BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

ACTIVITY ALARM FOR CRYPTOCURRENCY BLOCKCHAINS

ALARM NA AKTIVITY V BLOCKCHAINECH KRYPTOMĚN

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. LUKÁŠ VOKRÁČKO**
AUTOR PRÁCE

SUPERVISOR **Ing. VLADIMÍR VESELÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2018

Zadání diplomové práce/21185/2017/xvokra00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel:	**Vokráčko Lukáš, Bc.**
Obor:	Informační systémy
Téma:	**Alarm na aktivity v blockchainech kryptoměn**
	Activity Alarm for Cryptocurrency Blockchains
Kategorie:	Počítačové sítě

Pokyny:

1. Analyzujte syntax a sémantiku adres, transakcí a bloků v blockchainech nejvýznamějších kryptoměn jako Bitcoin, Ethereum, ZCash, Monero.
2. Prostudujte si implementace oficiálních klientů pro tyto kryptoměnové sítě, zaměřte se na dostupná API a RPC.
3. Navrhněte back-end a front-end aplikaci, která uživateli umožní vytvořit si alarm na specifickou kryptoadresu v transakci.
4. Dle doporučení vedoucího implementujte aplikaci.
5. Proveďte testování na reálné situaci, diskutujte dosažené výsledky a škálovatelnost systému pro více uživatelů a potenciálně stovky sledovaných adres.

Literatura:

- Bitcoin Project, *Developer Guide - Bitcoin*, [online], dostupné na https://bitcoin.org/en/developer-guide
- Monero Project*, Developer Guides | Monero - secure, private, untraceable*, [online], dostupné na https://getmonero.org/resources/developer-guides/
- Ethereum Project, *Ethereum Development Tutorial*, [online], dostupné na https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial
- ZCash Project, *Zcash 1.0 "Sprout" Guide*, [online], dostupné na https://github.com/zcash/zcash/wiki/1.0-User-Guide

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí:	**Veselý Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání:	1. listopadu 2017
Datum odevzdání:	23. května 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstract

Cryptocurrencies are becoming popular and the demand for monitoring transactions inside them increases alongside with it. In this thesis, I describe few of the most widespread cryptocurrencies built on top of a blockchain and how to obtain information of their transactions in order to raise alarms. I discuss existing solutions and describe application Cryptoalarm designed for monitoring transactions involving specific addresses in order to raise alarms. Cryptoalarm scans blockchains of cryptocurrencies such as Bitcoin, Bitcoin Cash, Litecoin, Zcash, Dash, Ethereum and raises alarms about address activities in real-time.

Abstrakt

Popularita kryptoměn roste a s ní také poptávka po nástrojích, které umožňují monitorování transakcí. V této práci se zabývám několika nejrozšířenějšími kryptoměnami postavených na technologii blockchain a popisuji možnosti, jakými je možné detekovat výskyt specifických adres v transakcích. Zhodnotil jsem již existující nástroje a popisuji aplikaci Cryptoalarm, která je navržena pro systematické monitorování transakcí za účelem detekce zapojení specifických adres v transakcích. Cryptoalarm skenuje blockchainy kryptoměn Bitcoin, Bitcoin Cash, Litecoin, Zcah, Dash, Ethereum a umožňuje zasílání notifikací v reálném čase.

Keywords

Blockchain, cryptocurrency, alarm, monitoring, notification, transaction, Bitcoin, Bitcoin Cash, Litecoin, Dash, Zcash, Ethereum

Klíčová slova

Blockchain, kryptoměna, alarm, monitorování, notifikace, transakce, Bitcoin, Bitcoin Cash, Litecoin, Dash, Zcash, Ethereum

Reference

VOKRÁČKO, Lukáš. *Activity Alarm for Cryptocurrency Blockchains.* Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Vladimír Veselý, Ph.D.

Rozšířený abstrakt

Kryptoměny jsou distribuovaná a čistě virtuální alternativa k běžným, státem vydávaným měnám. Na rozdíl od státních měn, kryptoměny jsou založeny na síti distribuovaných uzlů, nevyžadují důvěru ve třetí strany a jsou zabezpečeny asymetrickou kryptografií. Většina kryptoměn máte také limitovaný počet mincí a neumožňuje umělou inflaci. Kryptoměny jsou imunní vůči regulacím a neumožňují zakázat transakce mezi vybranými lidmi nebo skupinami. V poslední době se kryptoměny těší zvýšené pozornosti a díky tomu se dostávají do hledáčku lidí, kteří se o kryptoměny do této chvíle nezajímali. Díky tomu se objevuje i zájem o sledování pohybů mezi určitými účty tak, jak je možné sledovat ve všech bezhotovostních platbách v bankovním systému. Blockchain (technologie, na které jsou kryptoměny postaveny) je ze své podstaty transparentní. Libovolná osoba tak může zkoumat transakce mezi jednotlivými účty. Na druhou stranu tu pak jsou technologie rozšiřují blockchain o možnost skrytí detailních informací o transakcích za použití další kryptografické vrstvy. Tato práce je zaměřena na monitorování transakcí v reálném čase mezi několika nejrozšířenějšími kryptoměnami, jmenovitě Bitcoin, Bitcoin Cash, Litecoin, Dash, Zcash, Monero a Ethereum. Během analýzy jednotlivých kryptoměn jsem zjistil, že v kryptoměně Monero není možné transakce monitorovat, jelikož detaily transakce jsou skryté pomocí kruhových podpisů a jednorázových adres pro příjem mincí. Přínosem této práce je nově vytvořená aplikace Cryptoalarm, která umožňuje systematické monitorování transakcí pro uživatelem definované adresy. Tato aplikace může najít uplatnění jak mezi normálními uživateli kryptoměn, tak i pro vládní a finanční instituce nebo orgány činné v trestním řízení. Tato aplikace je vyvinuta v rámci sady nástrojů pro forenzní analýzu kryptoměn projektu Tarzan zastřešeného výzkumnou skupinou NES@FIT zabývající se počítačovými sítěmi a vestavěnými systémy na Fakultě informatiky Vysokého učení technického v Brně. Aplikaci Cryptoalarm jsem navrhl tak, aby byla snadno rozšiřitelná o další kryptoměny. Mimo monitorovaní transakcí také umožňuje sledování převodů tokenů v systému smart kontraktů vycházejících ze specifikace ERC20 pro kryptoměnu Ethereum. Navržená aplikace umožňuje zasílaní notifikací v okamžiku výskytu monitorované adresy uvnitř transakce, a to buď jako email nebo pomocí REST API. Pro správu monitorovaných adres je vytvořena webová aplikace, ve které je možné definovat typ výskytu adresy, a to buď jako odesilatel nebo příjemce. Dalším z výstupu této práce je parser identit, který zpracovává data z internetového fóra Bitcointalk a páruje získané identity s jejich adresami. Dále vznikla komponenta pro identifikaci kryptoměn, do kterých adresa patří, a to na základe formátu adres jednotlivých kryptoměn. Pro verifikaci správné funkčnosti vyvinuté aplikace byla vytvořena testová sada pokrývají interakce s blockchainy kryptoměn a generování notifikací. Jelikož má aplikace sloužit pro monitorovaní transakcí na velkém počtu adres, Cryptoalarm byl podroben i testovaní škálovatelnosti. Navrhl jsem několik testů s různým počtem monitorovaných adres a zjišťoval, jak se mění čas zpracování transakcí. Pro testovaní transakcí jsem vytvořil blok s 2073 transakcemi (což odpovídá průměrnému počtu transakcí v posledních 10 000 blocích Bitcoinu) d průměrem 2.5 vstupními adres a 2.5 výstupních adres pro každou transakci. Z naměřených výsledků vyplívá, že Cryptoalarm je možné používat i pro monitorovaní tisíců adres bez značného dopadu na výkon. Cryptoalarm dosahuje času zpracováni v jednotkách sekund, a to i pro tisíce monitorovaných adres.

Activity Alarm for Cryptocurrency Blockchains

Declaration

Acknowledgements

I would like to thank my supervisor Mr. Ing. Vladimír Veselý, Ph.D. for his guidance and valuable suggestions.

Contents

Chapter 1

Introduction

Cryptocurrencies are distributed and pure virtual alternative to fiat currencies issued and backed by governments. Unlike any fiat currency, cryptocurrencies are based on distributed network and do not require trust among its users. They are secured by an asymmetric cryptography. Most of them have only limited supply of coins and do not allow artificial inflation. They are also immune to regulations and do not allow authorities to froze funds or to forbid transactions for specific people or groups. These properties have been with cryptocurrencies from the beginning. The first cryptocurrency was Bitcoin introduced in 2008, but the whole field is starting to gain more traction only recently. Even among people outside of the typical audience. So far, most users of cryptocurrencies have been people interested in the underlying technology or people seeking refuge from the old-fashioned banking system or even the governments for various reasons. Some of these reasons could be classified as permitted by law (i.e. privacy) or outside of it (i.e. payments for prohibited substances or money laundering). Demand for tracking transactions grows alongside with cryptocurrencies' audience as it is standard feature available to all cash-less transactions in fiat banking. And blockchain, the technology behind the majority of cryptocurrencies, is by its nature transparent. Everyone can see details of every transaction and thus allows such tracking. On the other hand, source code of cryptocurrencies is usually published as an open-source. People with technical and IT background are able to quickly create their own cryptocurrencies based on open-source Bitcoin/blockchain codebase. And some of those new cryptocurrencies aim to provide additional privacy over Bitcoin, where the possibility to track transactions, its senders and recipients is not possible. This is done by obscuring transaction data by using another layer of cryptography which does not violate any of the above-mentioned properties of cryptocurrency.

This thesis's focus is placed on real-time transaction monitoring on few of the most popular cryptocurrencies, specifically Bitcoin, Bitcoin Cash, Litecoin, Dash, Zcash, Monero and Ethereum. The results of this work are an application *Cryptoalarm* for real-time transaction monitoring and web application providing user interface to manage address watchlists. This application can be useful for tracking movements of illegal funds for governments, financial institutions or law enforcement agencies.

In the chapter Analysis 2, I discuss each cryptocurrency from a standpoint of monitoring transactions and describe how one can obtain such information from cryptocurrencies' underlying network. In chapter 3, I describe the design of newly created application *Cryptoalarm* providing means to perform this monitoring in systematic manner across multiple cryptocurrencies. I described its implementations in chapter 4 and testing in chapter 5. I discuss performance of *Cryptoalarm* while monitoring large number of addresses in chapter 5.3.

Cryptoalarm was developed as part of a project Tarzan [4], a set of tools for forensic analysis of cryptocurrencies. Project Tarzan is developed by NES@FIT, networked and embedded systems research group at the Faculty of Information technology of Brno University of Technology.

Chapter 2

Analysis

In this chapter, I describe what cryptocurrency is and its underlying technology - blockchain. Next, I describe each cryptocurrency in more detail in regards of transaction processing. Cryptocurrencies discussed in this chapter are Bitcoin, Bitcoin Cash, Litecoin, Dash, Zcash, Monero and Ethereum.

2.1 Cryptocurrency

Cryptocurrency is a general term used to describe digital currency that is secured by asymmetric cryptography. It is relatively new field started by the work of Satoshi Nakamoto (pseudonym), who published a paper about the first cryptocurrency called Bitcoin [18]. Bitcoin is the cryptocurrency that the most of cryptocurrencies are built on top of (and some of them discussed in this thesis).

Cryptocurrency is based on a network of computers (called nodes) that utilizes distributed peer-to-peer topology. A distributed system is a system whose components are connected with computer network and pass messages as means of communication. Distributed architecture and comparison to other architectures is shown on figure 2.1. Peer-to-peer model refers to a network of nodes where every node is equal to all other nodes.

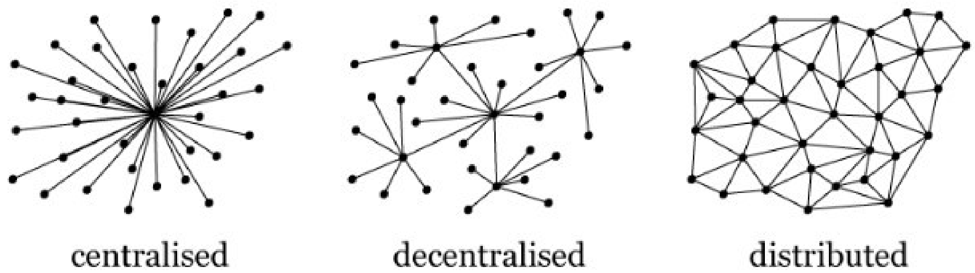


Figure 2.1: Network topology - centralized, decentralized and distributed[1]

A network of equal nodes located across computer network is the result of a combination of these two models. This network does not have designated authority (as nodes are equal)

[1]Gosselin, T.: *The Blockchain: An Infrastructure for the Commons.* [Online; visited 12.3.2018]. Retrieved from: https://www.namaste.org/blog/the-blockchain-an-infrastructure-for-the-commons

or single point of failure. Every decision in the cryptocurrency's underlying network is made by a network consensus. The network consensus is a process of reaching agreement between the majority of network nodes. This is a different approach to currency governance. In current banking systems, the currency is completely under the control of a central authority (an issuing state). This centralized model has a few flaws and cryptocurrencies are trying to address some of them. State's control of the currency supply is one of the main flaws of centralized banking. By controlling the supply of currency, the state can drastically lower the real value of the currency by emitting new money. This decision is completely under the control of the government. In cryptocurrencies, the coin supply (money supply) is controlled by a deterministic emission rate or even limited (hard-capped). In cryptocurrencies with a limited coin supply, new coins are generated only until reaching fixed threshold. After that, no new coins can be created inside the system. In cryptocurrencies with an artificial inflation, new coins are generated according to the emission rate which is deterministic process of rewards that are predictable and decreasing in time. Another problem with state issued currency is banks' control of transactions. The bank or authority can prohibit transactions of larger amounts or even specific payments (i.e. gambling). There is no such thing as a transaction control in cryptocurrencies. Every single transaction is treated equally regardless on its senders and recipients. Also, no one can freeze user's funds or take possession of given coins comparing to a common practice experienced by fiat currencies. Downside of it is inability to deal with misplaced transactions, lost credentials or theft. No transaction can be undone or funds recovered. There is a no way the user can recover lost credentials. The movement of stolen coins can only be tracked (in transparent cryptocurrencies) but recovery is not possible. The ability to recover stolen coins would undermine the core idea of a cryptocurrency as it would require central authority. Counterfeiting new coins is also not possible as cryptocurrencies are secured by a cryptographic riddle. An attempt to counterfeit coins would be detected and rejected by the network.

Also, cryptocurrencies are (pseudo-)anonymous[2]. This means that user's real identity is not attached to an account unless the user voluntarily associates himself with a specific account or accounts. There is an exception to this rule when the user wants to trade cryptocurrency in regulated exchanges. Some exchanges may require verification only after a certain volume is traded or when transferring a fiat currency. Also, some exchanges are completely decentralized and operate inside a cryptocurrency blockchain such as EtherDelta[3]. Downside to decentralized exchanges is usually a limited number of trading pairs and the missing ability to trade a fiat currency.

Another drawback of centralized banking system in comparison to cryptocurrency is that users have to trust a third party (a bank) to handle their transactions (unless the user uses only cash). In cryptocurrencies, the user only needs a computer with network connection to broadcast transactions. There is no need to trust anyone unless the user decides to keep his coins in centralized point (cryptocurrency exchange).

Transparency is another property of cryptocurrencies. In cryptocurrencies, every transaction is stored in blockchain (see 2.1.1) and its details are publicly available. Transparency can be seen as advantage or disadvantage. It depends on the perspective. Everyone's ability to view all transactions of every address can be seen as disadvantage for personal finance. Everyone can view user's spending habits once the identity of an address' user is revealed. And not just one specific address. Other addresses of the given user can be clustered together from multiple transactions. Clustering can be done if one transaction has multiple

[2]pseudonymity - users are identified by pseudonyms

[3]EtherDelta, `https://etherdelta.com/`

inputs. These addresses belong to the same user in the majority of cases (except CoinJoin transaction, see 2.2.3). On the other hand, transparency can be seen as an advantage in case of public funding where everyone can see how budgeted money is spent.

Another advantage of cryptocurrencies is that they are not limited by state borders and can be used worldwide. In the banking system, there are only few currencies that are accepted worldwide. And even then, the transfer across a state border is usually slow and expensive. Cryptocurrencies can be used to transfer any amount of coins to anyone, regardless of their location (physical or network). Also, transaction fees do not depend on the transferred amount. Transaction fees are usually lower compared to the banking system. But in certain circumstances, they can get very high as could be seen in Bitcoin on December 2017 [14]. Bitcoin's transaction fees skyrocketed as a large number of users wanted to transfer their coins at the same moment. In cryptocurrencies, transactions with higher fees are processed first. This is because the fees are used as a reward for the network node who processes given transaction. Details about how the network of cryptocurrency works and processes transactions is described in chapter 2.1.1.

2.1.1 Blockchain

Blockchain [18] is the technology originally developed for Bitcoin (the first cryptocurrency) by Satoshi Nakamoto. Blockchain was invented as a means to prevent Bitcoin users to spend their coins more than once (double spending) without central authority. Blockchain is a replicated database. It is used as a public ledger of transactions in cryptocurrencies. All of the cryptocurrencies described in this thesis use blockchain (there is also DAG [15]). This ledger stores transaction data in a transparent format and verifiable manner. Blockchain, as its name suggest, is a chain of blocks — backwards linked list of blocks. It allows only appending new blocks. Block becomes immutable once it a part of the blockchain. Immutability is achieved by using hash function to calculate a block hash. Then, the block hash is used to link blocks together to form a chain. Each block of the blockchain contains:

- transaction data
- hash of previous block
- nonce
- time stamp
- size
- difficulty
- and other attributes

Each block of a blockchain is identified with its hash. The block hash is computed from most but not all of its attributes. Most importantly, the hash of previous block is also included, which provides immutability of the whole chain. Simplified visualization of a blockchain and block's connection by hashes is shown on figure 2.2.

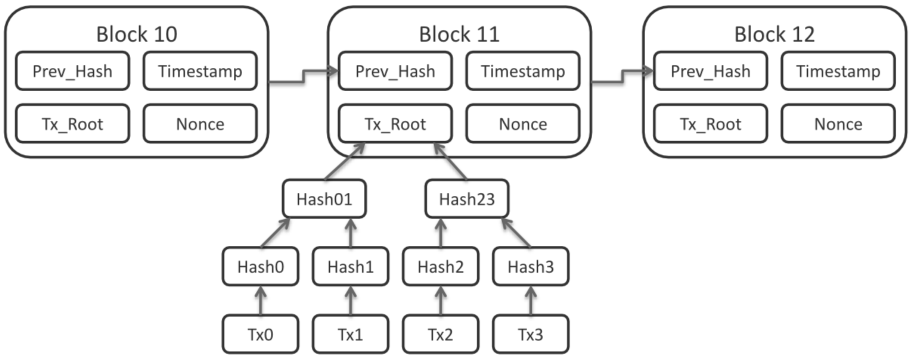


Figure 2.2: Simplified visualization of a blockchain[4]

Blockchain is managed in a distributed peer-to-peer network of computers called nodes. Nodes are essential part of any cryptocurrency network. They provide several services such as mining new blocks, verifying blocks mined by other nodes, validating transactions, storing all blockchain data and sharing those data with other nodes.

Mining is a process of creating new blocks and can be described in several steps. First, node chooses several transactions from a mempool (a pool of transactions waiting to be processed) and checks their validity. After that, the node creates a coinbase transaction. Coinbase transaction is used for miner's reward. It is a transaction with amount equal to current block reward and its recipients are chosen by the miner. Next, the node needs to solve a cryptographic puzzle. This puzzle consists of finding *nonce* that will result in a block hash according to current difficulty. Difficulty defines format for the resulting block hash which needs to be lower than given difficulty. When the resulting hash is not in the valid format, the node can alter *nonce* attribute and recalculate the block hash until hash in the valid format is produced. After producing correct block hash, the new block is broadcast to the network. Other nodes validate given block and each transaction inside it. The transaction validation is performed in two steps. First, the transaction signature is verified against the source address. Second, it is verified that address possesses the transferred amount of coins. After that, a hash of the transaction is computed. A block validation consists of checking if selected attributes of the block result in given hash. Then, the block hash is computed with the given difficulty. All blocks that are not valid (transactions, hash, rewards, difficulty) are rejected by the network and the miner does not receive any compensation. Mining new blocks is on purpose very resource expensive and thus miners are motivated to include as many transactions as they can in a single block. For each included transaction, the miner receives transaction fee which is specified by transaction's sender. Transactions with the highest fees are processed first as it generates the biggest reward. The mining difficulty is set in a way that it takes approximately *block time* of computing power of the whole network to find a nonce resulting in valid block hash. Mining also secures blockchain against modification of past transactions and blocks — immutability. The block hash would change in case any information stored in the block would be modified. And as hash is used as for linking previous blocks, it would render every newer block hash invalid. A situation where multiple valid blocks are created at the same time can occur as multiple nodes compete to create a new block. In case of several valid blocks are created at approximately the same time, the network accepts all of them.

[4]Sharma, K.: *What is Blockchain and how does it work?*. [Online; visited 12.3.2018]. Retrieved from: https://blog.etherbit.in/2017/11/22/what-is-blockchain/

This leads to multiple branches of blockchain (chain-slit). This situation is visualized on figure 2.3. Later, when another block is created, it can link to exactly one previous block. At that time, the network decides to accept only the longest chain (the chain with the most blocks) as it consumed the most processing power. The probability of such event depends on various attributes such as block size and block time. This process of securing blockchain by mining is called proof-of work. Currently, there is also another way to secure blockchain called proof-of-stake [7].

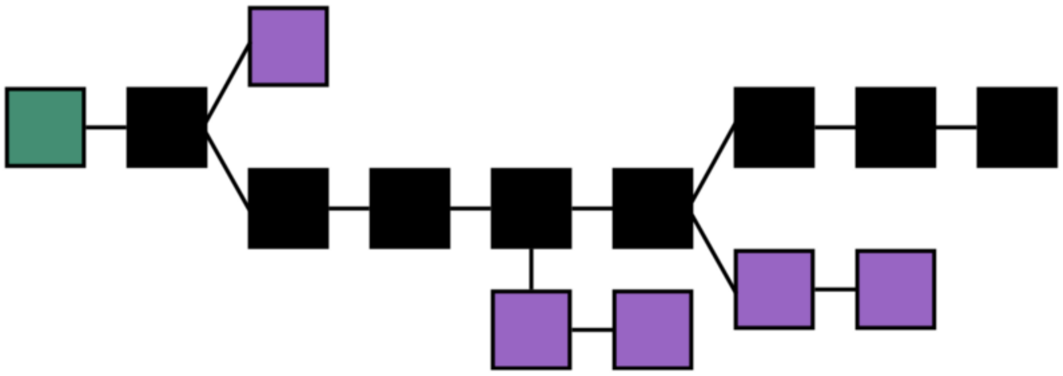


Figure 2.3: Blockchain splitting and orphaned block. Black represents the longest chain from genesis block (green) up to current block. Orphaned blocks are purple. These blocks never become part of the main chain.[5]

The number of transactions included in a single block is limited in every cryptocurrency using blockchain. This limit often depends on block time as it would not be possible to synchronize the whole cryptocurrency network with large blocks and block time in seconds. This configuration would result in chain splits (as shown on figure 2.3) happening with great frequency as miners would create new blocks with different preceding blocks.

Every information stored in blockchain is transparent. Blockchain data can be either downloaded from the internet and then processed or requested from the network nodes. Another way to obtain blockchain data is to run a network node. There are several relatively new cryptocurrencies that aim to obfuscate transaction details while keeping the ability to verify transactions. Zcash (2.6) provides optional feature to mask transaction details with zero-knowledge proofs [21]. Also, there is another protocol built on top of blockchain which improves privacy. It is called CryptoNote [23]. Monero is one of few cryptocurrencies built on top of CryptoNote and I describe it in more detail in chapter 2.7.

So far, I have talked about chain splitting as randomly occurring event. Chain splits can also be intentional. It is used for creating a new cryptocurrency from existing blockchain. This process is called forking. By forking, a subset of the cryptocurrency network starts to accept blocks with a different set of rules. This results in a new branch of blockchain. This branch continues to exist only in the nodes who adhere to the new rules. The nodes accepting the original rules will discard these blocks as invalid.

[5]Sekhon, J. S.: *The Consensus Conundrum in Blockchain Systems.* [Online; visited 12.3.2018]. Retrieved from: https://medium.com/@jna1x3/the-consensus-conundrum-in-blockchain-systems-f442eaf23160

2.1.2 Terminology

Terminology in cryptocurrencies is different from terminology of banking systems. There are terms as address, private key and wallet.

Address is used as an account number. Address is a public key of public/private key pair generated by an asymmetric cryptography.

Private key is used to sign transactions to prove the ownership of the given public key (address).

Wallet is a software for address management and network interactions. A wallet can manage multiple addresses and provides functionality needed to manipulate with user's funds. Also, a wallet can generate new addresses. A wallet needs to have an access to the whole blockchain to be able to calculate address balance. There are two types of wallets. First type is a full-node wallet — a wallet that stores the whole blockchain locally. Second type is a light wallet. A light wallet does not store blockchain data locally but requests them on demand from a network full-node.

2.2 Bitcoin

Bitcoin was the first cryptocurrency. The concept (blockchain and Bitcoin protocol) was published by Satoshi Nakamoto in *Bitcoin: A Peer-to-Peer Electronic Cash System* [18]. Bitcoin uses coins of the same name — Bitcoins. Currently, Bitcoin is the most widely known and used cryptocurrency. It dominates cryptocurrencies market with 34%[6] market share.

Bitcoin has an artificially limited supply of coins to approximately 21 million. New coins are emitted only when a new block is created. This is done by verifying a subset of pending transactions and computing block hash according to the difficulty rules as described in 2.1.1. A reward for creating new block is a certain number of Bitcoins. The miner who creates a block also includes coinbase transaction with the current reward. Coinbase transaction has only outputs that are chosen by the miner. The current reward for creating a block is 12.5 Bitcoins. This reward is being continuously halved every 210 000 blocks[7]. Next halving is estimated to happen on June 5. 2020[7]. The last block with reward is projected to be mined in the year 2140, reaching the final supply of 21 million Bitcoins. Current circulating supply is approximately 17 million Bitcoins. Transaction fees will be the only reward for miners when the block reward reaches zero. Bitcoin's mining difficulty is calculated every 2016 blocks[8] (roughly every 2 weeks). Ten minutes is the desired time that the whole network should spend to find a nonce that results in a valid block hash. This time is called a block time. The following formula is used to calculate the difficulty:

$$difficulty_y = difficulty_{n-1} \times \frac{20160}{elapsed\ time} \tag{2.1}$$

Elapsed time is the number of minutes it took to mine the last 2016 blocks.

Originally, Bitcoin was intended to run on user's computers and mine blocks on CPUs. Nowadays, Bitcoin is mined on graphic cards, which utilize massive parallelism for block's

[6]Bitcoin market share - 18.1.2018, `https://coinmarketcap.com/charts/#dominance-percentage`
[7]Bitcoin block reward, `http://www.bitcoinblockhalf.com/`
[8]Bitcoin difficulty, `hhttp://learnmeabitcoin.com/guide/difficulty`

hash calculations. Also, there are specialized hardware solutions (ASICs) designed especially to mine Bitcoin. Mining is a source of controversy as it is estimated that the whole Bitcoin network has a power consumption of 62.7TWh (April 24, 20018) [1]. This is comparable to the total annual power consumption of Czech Republic in 2015 (61TW/h) [10]. Mining on specialized hardware is usually done in mining farms which undermines the concept of decentralization as a significant part of the network is located in a one place. This is usually the place with cheap electricity.

Bitcoin blockchain stores transaction information in completely transparent way and does not provide any means to hide transaction senders, recipients or amounts. There are several tools to explore blockchain data even without running a Bitcoin node called blockchain explorers. Also, Bitcoin blockchain is a ledger of transactions and not balances. Inputs of a transaction are previous transactions, not addresses. The right to manipulate coins is transferred in transactions rather than the coins themselves. The right is transferred by specifying conditions under which another user can manipulate given coins. The knowledge of the private key corresponding to the given address is used as this condition. The user has to prove that he possesses this key in order to manipulate given coins. It is not possible to transfer the right to move only a fraction of coins for single input transaction. To send only a fraction of coins, the user needs to split them in a new transaction. This is done by separating the desired amount of coins to the payment amount and the rest. The payment amount is sent to the recipient and the rest is sent to change address. The same address as input can be used as the change address (address reuse). Address reuse is not recommended as it can reveal all payments made by the same user. Other possibility is to use a new address as the change address.

Transaction data can be acquired by running a full Bitcoin node or requested from a full node providing an RPC API. RPC call *getblock* has to be made with a block hash given as an argument to obtain information about the block. Block data structure (as shown in listing B.1) is the result of this call. Information relevant for transaction processing is stored in the attribute *tx*. RPC call *getrawtransaction* has to be made in order to get information about a specific transaction. Transaction data structure (as shown in listing B.2) is the result of this call. The attribute *vout* contains field *scriptPubKey*. Output addresses are stored in the attribute *addresses*. The attribute *addresses* contains only one address for each output of type *pubkeyhash*, which means a regular transaction. The output of type *multisig* can contain multiple addresses. This is used for scenarios where multiple addresses can manipulate with the transaction output. The attribute *vin* represents an array of inputs, each having:

- *txid* - a hash of the transaction used as an input for the current transaction
- *vout* - an index to the outputs of the input transaction where a specific address can be found

Bitcoin blockchain being a ledger of transactions and not balances, each transaction has a previous transaction as input instead of addresses. To find an address for the current transaction input, one needs to use the following formula:

As the most used cryptocurrency, Bitcoin is facing scalability issues. Bitcoin can fit only a certain number of transactions inside every block. This limit is not imposed on number of transactions but on a size of the block. Bitcoin's block weight is limited to 4 MB. I describe what a block weight is and how it correlates with its size in section 2.2.1. The number of transactions that can be included a block of 4 MB is approximately 4 000

Algorithm 1 Determine input addresses for Bitcoin transaction

```
Require: block - a block
Require: N - a transaction index in given block
  txid ← block["vin"][N]["txid"]
  index ← block["vin"][N]["vout"]
  tx ← getrawtransaction(txid)
  return tx["vout"][index]["scriptPubKey"]["addresses"]
```

(transaction size varies depending on its properties). In the end of a year 2017, Bitcoin was facing problems with transaction processing as there was greater demand for transactions than Bitcoin could handle. The maximum of 260 917[9] transactions waiting to be processed on December 22, 2017. This caused transaction fees to skyrocket as transactions with larger fees were processed first.

Bitcoin, being published as an open source, has been forked multiple times. The list of Bitcoin forks can be found on [3]. Bitcoin Cash is one of these forks and is described in chapter 2.3. Bitcoin is also used as a base from which new cryptocurrencies are derived. There are 227 cryptocurrencies based on Bitcoin (and another 346 already abandoned) according to [5] at the moment of writing this thesis.

2.2.1 Segregated witness

Segregated witness (SegWit) [16] was designed to allow more transactions to be included in a single block. Segregated witness is an optional improvement. Nodes can operate without implementing it. The maximum block size was 1 MB before the segregated witness. The block weight is a new limit defined by SegWit. The block weight is calculated by the following formula:

$$weight = base\ size \times 3 + total\ size$$

Total size is the size of a serialized block with transaction data. *Base size* is the size of a serialized block with the transaction data without any witness information. Witness information is stored in the transaction attribute *scriptSig* and consists of *signature* and *pubKey*. This information is used to prove that the sender has the right to manipulate the transaction outputs. SegWit nodes synchronize blocks with all information included. Non-SegWit nodes (legacy nodes) synchronize blocks without any witness information. This way, the legacy nodes still accept SegWit blocks as they do not violate the block size limit. The legacy nodes handle the SegWit transactions as can-be-spent-by-anyone thus, the legacy nodes accept any SegWit transactions as valid. A new address format was designed to differentiate between legacy and SegWit addresses. SegWit addresses start with `3` or `bc1`. Legacy addresses start with `1`. Examples of both address types are shown in the following list:

- SegWit
  - 3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy
  - bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq

[9] Mempool size, `https://jochen-hoenicke.de/queue/#1,all`

- Legacy: 1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2

SegWit was activated on block #481824 which was created at 10:50 AM on July 3, 2017.

2.2.2 Lighting network

Lighting network [20] is the latest improvement designed to solve Bitcoin's scalability issues. Lighting network is an improvement that implements payment channels. Payment channel is a channel that enables transactions to be made without writing any data into the blockchain. By using payment channels, users can receive payments almost immediately as they do not have to wait for the payment confirmation (transaction being included in block). To create a payment channel, two parties deposit a certain amount of coins inside this channel. Payment channel remains open until the deposited amount is exceeded or the channel is closed by one of the parties. Lighting network also allows for payments using intermediaries. Intermediary is a user who has open payment channels with both users or is a part of a link between these two parties. If the user A wants to send a payment to the user C, he can simply transfer coins using his payment channel with the user B. Then, the user B transfers those coins with a payment channel with the user C.

Lighting network is called off-chain scaling as transactions inside payment channels happen outside of the blockchain. Transactions for creation and destruction of payment channel are the only transactions that need to be stored in the blockchain.

Transactions sent through Lighting network payment channel cannot be intercepted (except by intermediary) as they are not written into the blockchain. Transaction monitoring inside a payment channel would require an open payment channel with every user. And even then, every user would have to transfer coins using this payment channel. From the perspective of this work, the transactions that happened inside payment channels cannot be monitored. The creation and destruction of payment channel are the only transactions that can be detected.

2.2.3 CoinJoin, Coinmixing

CoinJoin [12] is a technique to improve transaction ambiguity. CoinJoin utilizes Bitcoin's ability to have multiple inputs and outputs in a single transaction. Users can cooperate together and merge their intended transactions into one transaction. This way, the association between the sender and the recipient diminishes. An observer who monitors transactions cannot determine which sender has sent the coins to which recipient, unless the sum of a certain outputs equals exactly to a single input. The transaction identified by the hash *92a78def188053081187b847b267f 0bfabf28368e9a7a642780ce46a78f551ba*[10] is an example of CoinJoin transaction between three participants. It has the following inputs and outputs:

- Inputs
  - A: 0.19280926 BTC
  - B: 0.01 BTC
  - C: 0.01 BTC

[10] CoinJoin transaction, https://www.localbitcoinschain.com/tx/92a78def188053081187b847b267f0bfabf28368e9a7a642780ce46a78f551ba

- Outputs
  - W: 0.18230926 BTC
  - X: 0.01 BTC
  - Y: 0.01 BTC
  - Z: 0.01 BTC
  - Fee: 0.00103 BTC

The observer can look at the transferred amounts and try to identify who send coins to whom. For this transaction, the observer can determine who send coins to the address W as there is only one input of larger or equal denomination. Also, the output amounts cannot be the result of any other combination of inputs. This makes the link between the addresses A and W clear. Users sending the same denominations of coins can prevent linking inputs and outputs together.

Coinmixing is a different approach to solve the problem of transaction ambiguity. Instead of multiple users sending Bitcoins together in the same transaction, users send Bitcoins to a coinmixing service[11,12]. When multiple users send their coins to a coinmixing service, the service can later distribute those coins to different addresses as specified by the users. This way, the link between the sender and the recipient diminishes as coins sent by the coinmixing service can come from any user. Coinmixing service also sends transactions in randomized time delays to prevent linking those transactions together based on the time of their occurrence. Even with coinmixing, transactions can be linked together by matching their inputs and output amounts as in CoinJoin. A trusted third-party is a downside of coinmixing service.

Both of these approaches weaken the association between the input and the output addresses. This does not interfere with the ability to detect a transaction involving these addresses.

2.2.4 Bitcoin clients

Bitcoin clients can be run in two modes. A full Bitcoin client (full-node) is the combination of a wallet and a miner. Full-node stores the whole blockchain, participates in block validation and optionally can be used to mine new blocks. A light-node is a Bitcoin client that does not store the blockchain or stores only reduced information. Using a light-node is less insecure as it depends on information provided by other full-nodes. Also, a light-node does not perform block or transaction validation.

There are several implementations of a Bitcoin client. The official client is developed as an open-source by Bitcoin Core[13] and is used as the reference for the Bitcoin protocol. Alternative implementations are developed by third-parties. Those implementations are Armory[14] and Gocoin[15]. Those implementations comply with the Bitcoin Core RPC API and for purpose of this work can be used interchangeably.

[11] CoinMixer.se, `https://coinmixer.se/en/`
[12] CoinMixer.io, `https://cryptomixer.io/`
[13] Bitcoin Core, `https://bitcoincore.org/`
[14] Armory, `https://btcarmory.com/`
[15] Gocoin, `http://gocoin.pl/`

2.3 Bitcoin Cash

Bitcoin Cash is a fork of Bitcoin. Bitcoin Cash was created as the result of a disagreement in Bitcoin community regarding scaling issues. Bitcoin Cash's approach to improve scalability is to increase the block size (on-chain scaling). Bitcoin cash was created on the block #478559 that occurred on August 1, 2017.

At the moment of writing this thesis, the block size of Bitcoin Cash is set to 8 MB, which is 8× more than the original Bitcoin. Bigger block size offers more space to accommodate more transactions to each block. Faster confirmation times and lower transaction fees are the results of this modification. Bitcoin Cash's block size is not meant to be permanently 8 MB. Currently, there is a running experiment with the block size reaching up to 1 GB [22]. Increasing block size is not without drawbacks. Bigger block size favours users who run nodes on networks with more bandwidth as more data needs to be transmitted for each block. This puts regular users who run nodes in disadvantage. These users might not be able to synchronize new blocks in time. Right now, it might not a significant problem as 8 MB can propagate through the network without any obstacles. It could become a problem when the block size increases significantly. This could lead to increased centralization. With centralized network, it would become easier for authorities to gain access to nodes run in their jurisdiction and could potentially interfere with transaction processing.

The block time of Bitcoin Cash is set to ten minutes. The coin supply of Bitcoin Cash is the same as in Bitcoin, approximately 21 million. A new algorithm for difficulty adjustment was developed for Bitcoin Cash. The difficulty is adjusted every block and is calculated from the weighted block times of previous 144 blocks [17]. Other parameters such as the block reward and reward halving are kept the same as in Bitcoin.

There are several implementations of Bitcoin Cash node, namely BitcoinABC[16], Bitcoin Unlimited[17], Bitcoin XT[18], Parity Bitcoin[19] and Bitprim[20]. All of these implementations comply with the Bitcoin's RPC API definitions (see 2.2.4).

2.4 Litecoin

Litecoin was created as a clone of Bitcoin. Unlike Bitcoin Cash 2.3 which stared from Bitcoin's blockchain and share its transaction history and accounts, Litecoin started with empty blockchain. The idea behind Litecoin was to create a lightweight alternative to Bitcoin. Litecoin is compared to digital silver while Bitcoin to digital gold.

Litecoin aims to be faster than Bitcoin. Litecoin has a block time of 2.5 minutes. Shorter block time allows for more payments to be processed in the same time. In ten minutes (Bitcoin block time), four Litecoin blocks are created. Litecoin has a total supply of 84 million coins. Block reward is halved every 840 000 blocks. This results in the same interval as in Bitcoin because block time is four times shorter. The difficulty of Litecoin mining is computed by the same formula as for Bitcoin 2.1. Also, Litecoin has implemented Segregated Witness (see 2.2.1) with the block weight limit of 4 MB. To differentiate between legacy and SegWit addresses, Litecoin uses addresses in a different format. Legacy addresses

[16]BitcoinABC, `https://www.bitcoinabc.org/`
[17]Bitcoin Unlimited, `https://www.bitcoinunlimited.info/`
[18]Bitcoin XT, `https://bitcoinxt.software/`
[19]Parity Bitcoin, `https://github.com/paritytech/parity-bitcoin/`
[20]Bitprim, `https://www.bitprim.org/`

start with `L`. SegWit addresses start with `3` or `M`. Examples of both types of addressess are shown in the following list:

- SegWit
  - 3AnpJ61g2UcLhVYrrsWm4jFrXZwTtaWhV4
  - MGzxbyRdybTmVzpkxkW6tNWFrGXutBd2gr
- Legacy: LTU2cds4aSdXFip9sV4gXphnhxGQjgfjmg

Litecoin (being a copy of Bitcoin) uses the same RPC API to obtain information from underlying network as described in chapter Bitcoin 2.2. Litecoin has two implementations of a full-node. Those implementations are Litecoin Core[21] and ltcd[22].

2.5 Dash

Dash [11] is a cryptocurrency based on Bitcoin. Dash introduces several improvements over the Bitcoin protocol. These improvements are focused on faster (instant) payment confirmations and on privacy. Dash uses master nodes. Master nodes provide additional services to cryptocurrency users. Dash uses feature called InstantSend for instant payment confirmations. PrivateSend is used to improve privacy and fungibility[23] of coins. Both of these features are optional and Dash also provides regular transactions as Bitcoin does.

Dash has limited supply of coins. The final number of coins is not yet determined. The block reward depends on the difficulty of new blocks. The following formula is used to keep the reward in interval $< 5, 25 >$:

$$reward = min\left(max\left(\frac{2222222}{\left(\frac{difficulty+2600}{9}\right)^2}, 5\right), 25\right) \tag{2.2}$$

The dynamic reward is used to keep the network hash rate above 79Gh/s (giga hash per second). New miners are motivated to start mining Dash as it becomes more profitable when the block reward increases as the consequence of low difficulty. The mining difficulty is adjusted for every block in order to keep the block time around 2.5 minutes. Dash has the following structure of block rewards:

- 45% for the miner
- 45% for the master node network
- 10% is allocated for funding community projects

Dash is designed to decrease the block reward every year by 7%. It is estimated that the last block with reward will be mined around the year 2300. A total supply of coins can be only estimated as it depends on amounts allocated for community projects and a hash rate of the network. The total supply can be approximated with the following conditions:

[21] Litecoin Core, https://litecoincore.org/

[22] ltcd, https://github.com/ltcsuite/ltcd

[23] Fungibility means that coins can be used interchangeably and do not carry history

- All super blocks are fully allocated
- Hash rate never drops below 79Gh/s

With these conditions, the total supply of coins is approximately 18 million of coins. Treasury and super blocks are another differences from Bitcoin. Treasury is a monthly budget reserved for community projects. Community projects are voted for by master nodes. Community projects can be proposed by anyone and are publicly listed[24]. Super block is a block created once a month. Super block can generate up to 10% of all block rewards in given month. The amount of coins generated in super blocks depends on the allocation of treasury budget.

Dash RPC API adheres to the specification of Bitcoin API discussed in 2.2.

2.5.1 Master nodes

Dash master node is a full-node running Dash wallet and providing additional services to the network. These services are PrivateSend, InstantSend and voting for proposals. Master nodes uses a reward system. Master nodes receive 45% of the block reward. This is designed to motivate users to running master nodes as it places strain on users' bandwidth and storage. Improved decentralization is the result of more users run master nodes.

To run a master node, the user must deposit 1 000 DASH into the master node as a collateral. The collateral is never forfeit. The collateral is used to prevent a potential attacker to control more than 50% of all master nodes. The attacker would need 2 300 000 DASH (2400 master nodes running at this moment)[25] to have at least 50% of the network. He would need to purchase these coins from an open market. That would result in a spike of the price and would be financially unfeasible.

2.5.2 PrivateSend

PrivateSend is designed to improve privacy of transactions. PrivateSend is an improved version of CoinJoin. The weakness of CoinJoin is observer's ability to determine the association between senders and recipients by merging inputs of transaction until one combination results in the exact value that was send to output addresses. Dash improves CoinJoin by requiring at least three participants to join transactions with equal denominations of inputs and outputs. By using the equal denominations, an observer cannot differentiate between the senders. Users can anonymize amounts of denominations 0.1, 1, 10, 100 and 1 000 DASH. PrivateSend only hides the association between senders and recipients. It does not mask the fact that the transaction occurred on a given address.

2.5.3 InstantSend

InstantSend is used to instantly transfer coins. A user does not have to wait for the transaction being included in a block. This is achieved by master node network quorum. A transaction sent using InstantSend is broadcast to the master nodes for validation. Valid transaction is then locked within master nodes. Coins specified in this transaction cannot be spent in any other transaction. Master nodes reject all blocks with a different transaction spending the same coins.

[24]Dash community project proposals https://www.dash.org/network/#section-governance
[25]Dash master nodes, https://github.com/dashpay/dash/wiki/Whitepaper#23-trustless-quorums

2.6 Zcash

Zcash is another cryptocurrency based on Bitcoin. Zcash [13] aims to improve privacy using different approach than previously described Dash. Zcash uses zero-knowledge proofs to mask transaction amounts, senders and recipients. The process of masking transaction details is called shielding.

Zero-knowledge proof [21] refers to a cryptographic algorithm that can be used by an actor to prove possession of certain knowledge without disclosing information in question without any interaction with the verifying actor. In Dash, zero-knowledge proofs are constructed using zk-SNARKs [6]. Zero-knowledge proofs allow for the transaction verification even with shielded information and thus not violating trust-less property of the system.

There are two types of addresses used in Zcash:

- z-address
- t-address

Transactions between t-addresses (transparent addresses) are not shielded by zero-knowledge proof. Anyone is can read transaction inputs, outputs and amounts. Transactions between z-addresses are shielded and do not disclose sender, recipient or transaction amount. Only certain information is shielded in cases transaction involves both types of addresses. See table 2.1 for details.

Transaction	Sender	Recipient	Amount
$t-addr \rightarrow t-addr$	disclosed	disclosed	disclosed
$z-addr \rightarrow t-addr$	shielded	disclosed	disclosed
$t-addr \rightarrow z-addr$	disclosed	shielded	shielded
$z-addr \rightarrow z-addr$	shielded	shielded	shielded

Table 2.1: Disclosed information in Zcash transactions

Zcash total supply of coins is approximately 21 million. Block rewards follow the model of Bitcoin. Zcash also uses Founder's reward. Zcash developers will receive 20% of all mined ZEC in the first 4 years. 100% of mined coins will be rewarded to the miners after that. Founder's reward is not taken from transaction fees but only from blocks. For coinbase transactions, block reward is always sent to z-address. Zcash aims to have block time of 2.5 minutes. Block size is set to 2 MB. Difficulty is calculated from difficulty of the previous block by following formula:

$$difficulty_n = difficulty_{n-1} \times \sqrt{\frac{150}{last\ solve\ time}} \quad (2.3)$$

The official client Zcash[26] is the only implementation of Zcash node at the moment of writing this thesis. Currently, there is a proposal for fully verifying client written in Rust[27]. Zcash RPC API complies to Bitcoin RPC API. Description and examples can be found in chapter Bitcoin 2.2. Results of RPC calls to obtain transaction information differ when z-address is involved:

[26]Zcash, https://github.com/zcash/zcash

[27]Zcash rust node proposal, https://github.com/ZcashFoundation/GrantProposals-2017Q4/issues/32

- Attribute *vout* does not contain addresses of type z-address
- Attribute *vin* does not contain any information if the source transaction was sent to z-address

2.7 Monero

Monero is a cryptocurrency built on top of CryptoNote protocol [23]. CryptoNote introduces several improvements over the Bitcoin protocol and is truly anonymous. CryptoNote transactions are untraceable and unlinkable. This is achieved by combination of three approaches: one-time ring signatures, stealth addresses and ring confidential transactions.

Stealth addresses are used to hide the recipient's real address. Stealth address is a one-time address generated from a recipient's public address. By using stealth addresses, each transaction sent to the same recipient will be sent to a unique address. Stealth addresses cannot be linked back to the recipient's original address. Only sender and recipient have the knowledge of the real address.

Ring signatures are used to hide sender's address. A ring signature is a type of signature that can be created by any member of a particular group. When message is signed by ring signature, it should be unfeasible to determine which person's key was used to sign the transaction as all members of the ring are equal and valid. To form a group, user's key is combined with a number of public keys (outputs) obtained from the blockchain using a triangular distribution method. Past outputs can be used multiple times to participate in different groups over the course of time. Ring confidential transactions (RingCT) [19] is an improvement of ring signatures that allows to hide even transaction amounts.

User will obtain three keys when creating new Monero account:

- *View key* is used to view incoming transactions of the given account. *View key* can be kept private or can be shared.
- *Private spend key* is used to spend coins for the given account.
- *Public address* is used to create stealth addresses for receiving payments.

Proving a payment is as simple as looking up a payment in blockchain explorers in completely transparent cryptocurrencies. In Monero, user have to disclose the following information to prove a payment:

- transaction ID
- recipients address
- transaction key

Transaction ID is the same as in other cryptocurrencies and can be found in blockchain explorer. Transaction key is the key used to generate stealth address in combination with recipient's public key (address). User has to query his wallet to obtain a transaction key. With this information, anyone is able to query a node and see transaction details in readable manner.

Monero has an infinite supply of coins. Coin emission is defined by two emission curves. With the first main curve, approximately 18 million coins will be emitted by the end of May

2022. The main curve uses smooth emission where the block reward is decreased gradually over time until it reaches threshold of 0.6 XMR per block. Tail emission curve will be used after that. Tail emission is designed to emit exactly 0.6 XMR per block. This translates to $< 1\%$ inflation gradually decreasing over the time.

Monero aims to have block time 2 minutes. This is achieved by calculating mining difficulty every block. In order to calculate a new difficulty, block times from the last 720 blocks are taken into account while excluding 20% of outliers [24].

Monero Project[28] is the only implementation of Monero node. With above mentioned privacy features (one-time ring signature, stealth address, ring confidential transaction) it is impossible to determine transaction senders and recipients without user's compliance. Transaction monitoring in Monero is not included in developed application for this reason.

2.8 Ethereum

Ethereum [9], rather than a cryptocurrency, is a blockchain application platform with Turing-complete programming language. Ethereum is not built on top of Bitcoin protocol but uses own protocol. Besides standard features to transfer coins, Ethereum allows users to publish smart contracts. Smart contract is a fully autonomous program stored on blockchain. Smart contracts react on incoming transactions. Ethereum transactions have just single sender and single recipient. Also, sender's and recipient's address are stored directly in transactions.

Ethereum node provides an RPC API to obtain information about blocks and transactions. The following RPC calls can be performed to acquire data required for transaction monitoring:

- *eth_getBlockByNumber* is used to access information about block. Block data structure (as shown in listing B.3) is the result of this call. The important field is *transactions*. It contains a hash of every transaction included in requested block. The latest block can be obtained via this API call if string `latest` is passed as an argument.

- *eth_getTransactionByHash* is used to get information about transaction. Transaction data structure (as shown in listing B.4) is the result of this call. Important attributes from this object are:

  - *from* - sender's address
  - *to* - recipient's address

Ethereum specifies block time 12 seconds in its whitepaper [9]. However, the average block time in the network oscillates around 15 seconds[29]. The reward for mined block is constant, 3 Ether. Ethereum rewards miners whose blocks are valid but do not become part of the blockchain — orphaned blocks, *uncles* in Ethereum terminology. Reward for orphaned block is 7/8 of the block reward: 2.625 Ether[30]. The total supply of Ethereum is unlimited. Ethereum mining difficulty is recalculated every block. In order to keep maintain the block time constant, the difficulty is calculated from the last two blocks using the following formula [8]:

[28]Monero, https://getmonero.org/

[29]Ethereum average block time, https://etherscan.io/chart/blocktime

[30]Ethereum rewards, https://github.com/ethereum/wiki/wiki/Mining

$$adj\_factor = max\Big(\big(2\ if\ len(uncles_{n-1})\ else\ 1\big) - \frac{timestamp_n - timestamp_{n-1}}{9}, -99\Big) \tag{2.4}$$

$$d_n = max\Big(d_{n-1} + \frac{d_{n-1}}{2048} * adj\_factor, min\big(d_{n-1}, MIN\_DIFF\big)\Big) \tag{2.5}$$

Another difference between Ethereum and Bitcoin based cryptocurrencies is in transaction processing. Each Ethereum transaction specifies:

- Gas limit
- Gas price

Gas the term used for transaction fee. Gas price is a price the miner receives for every operation performed while executing transaction. Gas limit is a limit of how much Gas can transaction consume. Total miners reward is then calculated as $gas\ price \times gas\ used$. Gas limit is specified for each transaction as it can take significant time to execute all operations specified in smart contracts. Every modification made by transaction is revoked and transaction is not included in block if gas limit is reached before transaction is completed. In case of failed transaction, all Gas is rewarded to a miner.

There are multiple implementations of Ethereum node: cpp-ethereum[31], Go Ethereum[32], ethereumj[33] and Nethereum[34]. All of these implementations adhere to the same RPC API and can be used interchangeably.

2.8.1 Smart contracts

Smart contracts are accounts that are not controlled by private key but rather by contract's code. Once a smart contract is created, it behaves according to its specification. Smart contracts can also store information as they have internal storage. Transaction (message) has to be sent to smart contract to trigger its code. Smart contract can perform another transaction or modify its internal storage as a reaction to incoming transaction.

Smart contracts have multiple use cases. A few of them are listed in following list:

- Token systems
- Financial derivatives and Stable-Value Currencies
- Identity and Reputation Systems
- Decentralized Autonomous Organizations

[31]cpp-ethereum, https://github.com/ethereum/cpp-ethereum/
[32]Go Ethereum, https://geth.ethereum.org/
[33]ethereumj, https://github.com/ethereum/ethereumj/
[34]Nethereum, http://www.nethereum.com/

ERC20

ERC20 [2] is a standardized smart contract for token systems. ERC20 specifies basic functions for token transaction - sending and receiving tokens. Token system use internal storage to store token balance of each address. To send tokens, user creates transaction with recipient as token address. Address of token recipient and transferred amount is stored inside transaction data.

ERC20 token implements following methods with given signatures:

- dd62ed3e *allowance(address,address)*
- 095ea7b3 *approve(address,uint256)*
- 70a08231 *balanceOf(address)*
- 313ce567 *decimals()*
- 06fdde03 *name()*
- 95d89b41 *symbol()*
- 18160ddd *totalSupply()*
- a9059cbb *transfer(address,uint256)*
- 23b872dd *transferFrom(address,address,uint256)*
- 54fd4d50 *version()*

Signature can be used to identify invoked method by inspecting the first 8 characters of *input* field of transaction.

2.9 Summary

In previous sections, I have described several cryptocurrencies and discussed how to obtain information from their underlying networks. In table 2.2, I have listed technical aspects of each cryptocurrency. Table 2.3 shows information about transaction details that is possible to acquire for each cryptocurrency.

Cryptocurrency	Coin	Total supply	Block time [m]	Protocol	Features
Bitcoin	BTC	21M	10	Bitcoin	Lighting network
Bitcoin Cash	BCH	21M	10	Bitcoin	
Litecoin	LTC	84M	2.5	Bitcoin	Lighting network
Dash	DASH	18-21M	2.5	Bitcoin	PrivateSend InstantSend
Zcash	ZEC	21M	2.5	Bitcoin	Shielding
Monero	XMR	infinite	2	CryptoNote	Privacy
Ethereum	ETH	infinite	0.5	Ethereum	Smart contracts

Table 2.2: Technical aspects of cryptocurrencies

Cryptocurrency	Sender	Receiver	
Bitcoin	✓	✓	
Bitcoin Cash	✓	✓	
Litecoin	✓	✓	
Dash	✓	✓	
Zcash	✓	✓	$t \to t$
	✓	×	$t \to z$
	×	✓	$z \to t$
	×	×	$z \to z$
Monero	×	×	
Ethereum	✓	✓	

Table 2.3: Obtainable information

2.10 Identity pairing

Cryptocurrencies are by nature (pseudo)anonymous. One way how to pair address with user's identity is to request information from a cryptocurrency exchange with court order. This approach can be used only for exchanges that require verification. On some exchanges, user's verification is not mandatory or is mandatory only when certain conditions are met. Exceeding certain volume of trades is the most used condition to request verification of user's identity. Also, exchanges can be outside of state's jurisdiction and do not need to comply with court requests.

To get at least user's pseudonym, one can find information on web pages about cryptocurrencies where users have public profiles. In some cases, they provide their cryptocurrency addresses in their profiles. From this information, the pseudonym can be paired with his published addresses. And as users tend to use the same pseudonym across several web pages, one can link those profiles together. This can reveal possible identity in case another web page shows (most likely) real information. Facebook can be used as an example. Some users have their pseudonym same as their custom URL of Facebook's profile (`http://facebook.com/<username>`). Other way to get more identity information is to find the same pseudonym on website that operates inside a state's jurisdiction and request information with court order. This way, law enforcement personnel can get a user's IP address. Then, law enforcement can inspect network traffic of this IP address and scan for cryptocurrencies transactions to verify the assumption of user's identity.

2.11 Existing solutions

There are already two types of existing tools for analyzing the content of cryptocurrencies: blockchain explorers and transaction notificators.

Blockchain explorers can be used to browse cryptocurrency blockchain in a web browser. They allow users to search for the specific transaction or address by hash. They provide a list of all transactions related to the given address. This can useful for user who wants to know details about transactions of single address. It is not possible to search for multiple addresses simultaneously and user must perform one search for each address. Also, there is no possibility to send notifications when transaction (or any other event) occurs. Each cryptocurrency has own blockchain explorer, which makes systematic transaction monitoring fragmented. One of many Bitcoin's blockchain explorers is Blockchain.info[35].

Another set of existing tools is based on sending email notifications when specified address is recipient of transaction. This can be useful for users who want to be notified about incoming transactions but is far from ideal solution for tracking a large number of addresses across multiple cryptocurrencies. User cannot specify the type of involvement to be notified about (sender/receiver). Also, these services do not offer an API for push notifications which would be better suited for effective processing. Another problem is that these services support monitoring only on a small set of cryptocurrencies. Majority of them supports only Bitcoin, thus there is no way for user to monitor transactions in other cryptocurrencies. BitNotify[36] is one of these services.

[35]https://blockchain.info/
[36]http://bitnotify.com

Chapter 3

Design

This chapter is describing design of a new application, *Cryptoalarm*. *Cryptoalarm* is an application developed to satisfy needs for transaction monitoring in multiple cryptocurrencies. It allows for multiple filters for monitored addresses and is able to send REST notifications. REST notifications allow for integration with applications for post processing of transactions that match watched addresses.

3.1 Requirements

Requirement for *Cryptoalarm* was to create an application that allows transactions monitoring in multiple cryptocurrencies. Demand to split *Cryptoalarm* into two major components (monitoring application and web user interface) was placed from project Tarzan. This is due to integration of web user interface into ecosystem already developed for project Tarzan.

Following requirements were specified for an application designated to transaction monitoring:

- Ability to monitor transactions in the wide spectrum of cryptocurrencies independently
- Transaction processing in a real-time
- Offer scalability for further extension with new cryptocurrencies
- Process transactions in blocks of the main chain when chain split is detected

Following requirements were specified for a web application designated to watchlist management:

- Multi-user watchlist management
- Address and cryptocurrency specification
- Input/Output involvement selection
- Notification type selection
- Notification destination
- Notification email customization

Besides watchlist management, web application should display a list of created notifications. Notification that occurred on address specified in any watchlist needs to be shown in dashboard. The list of notifications that occurred in the specific watchlist needs to be listed separately.

3.2 Database

The database of *Cryptoalarm* needs to store information about monitored address, detected transactions and processed blocks. All database tables and its columns are described in following sections. Database schema is shown in figure 3.1.

Users

The table *users* stores information about *Cryptoalarm's* users. This table has the following columns:

- *id* - primary key
- *email*_address - unique user identification and destination of email notifications
- *password* - hashed user password
- *rest_url* - URL address to which rest notification will be sent

Settings

The table *settings* represents a key-value pairs. This table is used to store configuration settings such as default template for email notifications or email subject.

- *key* - setting identification
- *value* - setting value

Coins

The table *coins* represent cryptocurrencies supported by *Cryptoalarm*. It has the following columns:

- *id* - primary key
- *name* - cryptocurrency name as a ticker
- *explorer_url* - URL of blockchain explorer

Blocks

The table *blocks* stores information about every block processed by *Cryptoalarm*. It has the following columns:

- *id* - primary key
- *coin_id* - foreign key to table *coins* identifying cryptocurrency that block belongs to

- *number* - block's number (height)
- *hash* - block's hash

The relation between *blocks* and *coins* is necessary as blocks of each cryptocurrency can have the same number. Block's hash is stored for the purpose of detecting chain-split and block in which it occurred. This is possible due to mismatch between block's hash of given number and hash returned from cryptocurrency node.

Addresses

The table *addresses* represents every address known to *Cryptoalarm.* It has the following columns:

- *id* - primary key
- *hash* - address hash
- *coin_id* - foreign key to table *coins* identifying cryptocurrency that address belongs to

The relation between *addresses* and *coins* is necessary as multiple cryptocurrencies can have the same format of addresses. Bitcoin and Bitcoin Cash can be used as an example.

Watchlists

The table *watchlists* represents watchlists created by *Cryptoalarm's* users. It has the following columns:

- *id* - primary key
- *name* - name of a watchlist
- *type* - address' involvement type (in/out/inout)
- *notify* - notification type (none/email/rest/both)
- *address_id* - foreign key to table *addresses* identifying which watched address
- *user_id* - foreign key to table *users* identifying user that watchlist belongs to
- *email_template* - template for email notifications

Address is saved outside of watchlist to prevent information duplication as multiple users can create watchlist for the same address.

Identities

The table *identities* represents every identity that is parsed by *IdentityParser.* It has the following columns:

- *id* - primary key
- *url* - URL where identity was found

- *label* - pseudonym
- *source* - web page name
- *address_id* - foreign key to table *addresses* identifying address paired with identity

Address is referenced to the table *addresses* which allows to match identities, addresses and existing watchlists.

Notifications

The table *notifications* represents notifications generated when any transaction involves a watched address. It has the following columns:

- *id* - primary key
- *watchlist_id* - foreign key to table *watchlists* identifying watchlist that notification belongs to
- *block_id* - foreign key to table *blocks* identifying block that transaction occurred in
- *tx_hash* - hash of transaction
- *created_at* - time of creation

Relation to the table *blocks* is necessary as notifications created for transaction in blocks not included in main chain need to be deleted. Automatic deletion is achieved by constraint `ON DELETE CASCADE`. The relation to the table *watchlists* is designed to identify watchlist that generated given notification.

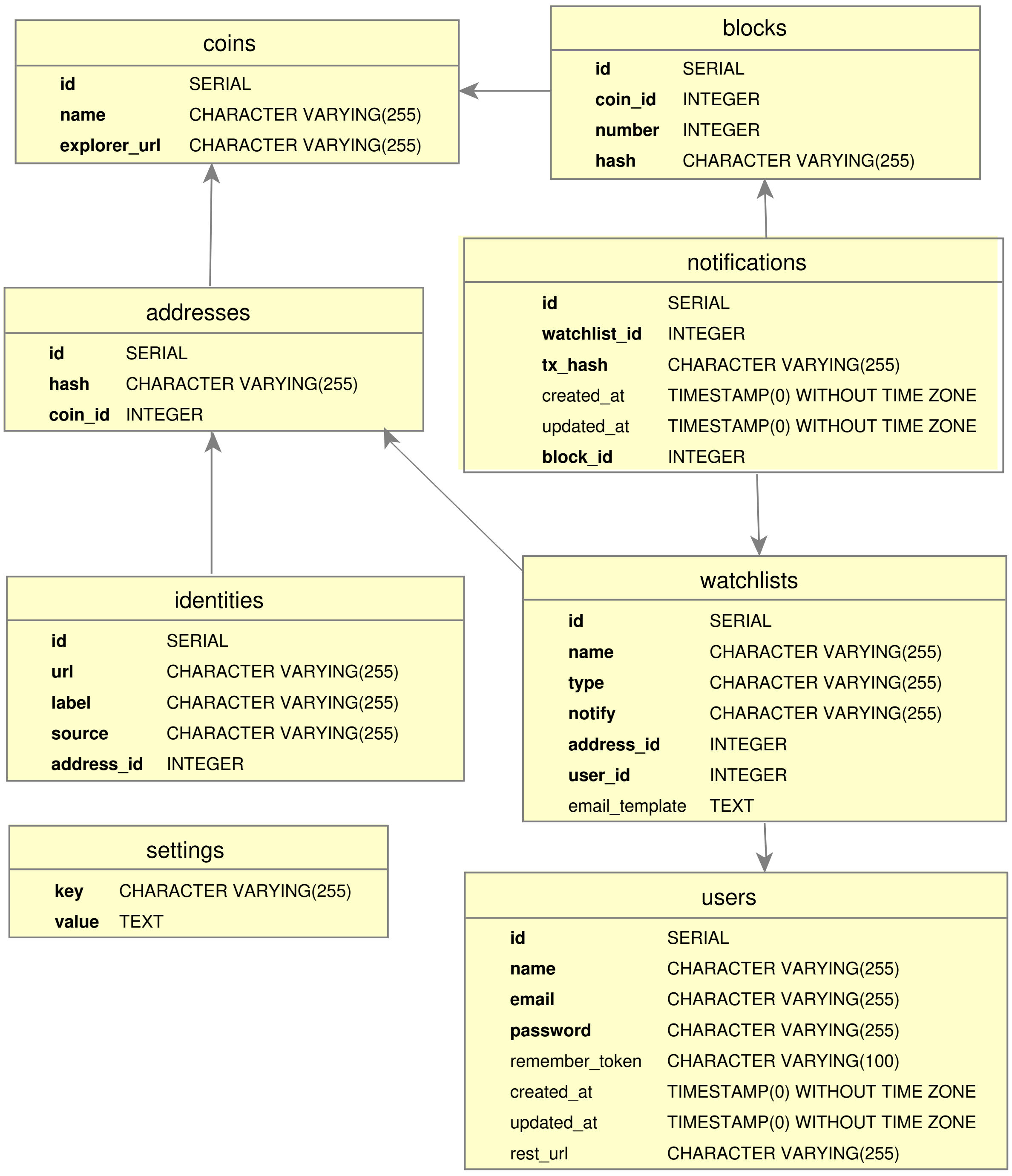

coins
id SERIAL
name CHARACTER VARYING(255)
explorer_url CHARACTER VARYING(255)
blocks
id SERIAL
coin_id INTEGER
number INTEGER
hash CHARACTER VARYING(255)
notifications
id SERIAL
watchlist_id INTEGER
tx_hash CHARACTER VARYING(255)
created_at TIMESTAMP(0) WITHOUT TIME ZONE
updated_at TIMESTAMP(0) WITHOUT TIME ZONE
block_id INTEGER
addresses
id SERIAL
hash CHARACTER VARYING(255)
coin_id INTEGER
identities
id SERIAL
url CHARACTER VARYING(255)
label CHARACTER VARYING(255)
source CHARACTER VARYING(255)
address_id INTEGER
watchlists
id SERIAL
name CHARACTER VARYING(255)
type CHARACTER VARYING(255)
notify CHARACTER VARYING(255)
address_id INTEGER
user_id INTEGER
email_template TEXT
settings
key CHARACTER VARYING(255)
value TEXT
users
id SERIAL
name CHARACTER VARYING(255)
email CHARACTER VARYING(255)
password CHARACTER VARYING(255)
remember_token CHARACTER VARYING(100)
created_at TIMESTAMP(0) WITHOUT TIME ZONE
updated_at TIMESTAMP(0) WITHOUT TIME ZONE
rest_url CHARACTER VARYING(255)

Figure 3.1: Database schema

3.3 Monitoring application

The purpose of monitoring application is to scan blockchains of cryptocurrencies in order to detect new blocks, to process transactions and to send notifications when monitored address is involved in any transaction. The scanning of each cryptocurrency's blockchain needs to be done in parallel. This is due to the different block times of each cryptocurrency. Those intervals can overlap or new blocks can be generated in approximately at the same time. Transaction processing of one cryptocurrency could become delayed as it would wait until blocks of all other cryptocurrencies were processed in case of serial processing. Also, transaction processing for each cryptocurrency can take significantly different amounts of time. This is caused by each cryptocurrency storing distinct types of information about inputs of transaction. Ethereum can used as an example. Only one RPC call is required to obtain information about transaction inputs. On the other hand, transaction in Bitcoin (and its derivates) requires $1 + len(inputs)$ RPC calls. This is caused by the process of identifying senders' address. The relation between these classes can be seen on figure 3.2. I have designed class *Coin* that defines methods interacting with blockchain and processing results. From this class, there are two direct child classes implementing specific operations for two cryptocurrencies, Bitcoin and Ethereum.

Cryptoalarm is designed to be easily extensible to perform transaction monitoring in new cryptocurrencies with minimum effort. It is an important aspect of *Cryptoalarm* as new cryptocurrencies are created almost on daily basis. By default, *Cryptoalarm* supports six cryptocurrencies described in chapter Analysis 2 with the exception of Monero. Monero does not allow for transaction monitoring as its addresses are obfuscated (see 2.7). Functionality of monitoring application is split into several classes. Description of each class is in the following paragraphs.

Class *Monitor* monitors cryptocurrencies blockchains and detects new blocks. Beside monitoring, it processes blocks and transactions. Also, it handles the lifetime of classes *Notifier* and *Database*. For each cryptocurrency, a new thread is started and method *worker* invoked. The worker is awakened in intervals corresponding to block time of each cryptocurrency. When a new block is detected, the list of transactions is obtained. Subsequently, each transaction is processed and inserted into the queue synchronized with *Notifier*. Method *test_connection* is used to test connectivity of *Notifier* and each cryptocurrency's RPC node. A handler of *SIGINT* is attached to function *shutdown* to perform graceful shutdown. The block processing is stopped after all remaining blocks are processed when the signal *SIGINT* is received. Block processing is terminated immediately in case of a network error. This behaviour is designed to prevent infinite repetition of failed requests which would prevent the application from shutting down.

Class *Coin* is designed to interact with cryptocurrency's network node by using its RPC API. Classes for each cryptocurrency are inherited from *Coin*. Major classes are *BTC* for Bitcoin and *ETH* for Ethereum. These classes specify names and arguments of RPC calls and handle block and transaction processing. Classes for other cryptocurrencies are inherited from *BTC*, specifically *BCH* (Bitcoin Cash), *LTC* (Litecoin), *DASH* (Dash) and *ZEC* (Zcash). Those classes only specify block time as it is their only difference from Bitcoin in terms of transaction processing. In case a network error is encountered during RPC API call to cryptocurrency's full node, this call is repeated in doubling intervals until it reaches maximum threshold. After that, a static interval between repetitions is used. Each class representing a cryptocurrency stores the detail of every transaction requested by processing a new transaction. Multiple inputs of a single transaction can originate from the same transaction. Input transactions are stored to prevent repeated requests for the same data which improves processing times. Class *ETH* also parses transactions in smart contracts based on ERC20 specification. It is possible to monitor every interaction with smart contract by specifying its address into a watchlist. In case an address is recipient of token transfer, this address is neither sender nor receiver of transaction. Sender is the user, who initiates token transfer, and recipients is the smart contract of given token. Receiver's address is specified inside transaction data. This data is parsed and if method signature matches transfer function of ERC20 contract, its recipient is also added as receiver of transaction.

To extend *Cryptoalarm* with a new cryptocurrency, a new class must be derived from *Coin*. If this cryptocurrency is based on Bitcoin then block time change is needed. All of the following methods have to be implemented in case of cryptocurrency using different protocol than Bitcoin:

- *get_last_block_number* - obtain the number of last block
- *get_block_hash* - obtain the hash of given block
- *get_block_creation_time* - obtain the time of block creation
- *get_block* - obtain block data structure for the given block
- *get_block_transactions* - obtain the list of transaction hashes from the given block
- *get_transaction* - obtain data structure of the given transaction
- *get_transaction_io* - return sets of inputs and outputs for the given transaction

I designed the class *Notifier* to create notifications. After all transactions are processed, its inputs, outputs and hash are inserted in the queue synchronized with *Monitor*. When a new transaction is detected, its inputs and outputs are compared to watched addresses. Transaction is attached to its watchlist in case any of its inputs or outputs match watchlist's address. Watchlist data structure (as shown in listing 3.1) is used to store relevant data and metadata. This structure also stores block number and block id. Block id is identifier of the block record stored in database.

```
{
    "coin ticker": {
        "explorer_url": "url",
        "data": {
            "address1": {
                "txs": {
                    "in": [
                        [block_id, block_number, "tx hash"],
                        ...
                    ],
                    "out": [
                        [block_id, block_number, "tx hash"],
                        ...
                    ],
                },
                "users": {
                    "in": [user1, user2, ..., userN],
                    "out": [user1, user2, ..., userN],
                    "inout": [user1, user2, ..., userN],
                }
            },
        }
    },
}
```

Listing 3.1: Watchlist data structure

This structure stores each address and its related watchlists independently because multiple watchlists may be attached to the same address. All incoming transactions are saved in the attribute *in*. All outgoing transactions are saved in the attribute *out*. Users who created watchlist for given address are stored in attributes *users*→*in*, *users*→*out* and *users*→*inout*. This separation is intended to limit notifications for a particular address involvement previously chosen by a user. The reason for having users attached in the attribute *inout* instead of both *in* and *out* is that this would create two notifications for each user. The first notification would contain input transactions. The second would contain output transactions. User data structure (as shown in listing 3.2) stores all information about notification types, its destination and eventually email template.

```
{
    "type": string,
    "watchlist_name": string,
    "notify": string,
    "watchlist_id": number,
    "email_template": string,
    "user_id": number,
    "email": string,
    "rest_url": string
}
```

Listing 3.2: User data structure

All matched transactions are sent as notifications as specified in watchlist. Notifications are sent in customizable interval, which can be altered in configuration. Also, *Notifier* is designed to refresh watchlist data in specific interval. I decided to use data refresh instead of interacting directly with the database, thus experiencing a better performance in case of large number of watchlists. Fetching watchlist information from database for every transaction would lead to more queries and could become the bottleneck of the application.

Class *Sender* defines the interface for notification senders. It consists of methods:

- *add* - insert transaction which needs to be send as notification
- *send* - to send notifications
- *test_connection* - to test network connectivity

Classes *Mailer* and *Rest* are derived from *Sender*. Each of these classes implements functionality to send notifications to user's defined destination. Both classes can recover from network errors. All to-be-sent notifications are stored inside each class. Every item in this queue is processed and sent as notification when method *send* is invoked. Items that are not sent due to network errors are appended to the end of this queue again. This is due to possibility of temporary unreachable REST server. This way, user receives every notification even if his server is offline in time the notification is created. To create different types of notification, a new class must be derived from the class *Sender*.

Class diagram of the monitoring application is shown in figure 3.2.

3.4 Web application

The main purpose of web application is watchlist management. Additionally, it includes component *AddressMatcher* to match addresses with cryptocurrencies they belong. *IdentityParser* is a component used for parsing user's identities and their addresses.

Web application is split into several entities to achieve these goals. There is a *User*, *Coin*, *Address*, *Watchlist* and *Notification*.

- *User* represents user and user's settings such as email address, email template and REST URL
- *Coin* represents cryptocurrency. It contains name (a ticker) and URL to blockchain explorer.
- *Address* represents address and cryptocurrency it belongs to.
- *Watchlist* is a combination of a user and an address. Additionally, it contains monitor type (incoming, outgoing, both) and notification settings (rest/email).
- *Notification* represent a notification created when an address is involved in a transaction.

Web application is designed to have multiple screens. The first screen is *Dashboard*, which is designed to be a user's entry point into the application. It contains a list of notifications about any addresses that user specified watchlist on. This list shows watchlist name, transaction hash (linking to the blockchain explorer) and notification timestamp. Another screen is a *Watchlist*. The *Watchlist* shows all notifications created in according to rules specified in given watchlist. Also, there is a list of identities that match watchlist's address. Subsequently, there is an overview of rules defined for the given watchlist as cryptocurrency, involvement type and notification type. Also, there is a preview of a template used to create email notifications.

The screen for creating and editing watchlist offers a form to fill all required information. Additionally, the cryptocurrency is automatically identified with the use of the *AddressMatcher* when user specifies an address. Also, there is a field to specify email template. User can utilize placeholders which will be replaced with corresponding information when email notification is created:

- *name* - watchlist name
- *coin* - coin name
- *address* - address hash
- *txs* - list of transactions

In user's profile, it is possible to specify an URL that all REST notifications will be sent to. REST notifications have the following format:

```
{
   "address": hash,
   "coin": ticker,
   "watchlist": name,
   "transactions": [
       [block_number, hash],
       ...
   ]
}
```

Listing 3.3: REST notification format

Web application defines one API method *api/identify*. This method is designed to be called asynchronously to identify the related cryptocurrencies. The address is the only argument accepted by this method. The following data structure is returned by this method:

```
{
    "status": bool,
    "coins": [
        "coin1",
        ...
    ]
}
```

Listing 3.4: The result data structure of *api/identify*

The attribute *status* is set to *true* in case that the address matches at least one cryptocurrency. The attribute *coins* contains all cryptocurrencies that use the same format of an address.

3.4.1 AddressMatcher

AddressMatcher is a component of web application to identify cryptocurrency that address belongs to. *AddressMatcher* has two major usages in the web application. First one is to enhance user experience when creating or editing watchlist. When user specifies address, an asynchronous requested is sent to *AddressMatcher* (*api/identify*) which identifies appropriate cryptocurrencies. After that, the select box for cryptocurrency is preselected with the identified cryptocurrencies. Address format can correspond to multiple cryptocurrencies (i.e. Bitcoin and Bitcoin Cash). If multiple results are returned, user is allowed to selected only from those cryptocurrencies. Another usage of *AddressMatcher* is for *IdentityParser*. When web page is retrieved by *IdentityParser*, its source is then passed to *AddressMatcher*. *AddressMatcher* recognizes all supported addresses and returns all the list of identified cryptocurrencies. *AddressMatcher* has the following methods:

- *identify_address* - identify the given address
- *match_addresses* - match all possible addresses and identify them

3.4.2 IdentityParser

IdentityParser is designed to obtain all published addresses from user's profiles on Bitcointalk[1] forum. User profiles are selected for the probability of address occurrence. Another option would be to parse every thread across the whole forum but the probability of a reply containing address is lower than in a profile.

[1]Bitcointalk, http://bitcointalk.org

IdentityParser is designed to process every profile by incrementing id in the URL up to an identifier of the last registered user. The last processed profile is then saved in database table *settings(key=bitcointalk_last_id)*. When run multiple times, *IdentityParser* is designed to start from the last processed profile and continue upwards to the latest currently registered user. This way, it can be run on regular basis to process new profiles. Also, it could be run even for existing profiles as users can specify new addresses. This way, the most complete database of users' identities can be created.

After the source code of the page is obtained, it is then passed to *AddressMatcher*. *AddressMatcher* recognizes all possible supported addresses on given page and identifies cryptocurrencies that those addresses belong to. Then, a new identity is created for every recognized address and every corresponding cryptocurrency. Identities are stored in table *identities* with corresponding address linked to the table *addresses*.

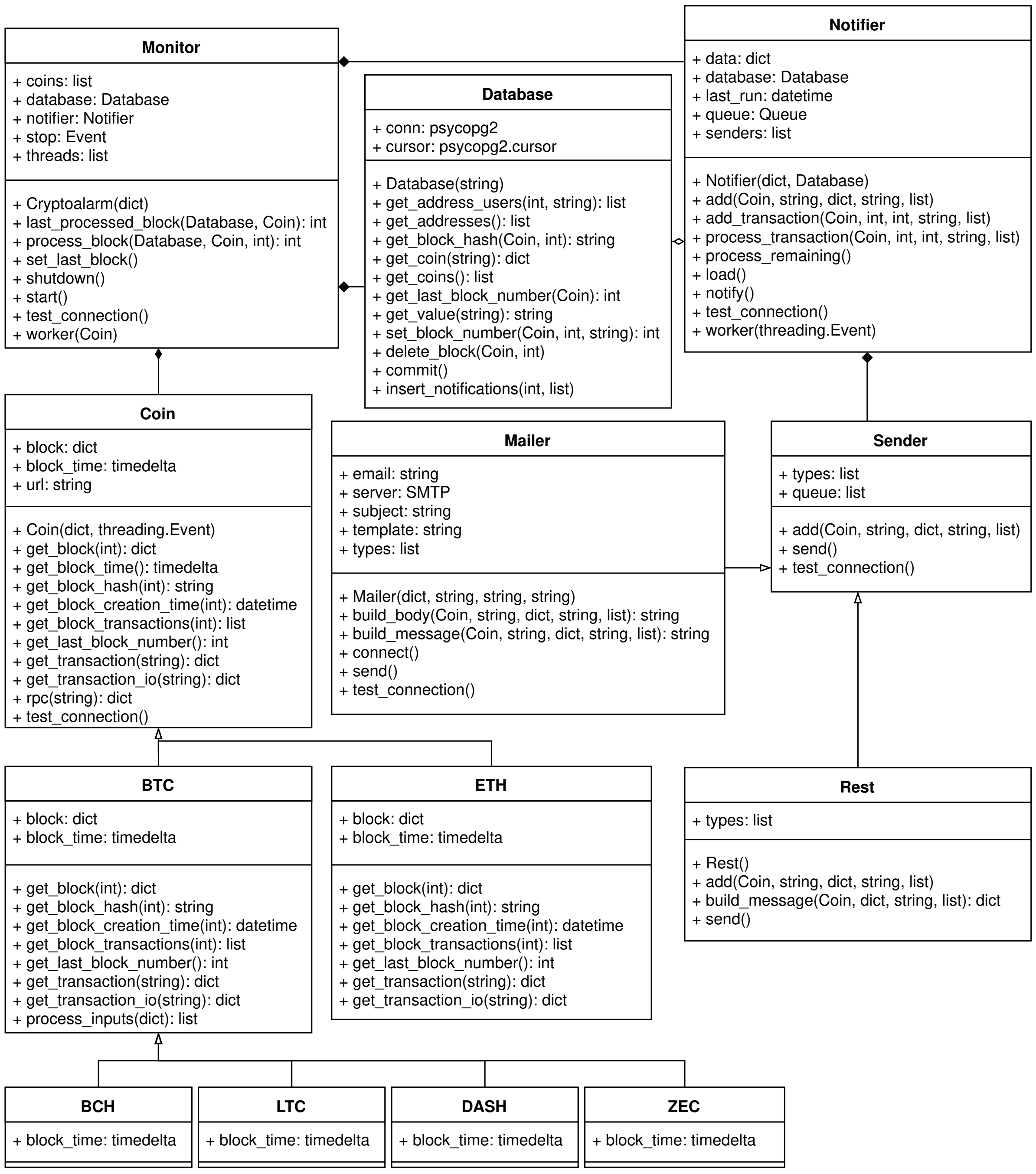

Monitor
+ coins: list
+ database: Database
+ notifier: Notifier
+ stop: Event
+ threads: list
+ Cryptoalarm(dict)
+ last_processed_block(Database, Coin): int
+ process_block(Database, Coin, int): int
+ set_last_block()
+ shutdown()
+ start()
+ test_connection()
+ worker(Coin)
Database
+ conn: psycopg2
+ cursor: psycopg2.cursor
+ Database(string)
+ get_address_users(int, string): list
+ get_addresses(): list
+ get_block_hash(Coin, int): string
+ get_coin(string): dict
+ get_coins(): list
+ get_last_block_number(Coin): int
+ get_value(string): string
+ set_block_number(Coin, int, string): int
+ delete_block(Coin, int)
+ commit()
+ insert_notifications(int, list)
Notifier
+ data: dict
+ database: Database
+ last_run: datetime
+ queue: Queue
+ senders: list
+ Notifier(dict, Database)
+ add(Coin, string, dict, string, list)
+ add_transaction(Coin, int, int, string, list)
+ process_transaction(Coin, int, int, string, list)
+ process_remaining()
+ load()
+ notify()
+ test_connection()
+ worker(threading.Event)
Coin
+ block: dict
+ block_time: timedelta
+ url: string
+ Coin(dict, threading.Event)
+ get_block(int): dict
+ get_block_time(): timedelta
+ get_block_hash(int): string
+ get_block_creation_time(int): datetime
+ get_block_transactions(int): list
+ get_last_block_number(): int
+ get_transaction(string): dict
+ get_transaction_io(string): dict
+ rpc(string): dict
+ test_connection()
Mailer
+ email: string
+ server: SMTP
+ subject: string
+ template: string
+ types: list
+ Mailer(dict, string, string, string)
+ build_body(Coin, string, dict, string, list): string
+ build_message(Coin, string, dict, string, list): string
+ connect()
+ send()
+ test_connection()
Sender
+ types: list
+ queue: list
+ add(Coin, string, dict, string, list)
+ send()
+ test_connection()
BTC
+ block: dict
+ block_time: timedelta
+ get_block(int): dict
+ get_block_hash(int): string
+ get_block_creation_time(int): datetime
+ get_block_transactions(int): list
+ get_last_block_number(): int
+ get_transaction(string): dict
+ get_transaction_io(string): dict
+ process_inputs(dict): list
ETH
+ block: dict
+ block_time: timedelta
+ get_block(int): dict
+ get_block_hash(int): string
+ get_block_creation_time(int): datetime
+ get_block_transactions(int): list
+ get_last_block_number(): int
+ get_transaction(string): dict
+ get_transaction_io(string): dict
Rest
+ types: list
+ Rest()
+ add(Coin, string, dict, string, list)
+ build_message(Coin, dict, string, list): dict
+ send()
BCH
+ block_time: timedelta
LTC
+ block_time: timedelta
DASH
+ block_time: timedelta
ZEC
+ block_time: timedelta

Figure 3.2: Monitoring application class diagram

Chapter 4

Implementation

Cryptoalarm consists of two components, the monitoring application and the web application. The monitoring application is responsible for interactions with blockchains of cryptocurrencies and generating notifications. The web application is responsible for watchlist management.

4.1 Monitoring application

The monitoring application of *Cryptoalarm* is implemented in Python. I choose Python because of its ability for fast prototyping, readable code and multiplatformity. Monitoring application can be run on any existing operating system with Python. Also, Python scripts do not require compilation, which makes development easier. Another reason is Python's built-in support for data structures such as JSON, lists and sets.

I decided to implement classes of each cryptocurrency on my own even though there are already several existing Python packages[1,2,3]. My reasons to create custom implementation is that those packages do not adhere to the same interface. I would need to implement an adapter for every cryptocurrency to be able to work with those packages in a unified manner. Limited support of all RPC API calls needed for *Cryptoalarm's* functionality is another reason. With custom implementation, *Cryptoalarm* is now easily extensible to support new cryptocurrencies. That would not be possible with third-party packages. To extend *Cryptoalarm* with a new cryptocurrency, the developer is only required to create a new class inherited from the class *Coin* and specify methods described in chapter 3. In case of cryptocurrency based on Bitcoin, it is possible to inherit from the class *BTC* instead of *Coin* and only specify block time (if it differs).

Package *Cryptoalarm* consists of four modules: *Coin*, *Monitor*, *Database* and *Notifier*. Module *Coin* implements all operations for interaction with blockchains of cryptocurrencies and processing of obtained data. A class is defined for every supported cryptocurrency. Module *Notifier* implements classes for notification processing.

Also, I have developed a *cryptoshell*. This utility can be used to call methods of each cryptocurrency independently on the *Cryptoalarm*. *Cryptoshell* takes the following arguments:

- *Coin* - the name of cryptocurrency as defined in module *Coin*

[1]python-bitcoinrpc, `https://github.com/jgarzik/python-bitcoinrpc`
[2]ethjsonrpc, `https://github.com/ConsenSys/ethjsonrpc`
[3]python-darkcoinrpc, `https://github.com/vertoe/python-darkcoinrpc`

- *method* - method that will be called in given cryptocurrency

Any additional arguments are passed to selected *method*. The documentation of monitoring application can be found in the attached CD in folder `/src/cryptoalarm/doc/`.

4.1.1 Dependencies

The monitoring application has the following dependencies:

- Python 3
- postgresql

4.2 Web application

I have used PHP framework Laravel[4] 5.5 for the web application implementation. This was one of the requirements as the rest of the applications developed for project Tarzan is written in Laravel. This allows for integration inside Tarzan project.

4.2.1 AddressMatcher

AddressMatcher uses the following regular expressions to identify cryptocurrency from the address format:

- Bitcoin:
  `([13][a-km-zA-HJ-NP-Z1-9]{25,33}|bc1([A-Za-z0-9]{39}|[A-Za-z0-9]{59}))`
- Bitcoin Cash: `[13][a-km-zA-HJ-NP-Z1-9]{25,33}`
- Litecoin: `[LM3][a-km-zA-HJ-NP-Z1-9]{25,33}`
- Dash: `X[1-9A-HJ-NP-Za-km-z]{25,33}`
- Zcash: `t|[a-zA-Z0-9]{34}`
- Ethereum: `0x[a-fA-F0-9]{40}`

4.2.2 IdentityParser

IdentityParser is implemented as a part of the web application. Bitcointalk forum uses Cloudflare DDOS protection[5]. To bypass this protection, a browser that with Javascript has to be used. I have decided to use Selenium framework which allows the connection to headless browser.

[4]Laravel, https://laravel.com/
[5]Cloudflare DDOS protection, https://www.cloudflare.com/ddos/

4.2.3 Dependencies

The web application has the following dependencies:

- postgresql
- composer
- npm
- Java
- Selenium standalone server[6]
- PHP >= 7.0.0, extensions: OpenSSL, PDO, Mbstring, Tokenizer, XML, pgsql

[6]Selenium standalone server, `https://www.seleniumhq.org/download`

Chapter 5

Testing

In this chapter, I describe how monitoring application of *Cryptoalarm* is verified and validated. Then, I describe how I measured the performance of *Cryptoalarm* with large amounts of watchlists. Performance evaluation is the most important aspect because *Cryptoalarm* was designed to be able to monitor on large number of addresses.

5.1 Verification

Verification is performed in two steps:

1. Blockchain interactions & transaction processing
2. Notification generation

I have developed a test suit to test blockchain interactions and transaction processing using *unittest*[1] Python framework. Blockchain interactions are calls to underlying cryptocurrency node. Developed test cases (*cryptoalarm/tests/test_coins.py*) include interaction with Bitcoin (*TestBTC*) and Ethereum (*TestETH*) nodes. The rest of the cryptocurrencies are not subjects of testing because all of them inherit functionality from class *Bitcoin* and behave exactly the same. Both test cases consist of four tests:

- *test_get_block_hash*
- *test_get_block_creation_time*
- *test_get_block_transactions*
- *test_get_transaction_io*

All test cases combined cover the complete functionality of each cryptocurrency. Other methods are tested indirectly as they are invoked by methods covered by those test cases. To test interactions with cryptocurrency nodes, I used cryptocurrency nodes run within the project Tarzan. This was possible for every cryptocurrency except Ethereum. To test interactions with Ethereum node, I have created a local Ethereum blockchain with the use of Truffle framework[2] and its tool Ganache[3]. Ganache is an implementation of Ethereum

[1] Unittest, https://docs.python.org/3/library/unittest.html
[2] Truffle framework, http://truffleframework.com/
[3] Ganache, http://truffleframework.com/ganache/

node with graphical interface. Ganache comes with a number of predefined addresses with test balances This allows for transactions testing even without mining new blocks to get a reward first. I have used Mist[4] to create transactions inside this blockchain. To create a smart contract, I have used a custom ERC20 token derived from OpenZeppelin [5] Standard Token.

Test case *TestNotifier* (*cryptoalarm/tests/test_notifier.py*) covers the functionality of notification generation. A mock object (*DatabaseMock*) of *Database* was created to remove the dependency on the database for this test. To set up a test environment, an instance of *Notifier* is created and then artificial transaction is processed. The following methods were created to verify functionality of *Notifier*:

- *test_watchlist*
- *test_process_transaction*
- *test_unique_notification_inout*

5.2 Integration testing

For integration testing, I have selected few of the most used addresses in each cryptocurrency. Then, I have created a watchlist for each address with combination of possible types of involvement (in/out/inout) and notifications types (rest/email/both). After 100 blocks were processed by *Cryptoalarm*, I have compared results of notifications shown in the web application to blockchain explorers. Notifications created for Ethereum were compared with the output of Ganache. Selected addresses were:

- Bitcoin: 1dice8EMZmqKvrGE4Qc9bUFf9PX3xaYDp - SatoshiDice[6]
- Bitcoin Cash: 1KFHE7w8BhaENAswwryaoccDb6qcT6DbYY - F2Pool[7]
- Litecoin: LhyLNfBkoKshT7R8Pce6vkB9T2cP2o84hx
- Dash: XpESxaUmonkq8RaLLp46Brx2K39ggQe226
- Zcash: t1KLGj3izuKveu1eFZUiwp3BEKHQAiYv2Z7

To test the processing of a smart contract transaction, I've created a watchlists for:

- Address of ERC20 token smart contract
- Address used as destination of token transfer

Then, I have observed if notifications for both addresses were created.

[4]Mist, https://github.com/ethereum/mist
[5]OpenZeppelin, https://github.com/OpenZeppelin/openzeppelin-solidity
[6]SatoshiDice, a gambling site, https://satoshidice.com/
[7]F2Pool, a mining cluster, https://www.f2pool.com/

5.3 Performance testing

To test the performance of *Cryptoalarm*, I have focused on the speed of transaction processing in the *Notifier.* I did not measure the performance of interactions with blockchains of cryptocurrencies as it is affected by the connection speed between the application and the node. Interactions with Zcash node running on the same device as *Cryptoalarm* were almost 4× faster compared to node located on the network with average response time of 15ms. Also, the number of cryptocurrencies supported does not have a significant effect on the performance because *Cryptoalarm* instance for every cryptocurrency is run in separated thread.

To test the performance of *Cryptoalarm*, I have measured the average number of transactions in Bitcoin blocks and average number of their inputs and outputs. I have chosen Bitcoin because it is currently the most used cryptocurrency. There are 20 730 000 transactions in the last 10 000 Bitcoin blocks. The average number of transactions per block equals to 2073. Those transactions have on average 2.577 inputs and 2.398 outputs. For the testing purposes, I have adjusted these numbers to 2.5 inputs and 2.5 outputs per transaction.

To the measure performance, I have created scenarios with:

- 10 watchlists
- 100 watchlists
- 1 000 watchlists
- 5 000 watchlists
- 10 000 watchlists

Then, for every scenario, I have created 2073 transactions with the average of 2.5 inputs and 2.5 outputs. Those transactions were generated artificially to have the occurrence of watched address of:

- 10%
- 25%
- 50%
- 75%

I have measure the performance in two separated phases. First phase covers processing transactions and storing this information inside *Cryptoalarm*. The results are shown on figure 5.1. As you can see the number of watchlists has major impact on performance. Percentage of watched address occurrence has only minor impact.

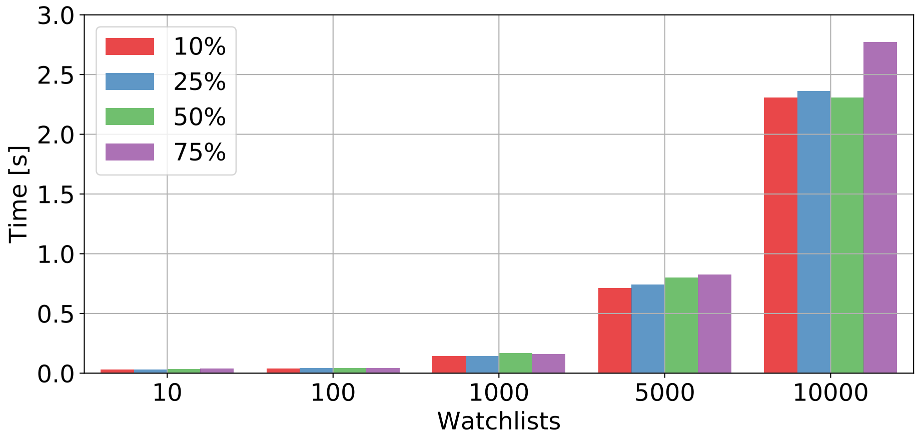


Figure 5.1: Processing times

Second phase consists of generating notifications for transactions matching watchlists. The results are shown on figure 5.2. The number of occurrences of watched addresses had more impact on notification times in comparison to processing time. This is due to the number of notifications that needs to be generated and saved in the database.

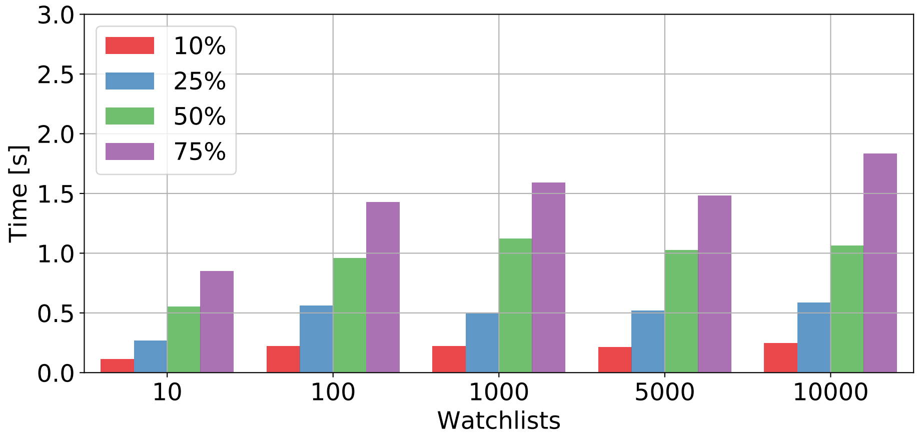


Figure 5.2: Notification times

Cryptoalarm allows to specify an interval for sending notifications and reloading watchlists. Interval for reloading watchlist should be set to reflect the needs of an operator of *Cryptoalarm*. I would recommend setting reload interval around one hour for *Cryptoalarm* run as publicly available service. I would not recommend setting reload interval in seconds as it can take few seconds to transfer all watchlist data. Reload times in relation to the number of watchlists is shown in table 5.1.

Watchlists	Time [s]
10	0.0158
100	0.0953
1 000	0.9779
5 000	4.7848
10 000	6.5604

Table 5.1: Reload times

All measurements were performed on Intel(R) Core(TM) i5-3427U CPU @ 1.80GHz. Presented data are the averages of 1 000 iterations.

Chapter 6

Conclusion

The goal of this thesis was to monitor activities in cryptocurrency blockchains in order to raise alarms when specified activity has been detected. Monitoring address involvement in transactions was the main focus of this thesis as transactions are the core activities that occur in cryptocurrency blockchains. This type of monitoring can be used by governments, banking institutions or law enforcement agencies to track movements of funds on problematic addresses. Ransomware or malware crypto miner can be used as the examples of these addresses.

In this thesis, I have discussed seven of the most used cryptocurrencies and the possibilities to monitor involvement of specific address in transactions. I have created a new application *Cryptoalarm* based on analysis of cryptocurrencies and existing tools. *Cryptoalarm* specializes in systematic transaction monitoring with the focus on extensibility with new cryptocurrencies as described in chapter 3. Currently, *Cryptoalarm* supports Bitcoin, Bitcoin Cash, Litecoin, Dash, Zcash and Ethereum. Also, it supports transfers inside ERC20 token systems built on top of Ethereum smart contracts. Monero uses several features to obfuscate address' details as described in chapter 2.7, which makes it hard to monitor Monero and yields why this cryptocurrency is not supported by *Cryptoalarm*. Another specific situation is for cryptocurrency Zcash. Zcash allows only monitoring of transaction with transparent addresses (t-addresses). Z-addresses are protected by zero-knowledge proofs, specifically zk-SNARKs. Besides console application for address monitoring, I have developed the web application for watchlist management. All these components form the *Cryptoalarm*, which allows users to set up custom watchlists with a filter for the specific involvement of addresses inside transactions. *Cryptoalarm* can raise alarms in case of a watched address detection in a transaction. To let users know about such events, alarms can be sent as notifications. Currently, the supported notification types are emails and REST calls.

To test the performance of created application, I have measured processing times with multiple scenarios (10 - 10 000 watchlists) and multiple transaction datasets. *Cryptoalarm* can process 2073 (average number of transactions in Bitcoin) with 10 000 watchlists in seconds. The exact results are described in chapter 5.3.

Cryptoalarm was developed as a part of project Tarzan [4], which aims on development of a set of tools for forensic analysis of cryptocurrencies.

I have extended thesis beyond the assignment with cryptocurrencies Dash, Litecoin and Bitcoin Cash. Another extension is a component *AddressMatcher* used to identify cryptocurrencies related to an address. Also, I have developed component *IdentityParser* used to obtain metadata related to Bitcointalk forum users and their addresses.

I have presented *Cryptoalarm* in student's conference Excel@FIT with a poster. *Cryptoalarm* is published as an open source in GitHub repository *vokracko/cryptoalarm*[1].

As the future work, it is possible to extend *Cryptoalarm* to support additional cryptocurrencies. This can be done easily due to the modular design of the application. Another possibility for the continuation is a creation of more identity parsers, which could parse user's identities from other publicly available sources. Also, the web application can be extended with importer of past transactions. This extension would provide user with a complete list of address' transactions accessible within the web application.

[1]Cryptoalarm repository, `https://github.com/vokracko/cryptoalarm`

Bibliography

[1] *Bitcoin Energy Consumption Index.* [Online; visited 14.3.2018]. Retrieved from: https://digiconomist.net/bitcoin-energy-consumption

[2] *ERC20 Token Standard.* [Online; visited 16.3.2018]. Retrieved from: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

[3] *Find My Coins - Bitcoin forks.* [Online; visited 16.3.2018]. Retrieved from: http://www.findmycoins.ninja/

[4] *Integrated platform for analysis of digital data from security incidents.* [Online; visited 20.4.2018]. Retrieved from: http://www.fit.vutbr.cz/units/UIFS/grants/index.php?id=1063

[5] *View the Bitcoin cryptocurrency specifications in detail.* [Online; visited 16.3.2018]. Retrieved from: http://mapofcoins.com/bitcoin

[6] Ben-Sasson, E.; Bentov, I.; Horesh, Y.; et al.: *Scalable, transparent, and post-quantum secure computational integrity.* Cryptology ePrint Archive, Report 2018/046. 2018. https://eprint.iacr.org/2018/046.

[7] Bentov, I.; Gabizon, A.; Mizrahi, A.: Cryptocurrencies Without Proof of Work. In *Financial Cryptography and Data Security*, edited by J. Clark; S. Meiklejohn; P. Y. Ryan; D. Wallach; M. Brenner; K. Rohloff. Berlin, Heidelberg: Springer Berlin Heidelberg. 2016. ISBN 978-3-662-53357-4. pp. 142–157.

[8] Buterin, V.: *Change difficulty adjustment to target mean block time including uncles.* [Online; visited 20.4.2018]. Retrieved from: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-100.md

[9] Buterin, V.; et al.: *A Next-Generation Smart Contract and Decentralized Application Platform.* [Online; visited 10.12.2017]. Retrieved from: https://github.com/ethereum/wiki/wiki/White-Paper

[10] Central Intelligence Agency: *The World Factbook 2018.* [Online; visited 14.3.2018]. Retrieved from: https://www.cia.gov/library/publications/the-world-factbook/rankorder/2233rank.html

[11] Duffield, E.; Diaz, D.: *Dash: A Privacy-Centric Crypto-Currency.* [Online; visited 11.12.2017]. Retrieved from: https://github.com/dashpay/dash/wiki/Whitepaper

[12] gmaxwell: *CoinJoin: Bitcoin privacy for the real world*. [Online; visited 14.3.2018].
Retrieved from: https://bitcointalk.org/index.php?topic=279249

[13] Hopwood, D.; Bowe, S.; Hornby†, T.; et al.: Zcash Protocol Specification.
Retrieved from:
https://github.com/zcash/zips/blob/master/protocol/protocol.pdf

[14] Lee, T. B.: *Bitcoin fees are skyrocketing*. [Online; visited 10.3.2018].
Retrieved from: https://arstechnica.com/tech-policy/2017/12/bitcoin-fees-are-skyrocketing/

[15] Lerner, S. D.: *DagCoin Draft*. [Online; visited 20.4.2018].
Retrieved from: https://bitslog.files.wordpress.com/2015/09/dagcoin-v41.pdfcryptonote.org/cns/cns010.txt

[16] Lombrozo, E.; Lau, J.; Wuille, P.: *Segregated Witness (Consensus layer)*. [Online; visited 20.4.2018].
Retrieved from:
https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki

[17] Mengerian: *Dgenr8's Difficulty Adjustment Algorithm Explained*. [Online; visited 20.4.2018].
Retrieved from: https://medium.com/@Mengerian/dgenr8s-difficulty-adjustment-algorithm-explained-e77aa47eb281

[18] Nakamoto, S.: *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online; visited 10.12.2017].
Retrieved from: https://bitcoin.org/bitcoin.pdf

[19] Noether, S.: Ring Signature Confidential Transactions for Monero. Cryptology ePrint Archive, Report 2015/1098. 2015. https://eprint.iacr.org/2015/1098.

[20] Poon, J.; Dryja, T.: *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. [Online; visited 16.12.2017].
Retrieved from: https://lightning.network/lightning-network-paper.pdf

[21] Quisquater, J.-J.; Guillou, L.; Annick, M.; et al.: How to Explain Zero-knowledge Protocols to Your Children. In *Proceedings on Advances in Cryptology*. CRYPTO '89. New York, NY, USA: Springer-Verlag New York, Inc.. 1989. ISBN 0-387-97317-6. pp. 628–631.
Retrieved from: http://dl.acm.org/citation.cfm?id=118209.118269

[22] Rizun, P.: *BUIP065: Gigablock Testnet Initiative*. [Online; visited 16.3.2018].
Retrieved from:
https://github.com/BitcoinUnlimited/BUIP/blob/master/065.mediawiki

[23] van Saberhagen, N.: *CryptoNote v 2.0*. [Online; visited 10.12.2017].
Retrieved from: https://cryptonote.org/whitepaper.pdf

[24] Werner, A.; Pliskov, M.: *CryptoNote Difficulty Adjustment*. [Online; visited 20.4.2018].
Retrieved from: https://cryptonote.org/cns/cns010.txt

Appendix A

The content of CD

- /doc/ - this thesis
  - src/ - source files for this thesis
  - doc.pdf - this thesis in pdf
- /src/ - source files for *Cryptoalarm*
  - README.md - installation instructions
  - cryptoalarm/ - source files for monitoring application
    - Makefile
    - Dockerfile - specification for docker container
    - config.json - configuration file
    - cryptoshell.py - cryptoshell application
    - requirements.txt - package dependencies
    - run.py - launcher
    - tests/ - test scripts
    - doc/ - documentation
    - cryptoalarm/ - modules of monitoring application
  - webapp/ - source files for web application
- poster.pdf - conference poster

Appendix B

RPC API responses

B.1 Bitcoin RPC API: block data structure

```
{
    "hash" : "hash",
    "confirmations" : n,
    "size" : n,
    "strippedsize" : n,
    "weight" : n,
    "height" : n,
    "version" : n,
    "versionHex" : "00000000",
    "merkleroot" : "xxxx",
    "tx" : [
      "transactionid",
      ...
    ],
    "time" : ttt,
    "mediantime" : ttt,
    "nonce" : n,
    "bits" : "1d00ffff",
    "difficulty" : x.xxx,
    "chainwork" : "xxxx",
    "previousblockhash" : "hash",
    "nextblockhash" : "hash"
}
```

Listing B.1: Bitcoin block

B.2 Bitcoin RPC API: transaction data structure

```
{
    "hex" : "data",
    "txid" : "id",
    "hash" : "id",
    "size" : n,
    "vsize" : n,
    "version" : n,
    "locktime" : ttt,
    "vin" : [
        {
            "txid": "id",
            "vout": n,
            "scriptSig": {
                "asm": "asm",
                "hex": "hex"
            },
            "sequence": n
            "txinwitness": ["hex", ...]
        },
        ...
    ],
    "vout" : [
        {
            "value" : x.xxx,
            "n" : n,
            "scriptPubKey" : {
                "asm" : "asm",
                "hex" : "hex",
                "reqSigs" : n,
                "type" : "pubkeyhash",
                "addresses" : [
                    "address",
                    ...
                ]
            }
        },
        ...
    ],
    "blockhash" : "hash",
    "confirmations" : n,
    "time" : ttt,
    "blocktime" : ttt
}
```

Listing B.2: Bitcoin transaction

B.3 Ethereum RPC API: block data structure

```
{
    "jsonrpc":"2.0",
    "id":83,
    "result":{
        "difficulty":"0xN",
        "extraData":"0xN",
        "gasLimit":"0xN",
        "gasUsed":"0xN",
        "hash":"hash",
        "logsBloom":"",
        "miner":"hash",
        "mixHash":"hash",
        "nonce":"0xN",
        "number":"0xN",
        "parentHash":"hash",
        "receiptsRoot":"hash",
        "sha3Uncles":"hash",
        "size":"0xN",
        "stateRoot":"0xN",
        "timestamp":"0xN",
        "totalDifficulty":"0xN",
        "transactions":[
            "hash",
            ...
        ],
        "transactionsRoot":"hash",
        "uncles":[
            "hash",
            ..
        ]
    }
}
```

Listing B.3: Ethereum block

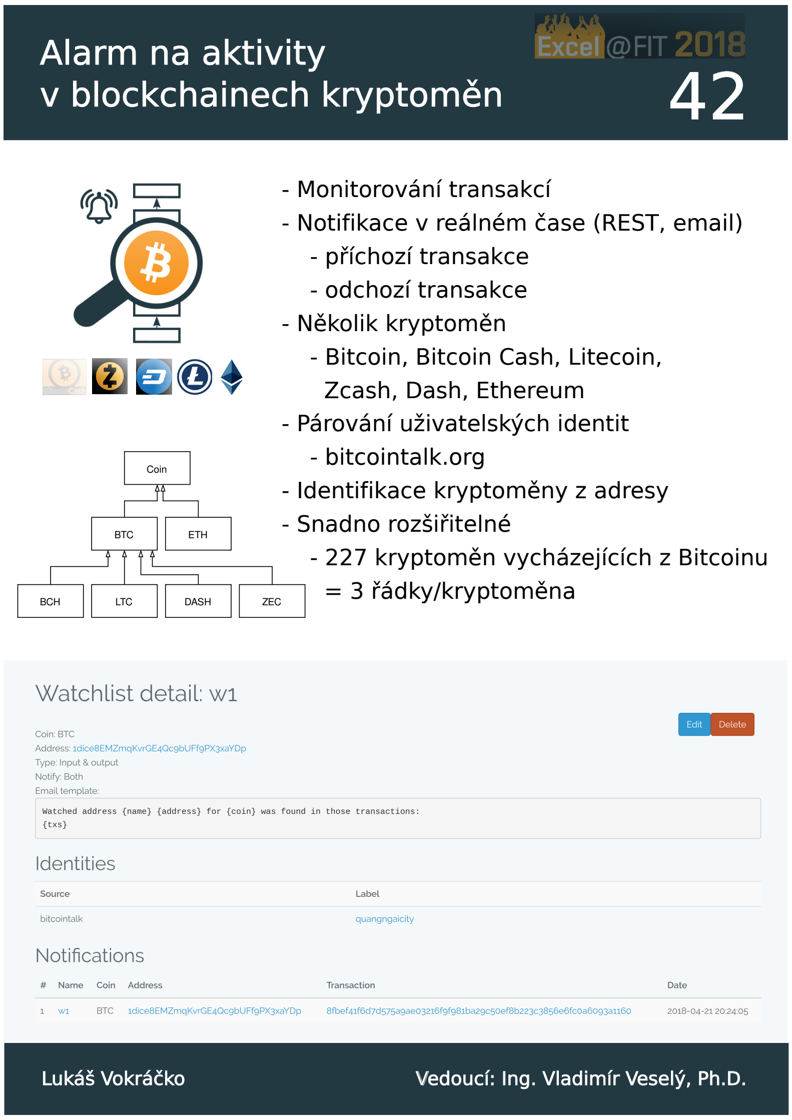
B.4 Ethereum RPC API: transaction data structure

```
{
    "jsonrpc":"2.0",
    "id":83,
    "result":{
        "blockHash":"hash",
        "blockNumber":"0x33266",
        "from":"hash",
        "gas":"0x15f90",
        "gasPrice":"0xba43b7400",
        "hash":"hash",
        "input":"0x",
        "nonce":"0x30d",
        "to":"address",
        "transactionIndex":"0xN",
        "value":"0xN",
        "v":"0x1c",
        "r":"hash",
        "s":"hash"
    }
}
```

Listing B.4: Ethereum transaction

Appendix C

Poster



Alarm na aktivity
v blockchainech kryptoměn

Excel@FIT 2018

42

- Monitorování transakcí
- Notifikace v reálném čase (REST, email)
    - příchozí transakce
    - odchozí transakce
- Několik kryptoměn
    - Bitcoin, Bitcoin Cash, Litecoin, Zcash, Dash, Ethereum
- Párování uživatelských identit
    - bitcointalk.org
- Identifikace kryptoměny z adresy
- Snadno rozšiřitelné
    - 227 kryptoměn vycházejících z Bitcoinu = 3 řádky/kryptoměna

Watchlist detail: w1

Coin: BTC
Address: 1dice8EMZmqKvrGE4Qc9bUFf9PX3xaYDp
Type: Input & output
Notify: Both
Email template:

```
Watched address {name} {address} for {coin} was found in those transactions:
{txs}
```

Identities

Source	Label
bitcointalk	quangngaicity

Notifications

#	Name	Coin	Address	Transaction	Date
1	w1	BTC	1dice8EMZmqKvrGE4Qc9bUFf9PX3xaYDp	8fbef41f6d7d575a9ae03216f9f981ba29c50ef8b223c3856e6fc0a6093a1160	2018-04-21 20:24:05

Lukáš Vokráčko

Vedoucí: Ing. Vladimír Veselý, Ph.D.

Figure C.1: Poster for conference Excel@FIT