minúty [13]. Následne príjemca ($\mathbb{C}_B$) realizuje fázu 2, pričom vykonáva podobné kroky ako vo fáze 1, čo znamená rovnaké zdržanie (synchronizačný interval inštancie B definuje operátor $\mathbb{O}_B$ a môže byť odlišný). Ak predpokladáme oba synchronizačné intervaly nastavené na 5 minút a zápis transakcie do bloku verejného blockchainu tiež 5 minút, dokopy sa pohybujeme okolo 26 minút. Ak prirátame rezervu na komunikáciu medzi klientmi, pre bezpečné dokončenie protokolu je vhodné nastaviť časový zámok HTLC kontraktu aspoň na 1 hodinu.

IOMC kontrakty implementujú nad natívnymi krypto tokenami koncept vytvárania a ničenia (anglicky *mint and burn*). Kontrakt určený pre odosielanie na konci funkcie `sendCommit()` zničí odosielané tokeny tým spôsobom, že ich odošle na nulovú adresu. Na druhej strane, vytváranie nových krypto tokenov v IOMC kontrakte určenom pre prijímanie je viackroková záležitosť. Samotný kontrakt nevytvára nové tokeny. Tie sú vytvárané operátorom, ktorý následne dotuje daný kontrakt. Funkcia `receiveClaim()` tak zasiela definovaný počet tokenov kontraktu na adresu príjmateľa. Avšak môže nastať situácia, kedy



Obr. 6.2: Volania `ecall` funkcie pomocou pomocných proxy funkcií (prevzaté z [2]).

stav účtu kontraktu je nižší ako posielané tokeny. V takomto prípade je pri vykonávaní transakcie vyvolaná udalosť, ktorá informuje operátora o nedostatočnom zostatku. Transakcia tak ešte nie je označená ako vykonaná, pričom po doplnení účtu operátorom pomocou transakcie `fund()` je možné ju zopakovať.

Enkláva. Programovanie enklávy pomocou vývojového nástroja OpenEnclave SDK je založené na definovaní funkcií, ktoré umožňujú prepínať kontext z užívateľského prostredia do prostredia enklávy. Tieto funkcie sú označované ako `ecall` (volania do enklávy) a `ocall` (volania z enklávy) a ich deklarácie sú uložené v `.edl` súbore. Pri zostavovaní programu je pomocou nástroja *oeedger8r*[3] a spomínaných deklarácií zhotovená sada proxy funkcií. Proxy funkcie tvoria obal okolo `ecall` a `ocall` funkcií, pričom každá takáto funkcia získa pri zostavovaní programu dve proxy funkcie – dôveryhodnú a nedôveryhodnú. Získané funkcie sú uložené v automatických generovaných súboroch a sú zahrnuté pri preklade. Každé volanie funkcie meniace kontext tak zahŕňa volanie vytvorených proxy funkcií. Na obrázku 6.2 je znázornený postup invokovania proxy funkcií pri volaní funkcie enklávy [2].

Pred vykonaním interoperabilitových transakcií v enkláve je potrebné vykonať dodatočné overenia, prípadne vytvorenia prístupových lístkov. Enkláva sleduje, či transakcia invokuje IOMC kontrakt pomocou cieľovej adresy. Ak nastane daný jav, podľa invokovanej funkcie vykoná potrebné kroky. Výsledná *proof-of-concept* implementácia získava dodatočné a zároveň nepodpísané argumenty takejto transakcie a predpripravuje miesto na ich potrebné overenie. Následne sú dodatočné argumenty z transakcie odobraté a ďalej sa do EVM dostanú len tie potrebné.

6.3 IMSC kontrakt

Smart kontrakt spravujúci identity inštancií je implementovaný v jazyku Solidity s použitím kompilátora vo verzii *0.5.16* (jedná sa o rovnakú verziu kompilátora akú používa IPSC kontrakt) [32]. Pridávanie nových inštancií musí byť odsúhlasené nadpolovičnou väčšinou. Z tohto dôvodu je nutné počiatočné nasadenie kontraktu do Ethereum blockchainu s adresami troch inštancií, resp. ich adresami IPSC.

Operátor inštancie N podáva podnet na registráciu pomocou transakcie `joinRequest()`, pri ktorom sa emituje udalosť. Tá je spozorovaná operátormi ostatných inštancií, ktorý po vzdialenej atestácií enklávy $\mathbb{O}_N$ odsúhlasia jej registráciu v IMSC. Akonáhle získa súhlas

[3]Súčasť vývojového nástroja OpenEnclave SDK.

Algoritmus 3: Program IMSC kontraktu

```
▷ Deklarácia typov a premenných:
  struct InstanceInfo {
      address operator,                                  ▷ Adresa operátora.
      bool joined,                                       ▷ Stav prijatia.
      uint256 approvedCount,                             ▷ Počet schválení pri registrácii.
      map(address ⇒ bool) approved},                     ▷ Operátori, ktorí schválili žiadosť o registráciu.
  map(address ⇒ InstanceInfo) instances
  uint256 instancesCount
▷ Deklarácia funkcií:
function constructor(ipscA, operatorA, ipscB, operatorB, ipscC, operatorC) public
    instances[ipscA] ← InstanceInfo(operatorA, true, 0);
    instances[ipscB] ← InstanceInfo(operatorB, true, 0);
    instances[ipscC] ← InstanceInfo(operatorC, true, 0);
    instancesCount ← 3;

function joinRequest(ipsc) public
    assert instances[ipsc].exists() = false;           ▷ Inštancia ešte nepodala žiadosť.
    instances[ipsc] ← InstanceInfo(msg.sender, false, 0);   ▷ Vytvorenie záznamu.
    emit joinRequested(ipsc);

function approveRequest(myIpsc, approvingIpsc) public
    assert instances[myIpsc].operator = msg.sender;    ▷ Kontrola odosielateľa.
    assert instances[myIpsc].joined = true;            ▷ Odosielateľ je registrovaný.
    assert instances[approvingIpsc].operator ≠ address(0);   ▷ Žiadateľ existuje.
    r ← instances[approvingIpsc];
    r.approved[msg.sender] ← true;                     ▷ Odosielateľ potvrdzuje žiadosť.
    r.approvedCount + +;
    if r.joined = false ∧ r.approvedCount > instancesCount/2 then
        r.joined ← true;          ▷ Je považovaný za registrovaný, ak je schválený väčšinou.
        instancesCount + +;
        emit joined(approvingIpsc);

function isJoined(ipsc) public, view                   ▷ Overenie registrácie.
    return instances[ipsc].operator ≠ address(0) ∧ instances[ipsc].joined = true;
```

nadpolovičnej väčšiny aktuálne registrovaných inštancií, je považovaná za registrovanú. Overenie registrácie prebieha volaním funkcie `isJoined()` (viď. algoritmus 3).

6.4 Proof-of-concept

Keďže sa jedná o *proof-of-concept* implementáciu, nie všetky prvky sú riadne implementované. Pre zamýšľané fungovanie protokolu z návrhu 5.2.1 je nutné doimplementovať nasledujúce časti:

Z pôvodného návrhu Aquarea [32]:

- generovanie potvrdení o vykonaní transakcie,
- generovanie a overovanie dôkazov π_{hdr}^{mem}, π_{rcp}^{mk}, π^{inc}
- zabezpečenie komunikácie medzi klientom a operátorom (napr. pomocou OpenSSL)

Z rozšíreného návrhu o interoperabilitový protokol 5.2.1:

- implementácia Ethereum light klienta v enkláve spoločne s overovaním dôkazov
- generovanie a používanie prístupových lístkov externých klientov pre IPSC

6.5 Testovanie

Vývoj kontraktov z počiatku prebiehal pomocou webového vývojového prostredia *Remix*,[4] ktorý umožňuje jednoduchý vývoj, kompiláciu, ladenie a nasadenie do testovacej blockchain siete priamo vo webovom prehliadači. Avšak v tomto prostredí je pre overenie správnej funkcionality nutné vykonávať testy manuálne. Z toho dôvodu bol neskôr vo vývoji IOMC a IMSC kontraktov používaný framework *Truffle*,[5] ktorý v kombinácií s *Ganache*[6] vytvorí ideálne lokálne testovacie prostredie.

V rámci testovania kontraktov bola taktiež zisťovaná ich výpočtová náročnosť pri vykonávaní jednotlivých funkcií. Zistené hodnoty pomocou Remix prostredia sú v jednotkách gasu zobrazené v tabuľkách 6.1, 6.2 a 6.3. Je nutné poznamenať, že IOMC kontrakty sú vykonávané v privátnom blockchaine, kde cena gasu je minimálna, resp. zanedbateľná oproti verejným blockchainom.

Testovanie vzájomnej komunikácie implementovaných častí spoločne s overovaním správnosti postupu pri vykonávaní navrhnutého interoperabilitového protokolu bolo vykonávané pomocou nástroja *Pexpect*[7] v jazyku Python. Nástroj umožňuje súbežné spustenie a ovládanie viacerých programov (v tomto prípade inštancie Aquarea a programy klientov), ako aj kontrolovanie očakávaných výstupov.

Funckia	konštruktor	sendInitialize	sendCommit	sendRevert
Trovy transakcie	901 509	160 698	64 629	60 923
Trovy exekúcie	653 689	134 498	42 717	39 523

Tabuľka 6.1: Náročnosť vykonania funkcií odosielajúceho IOMC kontraktu v jednotkách gasu (privátny blockchain Aquarea).

Funckia	konštruktor	receiveInitialize	receiveClaim	fund
Trovy transakcie	716 330	139 218	61 245	23 168
Trovy exekúcie	509 366	112 762	39 653	1 896

Tabuľka 6.2: Náročnosť vykonania funkcií prijímajúceho IOMC kontraktu v jednotkách gasu (privátny blockchain Aquarea).

Funckia	konštruktor	joinRequest	approveRequest	isJoined
Trovy transakcie	830 074	48 629	69 642	0
Trovy exekúcie	567 838	25 949	45 554	0

Tabuľka 6.3: Náročnosť vykonania funkcií IMSC kontraktu v jednotkách gasu (verejný blockchain Ethereum).

[4] https://remix.ethereum.org/
[5] https://github.com/trufflesuite/truffle
[6] https://github.com/trufflesuite/ganache-cli
[7] https://github.com/pexpect/pexpect

Kapitola 7

Bezpečnostná analýza protokolu

Bezprostredne pri návrhu protokolu bola vynaložená značná časť úsilia do jeho bezpečnosti, a to najmä v zmysle zamedzenia nežiadúceho vzniku alebo zániku krypto tokenov. V tejto kapitole sú rozpracované rôzne pozície útočníka, ktorého cieľom je profitovať na danom prevode.

7.1 Zlomyseľný odosielateľ

Cieľom zlomyseľného odosielateľa je odoslať prostriedky príjemcovi tak, aby boli úspešne vyzdvihnuté, avšak odosielateľ o ne neprišiel. Odosielateľ by mohol pri odosielaní transakcie príjemcovi pozmeniť hodnotu posielaných tokenov. V takomto prípade by príjemca zistil, že hash zasielanej transakcie sa nezhoduje s hash hodnotu transakcie v potvrdení (v prípade, ak by útočník zmenil aj to, nezhodoval by sa podpis, ktorý nedokáže sfalšovať). Pozmenenie dôkazov taktiež vedie k následnému odhaleniu príjemcom. V dôsledku čoho, tak útočník v prvej fáze nie je schopný napadnúť navrhovaný protokol.

Ďalšia možnosťou útoku je tretia fáza. Avšak po vykonaní transakcie `sendCommit()` enklávou, odosielateľ natrvalo prichádza o odosielané tokeny. Ak by sa odosielateľ rozhodol nevykonať túto transakciu, následné sfalšovanie potvrdení a dôkazov je prakticky nemožné, čím by sa protokol v príjemcovej enkláve zastavil a príjemca by tak nezískal nič.

7.2 Zlomyseľný príjemca

Cieľom zlomyseľného príjemcu je pripísanie si tokenov bez toho, aby bol vytvorený prevod alebo využije prevod, ktorého nie je príjemca, resp. uplatní validný prevod viac krát, či prípadne nastaví vyššiu prijímajúcu čiastku.

Príjemca môže vykonať transakciu `receiveInit()` bez toho, aby dostal informácie o zahájení prevodu od príjemcu. Ak by sa v následnom kroku snažil tieto prostriedky získať, potreboval by zaslať enkláve dôkazy o vykonaní odosielajúcej transakcie `sendCommit()` na svoj účet. Takéto validné dôkazy môžu byť vytvorené len v enkláve jednej z inštancií. Ak existujú, enkláva príjemcu môže byť ubezpečená, že v inej inštancii sa tokeny odčítali, v dôsledku čoho môžu byť príjemcovi tokeny pričítané. Vyrobenie dôkazov o takej transakcii, príjemca nie je schopný vytvoriť. Avšak môže použiť už vytvorené dôkazy z inej transakcie. V takomto prípade by sa ešte pred overením dôkazov zistilo, že príjemca prevodu sa nezhoduje. Toto overenia vykonáva enkláva na základe parametrov vytvorenej udalosti v prijatom recepte. Ak by sa príjemca rozhodol znovu použiť správne prijatý prevod, overe-

nie by zlyhalo na rovnakom mieste, keďže jedným z parametrov udalosti je aj identifikátor prevodu príjemcu, ktorý by sa líšil od toho pôvodného. Zmena parametrov udalosti vedie k neúspešnému overeniu podpisu enklávy.

V prípade, ak by príjemca v druhej fáze protokolu zadal vyšší počet prijímaných tokenov, v následujúcej tretej fáze by odosielateľová enkláva nedokončila prevod. Ak by príjemca neposkytol opravenú transakciu s patričnými dôkazmi, odosielateľ by si po určitej dobe mohol vyzdvihnúť odosielané prostriedky, v dôsledku čoho by prevod zrušil.

7.3 Zlomyseľný operátor

Pri vykonávaní protokolu je možné, že operátor začne nespolupracovať resp. cenzuruje požiadavky alebo vykonáva inú zlomyseľnú činnosť. Návrh s takouto situáciou počíta a pri nespolupráci vykonávania transakcií (alebo dotazov) je klientovi umožnené túto skutočnosť ohlásiť a požiadať tak o jej rezolúciu pomocou IPSC kontraktu. Nevyriešené požiadavky sú verejne dostupné a znižujú operátorovi reputáciu. Keďže externý užívateľ taktiež pristupuje k operátorovi, je nevyhnuté, aby vedel podávať žiadosti o rezolúciu do IPSC kontraktu. Pre podávanie je nutné disponovať platným prístupovým lístkom, ktorý vydáva enkláva. Systém prideľovania lístkov je vysvetlený v prvej fáze protokolu v podsekcii 5.2.1. V prípade, ak zúčastnené strany prevodu neprijali daný lístok, nepokračujú ďalej v protokole.

Podľa teorému 6.1 a 6.2 v publikácií Aquareum [32], operátor nemôže modifikovať stav účtovej knihy bez rešpektovania sémantiky virtuálneho stroja enklávy a ani modifikovať uložené záznamy v účtovej knihe. Avšak zlomyseľný operátor môže na istý čas poskytnúť dvom rôznym klientom dva rozličné stavy $\mathbb{L}$, čím zavádza do systému nejednoznačnosť. Až uverejnením aktualizovaného koreňa $\mathbb{L}$ do verejného blockchainu dochádza k jednoznačnosti a klient tak dokáže overiť, či stav poskytnutý operátorom skutočne predchádzal aktuálnemu stavu. Čas finálnosti tak závisí na intervale zverejňovania koreňa (definovaného operátorom) a času finálnosti zvoleného verejného blockchainu.

Interoperabilový protokol spolieha na dopĺňanie krypto tokenov do prijímajúceho IOMC kontraktu operátorom. V prípade, ak by sa protokol pozastavil v dôsledku nedostatočného stavu účtu, príjemca by verejným spôsobom vyzval operátora o jeho doplnenie, podobne ako pri rezolúcií cenzúry.

7.4 Kompromitované TEE

V doterajších úvahách bolo považované prostredie TEE za neprelomiteľné, a teda bezpečné. Avšak, najnovšie publikácie [45] poukazujú, že aj zvolená technológia Intel SGX môže byť zraniteľná. Výmena za inú technológiu môže byť dočasným riešením, avšak je pravdepodobné, že časom môže byť taktiež prelomená. Potenciálnym riešením je súčasné využitie viacerých technológií, ktorých výsledky sú navzájom porovnávané. Nakoľko sa nepredpokladá, že väčšina použitých technológií bude obsahovať rovnakú chybu, v dôsledku čoho, by bol útok na konkrétnu technológiu ľahko identifikovateľný.

Kapitola 8

Záver

Cieľom práce bolo, v provom rade, dôkladne sa oboznámiť s princípmi technológie blockchain a trusted computing, hlavne so zameraním sa na Intel SGX. Následne boli preskúmané prístupy kombinujúce dané technológie, pričom najväčšie úsilie bolo vynaložené na preštudovanie fungovania projektu Aquareum. Z nadobudnutých vedomostí bol následne vytvorený návrh interoperabilitového protokolu Aquarea, ktorého cieľom je zabezpečiť externú interoperabilitu medzi viacerými jeho inštanciami.

Výsledkom tejto práce je okrem samotného návrhu protokolu aj jeho *proof-of-concept* implementácia. Tá, okrem navrhnutých nových súčastí, ako interoperabilitové mikro kontrakty a smart kontrakt na správu identít podporovaných inštancií, zahŕňa aj čiastočnú implementáciu navrhovaných prvkov systému z pôvodného návrhu. Jedná sa najmä o klientsky program, klient-server komunikáciu s inštanciou Aquarea a vlákno dispečera riadiaceho všetky vykonávané transakcie. Výsledná *proof-of-concept* implementácia je dôkladne otestovaná a zároveň nad samotným návrhom bola vykonaná bezpečnostná analýza.

Jedným z najambicióznejších využití platformy Aquareum je jeho adaptácia existujúcimi centrálnymi bankami, ktoré vďaka nemu poskytujú klientom digitálnu menu. Keďže centrálnych bánk exituje viacero, je zrejmé, že zároveň tak bude súčasne existovať viacero nezávislých inštancií Aquarea. Legitímnou požiadavkou teda je, aby tieto banky vedeli pri externých prenosoch meny navzájom spolupracovať. Prínos tejto práce možno práve preto predpokladať v uplatnení hlavne v medzibankových platbách jednotlivých klientov.

Navrhnutý protokol je možné ďalej rozšíriť o externé vyžiadanie krypto tokenov, ktoré by pozostávalo z vynechania prvej fázy. Aktuálna implementácia vychádza z predpokladu o krypto tokenoch rovnakej hodnoty. Námetom na rozšírenie tak môže byť podpora prenosu tokenov v rozličnej hodnote. Ďalšie rozšírenie práce vedie k navrhnutiu a implementácii protokolu invokovania externých mikro kontraktov, pričom pri ich vykonávaní môže byť upravený stav externej účtovej knihy alebo je možné upraviť stav internej aj externej účtovej knihy súčasne.

Literatúra

[1] *Bitcoin – Developer Guide* [online]. Bitcoin Project. [navštívené 13.12.2020]. Dostupné z: https://developer.bitcoin.org/devguide/index.html.

[2] *SGX 101* [online]. USA, Atlanta: Georgia Institute of Technology – Systems Software & Security Lab. Revidované 13.07.2019. [navštívené 22.11.2020]. Dostupné z: https://github.com/sslab-gatech/sgx101-gitbook/tree/master/sgx-bootstrap.

[3] TEE System Architecture. *GlobalPlatform technical overview*. verzia 1.2. November 2018.

[4] *Trusted Platform Module Library: Part 1: Architecture*. Level 00 Revision 01.59. TCG, november 2019.

[5] *Intel 64 and IA-32 Architectures Software Developer's Manual: Volume: 3D: System Programming Guide, Part 4*. 332831-073US. Intel Corporation, november 2020.

[6] 5200.28 STD, D. *Trusted Computer System Evaluation Criteria*. Dod Computer Security Center, december 1985.

[7] Alves, T. a Felton, D. Trustzone: Integrated Hardware and Software Security. Január 2004.

[8] Anati, I., Gueron, S., Johnson, S. a Scarlata, V. Innovative technology for CPU based attestation and sealing. In: ACM New York, NY, USA. *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*. 2013, sv. 13, s. 7.

[9] Bao, Z., Wang, Q., Shi, W., Wang, L., Lei, H. et al. When Blockchain Meets SGX: An Overview, Challenges, and Open Issues. *IEEE Access*. September 2020, zv. 8, s. 170404–170420. DOI: 10.1109/ACCESS.2020.3024254.

[10] Bentov, I., Ji, Y., Zhang, F., Breidenbach, L., Daian, P. et al. Tesseract: Real-Time Cryptocurrency Exchange Using Trusted Hardware. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, November 2019, s. 1521–1538. CCS '19. DOI: 10.1145/3319535.3363221. ISBN 9781450367479. Dostupné z: https://doi.org/10.1145/3319535.3363221.

[11] Bistarelli, S., Mercanti, I. a Santini, F. An Analysis of Non-standard Bitcoin Transactions. In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2018, s. 93–96. DOI: 10.1109/CVCBT.2018.00016.

[12] Buterin, V. et al. A next-generation smart contract and decentralized application platform. *White paper*. 2014, zv. 3, č. 37.

[13] Chen, H., Pendleton, M., Njilla, L. a Xu, S. A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses. *ACM Comput. Surv.* New York, NY, USA: Association for Computing Machinery. jún 2020, zv. 53, č. 3. DOI: 10.1145/3391195. ISSN 0360-0300. Dostupné z: https://doi.org/10.1145/3391195.

[14] Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N. et al. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution. *CoRR*. 2018, abs/1804.05141. Dostupné z: http://arxiv.org/abs/1804.05141.

[15] Copigneaux, B. et al. *Blockchain now and tomorrow: Assessing multidimensional impacts of distributed ledger technologies*. Luxembourg, European Union: Joint Research Centre, júl 2019.

[16] Copigneaux, B. et al. *Blockchain for supply chains and international trade: Report on key features, impacts and policy options*. Brussels, European Union: European Parliamentary Research Service, máj 2020.

[17] Costan, V. a Devadas, S. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* Massachusetts Institute of Technology. 2016, zv. 2016, č. 86, s. 1–118.

[18] Costan, V., Lebedev, I. a Devadas, S. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, August 2016, s. 857–874. ISBN 978-1-931971-32-4. Dostupné z: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan.

[19] Crosby, S. A. a Wallach, D. S. Efficient Data Structures for Tamper-Evident Logging. In: *Proceedings of the 18th Conference on USENIX Security Symposium*. USA: USENIX Association, 2009, s. 317–334. SSYM'09.

[20] Dannen, C. *Introducing Ethereum and Solidity*. Január 2017. ISBN 978-1-4842-2534-9.

[21] Das, P., Eckey, L., Frassetto, T., Gens, D., Hostáková, K. et al. FastKitten: Practical Smart Contracts on Bitcoin. In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, August 2019, s. 801–818. ISBN 978-1-939133-06-9. Dostupné z: https://www.usenix.org/conference/usenixsecurity19/presentation/das.

[22] Dessouky, G., Sadeghi, A.-R. a Stapf, E. Enclave Computing on RISC-V: A Brighter Future for Security? In: *1st International Workshop on Secure RISC-V Architecture Design Exploration (SECRISC-V), co-located with ISPASS-2020*. Apríl 2020.

[23] Dhillon, V., Metcalf, D. a Hooper, M. *Blockchain Enabled Applications: Understand the Blockchain Ecosystem and How to Make It Work for You*. 1st. USA: Apress, 2017. ISBN 1484230809.

[24] Frankenfield, J. *Cryptocurrency* [online]. Michael Sonnenshein. Investopedia. 05.05.2020. [navštívené 9.12.2020]. Dostupné z: https://www.investopedia.com/terms/c/cryptocurrency.asp.

[25] Ghimire, S. a Selvaraj, H. A Survey on Bitcoin Cryptocurrency and its Mining. In: *2018 26th International Conference on Systems Engineering (ICSEng)*. 2018. DOI: 10.1109/ICSENG.2018.8638208.

[26] Gjerdrum, A., Pettersen, R., Johansen, H. a Johansen, D. Performance of Trusted Computing in Cloud Infrastructures with Intel SGX. In:. Január 2017, s. 696–703. DOI: 10.5220/0006373706960703.

[27] Gueron, S. *A Memory Encryption Engine Suitable for General Purpose Processors*. Israel: [b.n.], február 2016.

[28] Haber, S. a Stornetta, W. S. How to Time-Stamp a Digital Document. *J. Cryptol.* Berlin, Heidelberg: Springer-Verlag. Január 1991, zv. 3, č. 2, s. 99–111. DOI: 10.1007/BF00196791. ISSN 0933-2790. Dostupné z: https://doi.org/10.1007/BF00196791.

[29] Han, R., Lin, H. a Yu, J. On the optionality and fairness of Atomic Swaps. In:. Október 2019. DOI: 10.1145/3318041.3355460. ISBN 978-1-4503-6732-5.

[30] Hardjono, T. a Smith, N. Decentralized trusted computing base for blockchain infrastructure security. *Frontiers in Blockchain*. MIT Connection Science & Engineering, Cambridge, MA, United States: Frontiers. 2019, zv. 2, s. 24.

[31] Hjálmarsson, F., Hreiðarsson, G. K., Hamdaqa, M. a Hjálmtýsson, G. Blockchain-Based E-Voting System. In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. Júl 2018, s. 983–986. DOI: 10.1109/CLOUD.2018.00151.

[32] Homoliak, I. a Szalachowski, P. *Aquareum: A Centralized Ledger Enhanced with Blockchain and Trusted Computing*. Máj 2020.

[33] Hua, Z., Gu, J., Xia, Y., Chen, H., Zang, B. et al. VTZ: Virtualizing ARM TrustZone. In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, August 2017, s. 541–556. ISBN 978-1-931971-40-9. Dostupné z: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/hua.

[34] Huanguo, Z., Jie, L., Gang, J., Zhiqiang, Z., Fajiang, Y. et al. Development of trusted computing research. *Wuhan University Journal of Natural Sciences*. Springer. 2006, zv. 11, č. 6, s. 1407–1413.

[35] Kshetri, N. Blockchain and the Economics of Food Safety. *IT Professional*. Máj 2019, zv. 21, č. 3, s. 63–66. DOI: 10.1109/MITP.2019.2906761.

[36] Lee, D., Kohlbrenner, D., Shinde, S., Asanović, K. a Song, D. Keystone: An Open Framework for Architecting Trusted Execution Environments. In: *Proceedings of the Fifteenth European Conference on Computer Systems*. New York, NY, USA: Association for Computing Machinery, 2020. EuroSys '20. DOI:

10.1145/3342195.3387532. ISBN 9781450368827. Dostupné z: https://doi.org/10.1145/3342195.3387532.

[37] Lee, S. a Kim, S. *Short Selling Attack: A Self-Destructive But Profitable 51% Attack On PoS Blockchains* [Cryptology ePrint Archive, Report 2020/019]. 2020. https://eprint.iacr.org/2020/019.

[38] Lee Thorp, A. *Attestation in Trusted Computing: Challenges and Potential Solutions*. 2010.

[39] Lind, J., Naor, O., Eyal, I., Kelbert, F., Sirer, E. G. et al. Teechain: A Secure Payment Network with Asynchronous Blockchain Access. In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. New York, NY, USA: Association for Computing Machinery, Október 2019, s. 63–79. SOSP '19. DOI: 10.1145/3341301.3359627. ISBN 9781450368735. Dostupné z: https://doi.org/10.1145/3341301.3359627.

[40] Maene, P., Götzfried, J., de Clercq, R., Müller, T., Freiling, F. et al. Hardware-Based Trusted Computing Architectures for Isolation and Attestation. *IEEE Transactions on Computers*. 2018, zv. 67, č. 3, s. 361–374. DOI: 10.1109/TC.2017.2647955.

[41] McGhin, T., Choo, K.-K. R., Liu, C. Z. a He, D. Blockchain in healthcare applications: Research challenges and opportunities. *Journal of Network and Computer Applications*. 2019, zv. 135, s. 62 – 75. DOI: https://doi.org/10.1016/j.jnca.2019.02.027. ISSN 1084-8045. Dostupné z: http://www.sciencedirect.com/science/article/pii/S1084804519300864.

[42] McKeen, F., Alexandrovich, I., Anati, I., Caspi, D., Johnson, S. et al. Intel® Software Guard Extensions (Intel® SGX) Support for Dynamic Memory Management Inside an Enclave. In: *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. New York, NY, USA: Association for Computing Machinery, 2016. HASP 2016. DOI: 10.1145/2948618.2954331. ISBN 9781450347693. Dostupné z: https://doi.org/10.1145/2948618.2954331.

[43] Mitchell, C. a Electrical Engineers, I. of. *Trusted Computing*. Institution of Engineering and Technology, 2005. Computing and Networks. ISBN 9780863415258. Dostupné z: https://books.google.sk/books?id=-298zQJnJSwC.

[44] Morbitzer, M. Scanclave: Verifying Application Runtime Integrity in Untrusted Environments. In: *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. 2019, s. 198–203. DOI: 10.1109/WETICE.2019.00050.

[45] Nilsson, A., Bideh, P. N. a Brorsson, J. *A Survey of Published Attacks on Intel SGX*. 2020.

[46] Pinto, S. a Santos, N. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* New York, NY, USA: Association for Computing Machinery. Január 2019, zv. 51, č. 6. DOI: 10.1145/3291047. ISSN 0360-0300. Dostupné z: https://doi.org/10.1145/3291047.

[47] Prashant, B., Makrant, I. a Mansi, M. *Migration from POW to POS for Ethereum.* engrXiv, Oct 2019. DOI: 10.31224/osf.io/ad8en.

[48] Sabt, M., Achemlal, M. a Bouabdallah, A. Trusted Execution Environment: What It is, and What It is Not. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. 2015, sv. 1, s. 57–64. DOI: 10.1109/Trustcom.2015.357. ISBN 978-1-4673-7952-6.

[49] Shen, C., Zhang, H., Wang, H., Wang, J., Zhao, B. et al. Research on trusted computing and its development. *Science China Information Sciences.* Beijing and Wuhan, China: Springer. 2010, zv. 53, č. 3, s. 405–433.

[50] Shen, Y., Tian, H., Chen, Y., Chen, K., Wang, R. et al. Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX. In:. New York, NY, USA: Association for Computing Machinery, 2020, s. 955–970. ASPLOS '20. DOI: 10.1145/3373376.3378469. ISBN 9781450371025. Dostupné z: https://doi.org/10.1145/3373376.3378469.

[51] Yaga, D., Mell, P., Roby, N. a Scarfone, K. Blockchain Technology Overview. *CoRR.* 2019, abs/1906.11078. Dostupné z: http://arxiv.org/abs/1906.11078.

[52] Yao, J. a Zimmer, V. Trusted Execution Environment. In: *Building Secure Firmware: Armoring the Foundation of the Platform.* Berkeley, CA: Apress, 2020, s. 681–743. DOI: 10.1007/978-1-4842-6106-4_17. ISBN 978-1-4842-6106-4. Dostupné z: https://doi.org/10.1007/978-1-4842-6106-4_17.

[53] Yuan, R., Xia, Y.-B., Chen, H.-B., Zang, B.-Y. a Xie, J. ShadowEth: Private Smart Contract on Public Blockchain. *Journal of Computer Science and Technology.* Máj 2018, zv. 33, s. 542–556. DOI: 10.1007/s11390-018-1839-y.

[54] Zhang, F., Cecchetti, E., Croman, K., Juels, A. a Shi, E. Town Crier: An Authenticated Data Feed for Smart Contracts. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.* New York, NY, USA: Association for Computing Machinery, Október 2016, s. 270–282. CCS '16. DOI: 10.1145/2976749.2978326. ISBN 9781450341394. Dostupné z: https://doi.org/10.1145/2976749.2978326.

[55] Zhang, M., Zhang, Q., Zhao, S., Shi, Z. a Guan, Y. SoftME: A Software-Based Memory Protection Approach for TEE System to Resist Physical Attacks. *Security and Communication Networks.* Marec 2019, s. 1–12. DOI: 10.1155/2019/8690853.

[56] Zhang, N., Sun, K., Shands, D., Lou, W. a Hou, Y. T. TruSense: Information Leakage from TrustZone. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications.* 2018, s. 1097–1105. DOI: 10.1109/INFOCOM.2018.8486293.

[57] Zheng, Z., Xie, S., Dai, H., Chen, X. a Wang, H. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In: *2017 IEEE International Congress on Big Data (BigData Congress).* 2017, s. 557–564. DOI: 10.1109/BigDataCongress.2017.85.

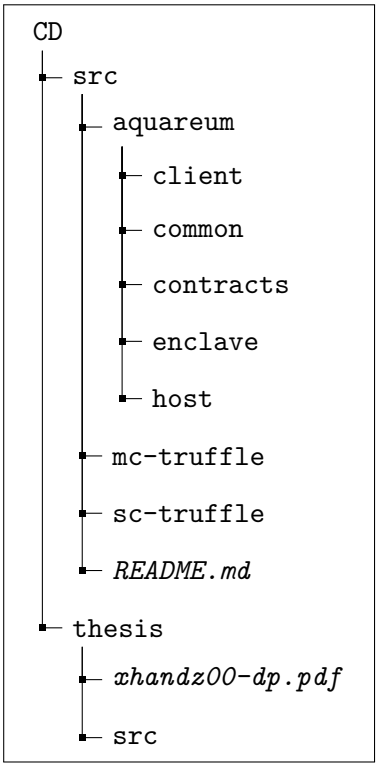
Príloha A

Obsah pamäťového média

Na priloženom pamäťovom médiu sa v adresári `src` nachádzajú zdrojové súbory implementovaného systému, ktoré sú rozdelené do následujúcich podadresárov:

- `aquareum` – obsahuje upravenú proof-of-concept implementáciu Aquarea a skladá sa z následujúcich podadresárov:
  - `client` – obsahuje zdrojové súbory klientskeho programu,
  - `common` – obsahuje implementáciu eEVM a knižnicu pre podpisovanie pomocou eliptickej krivky,
  - `contracts` – obsahuje skompilované mikro kontrakty vrátane IOMC,
  - `enclave` – obsahuje zdrojové kódy vykonávané enklávou,
  - `host` – obsahuje implementáciu operátora, dispečera a funkcionalitu serveru,
- `mc-truffle` – obsahuje zdrojové kódy interoperabilitových mikro kontraktov v jazyku Solidity spoločne s definovanými testami vo frameworku Truffle za použitia jazyka JavaScript,
- `sc-truffle` – obsahuje zdrojové kódy smart kontraktu IMSC a IPSC spoločne s testami.

Adresár `thesis` obsahuje text tejto práce vo formáte PDF a v podadresári `src` sa nachádzajú zdrojové súbory v jazyku LaTeXpre jej vygenerovanie.



Obr. A.1: Štruktúra média.