Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра оптимального керування і економічної кібернетики

Дипломна робота

на здобуття ступеня вищої освіти «магістр»

на тему: **«Доведення з нульовим розголошенням в задачах анонімізації фінансових операцій»**

«Zero-knowledge proof in the tasks of anonymization of financial transactions»

Виконав: студент денної форми навчання
спеціальності 113 Прикладна математика
Прокопов Еммануїл Костянтинович

Керівник: канд. техн. наук., доц. Мазурок І. Є.
Рецензент: канд. техн. наук, проф. Мороз В. В.

Рекомендовано до захисту:

Протокол засідання кафедри

№ ___ від _____ 2021 р.

Завідувач кафедри

_____ _____

Захищено на засіданні ЕК № _____

Протокол № ___ від _____ 2021 р.

Оцінка _____ / _____ / _____

Голова ЕК

_____ _____

Одеса — 2021 р.

Odesa I. I. Mechnikov National University
Faculty of Mathematics, Physics and Information Technology
Department of optimal control and economical cybernetics

Diploma thesis

master

Zero-knowledge proof in the tasks of anonymization of financial transactions

Fulfilled by: full-time student
specialty 113 Applied Mathematics
Prokopov Emmanuil

Supervisor: Ph.D. of Engineering Sciences,
Assoc. Prof. Mazurok I. Y.
Reviewer: Ph.D. of Engineering Sciences, Full
Prof. Moroz V. V.

Odesa — 2021

CONTENTS

ВСТУП

У цій роботі розглядається використання доведень з нульовим розголошенням для побудови децентралізованих фінансових систем, що могли б забезпечити високий рівень анонімності фінансових транзакцій (приховування від третіх осіб відправника, одержувача та сумми транзакції).

Ця тема є актуальною, оскільки в останні роки спостерігається значне зростання інтересу до децентралізованих фінансових систем. Перші системи такого типу (Bitcoin, Ethereum) забезпечували користувачам високий, у порівнянні з традиційними фінансовими системами, рівень анонімності, оскільки адреса користувачів не ставилася у відповідність до їх ідентифікаційних даних. Однак зазначені системи зберігають усю історію транзакцій у публічному доступі. З одного боку, це забезпечує більшу прозорість. З іншого, у випадку, якщо третім особам будь-яким чином стануть відомі ідентифікаційні дані власника певної адреси у системі, їм також буде доступна повна історія транзакцій цієї особи. Це є небажаним, у зв'язку з цим виникає інтерес до аналогічних систем, які приховували б адресу відправника, одержувача та суму транзакції від сторонніх осіб. Такі системи (Grin, Firo) використовують докази з нульовим розголошенням, щоб зробити можливою перевірку коректності транзакції третіми особами, уникаючи при цьому розголошення зазначених даних.

Надалі в роботі будуть використовуватися наступні означення:

Фінансова система — сукупність фінансових операцій, які проводяться суб'єктами фінансової діяльності з використанням певного фінансового механізму. У ваипадку децентралізованої фінансовохї системи таким механізмом є протокол, за яким діє ця система.

Транзакція у децентралізованій фінансовій системі — це група послідовних операцій, що змінюють стан системи. Зазвичай вона містить посилання на попередні транзакції та асоціює певну кількість одиниць валюти з одним або кількома адресами користувачів системи.

Майнер — вузол системи, що займається валідацією нових транзакцій у системі, отримаючи за це певну винагороду.

Об'єктом дослідження даної роботи є використання доведень з нульовим розголошенням в задачах анонімізації фінансових транзакцій. Предметом дослідження є доведення з нульовим розголошення та протоколи децентралізованих фінансових мереж що їх використовують, можливість їх використання для анонімізації транзакцій в існуючих фінансових мережах. Метою роботи є дослідження способів та доцільності застосування доведень з нульовим розголошенням для побудови розподілених фінансових систем, у яких фінансові транзакції були б анонімізованими.

У ході виконання роботи був розроблений програмний додаток для моделювання роботи розглянутих у роботі протоколів. За темою роботи наявні дві публікації [1, 2]:

1) Prokpov E. K.: Proof of zero-knowledge in the tasks of anonymization of financial transactions. Стан, досягнення та перспективи інформаційних систем і технологій / Матеріали XXI Всеукраїнської науково-технічної конференції молодих вчених, аспі-рантів та студентів. Одеса, 22-23 квітня 2021 р. — Одеса, Видавництво ОНАХТ, 2021 р. — 229 с. — С. 51-52.
2) Prokpov E. K.: Proof of zero-knowledge in the tasks of anonymization of financial transactions. Current issues of science, prospects and challenges. Vol. 2 / Collection ofscientific papers «SCIENTIA» with Proceedings of the I International Sci-entific and Theoretical Conference. Sydney, December 17 2021. — EuropeanScientific Platform, 2021. — 111 p. — pp. 43-44.

INTRODUCTION

This paper considers the usage of zero-knowledge proofs in decentralized financial systems that could provide a high level of anonymity of financial transactions (concealment of the transaction sender, recipient, and amount from third parties).

This topic is relevant because of a significant increase in interest in decentralized financial systems in recent years. The first systems of this type (Bitcoin, Ethereum) provided users a high level of anonymity compared to traditional financial systems, as users' addresses in the systems were not matched to their personal information. However, these systems keep the entire history of transactions publicly available. On the one hand, this provides greater transparency. On the other hand, if third parties somehow become aware of the identity of the owner of a certain address in the system, they also have access to the full history of transactions of this person. This is undesirable, so there appeared interest in similar systems that would hide the addresses of the sender, recipient, and the amount of the transaction from third parties. Such systems (Grin, Firo) use zero-knowledge proofs to make it possible for the third parties to verify the correctness of the transaction while avoiding the disclosure of the listed data.

Further in this paper, the following definitions will be used:

A financial system is a system that allows the exchange of funds between participants of the system, using a certain financial mechanism. In the case of a decentralized financial system, the financial mechanism is the protocol over which this system operates.

A transaction in a decentralized financial system is a group of successive operations that change the state of the system. It usually contains references to previous transactions and associates a certain amount of funds with one or more system user addresses.

A miner is a system node that validates new transactions in the system and receives a certain reward for this.

The object of research of this work is the use of zero-knowledge proofs

in the problems of anonymization of financial transactions. The subject of the study is zero-knowledge proofs and protocols of decentralized financial networks that use them, the possibility of using zero-knowledge proofs to anonymize transactions in existing financial networks. The aim of the work is to study the ways and expediency of using zero-knowledge proofs to build distributed financial systems in which financial transactions would be anonymous.

During the research, a software application was developed to model the work of the protocols considered in the paper. There are two publications on the topic of the paper [1, 2]:

1) Prokpov E. K .: Proof of zero-knowledge in the tasks of anonymization of financial transactions. Status, achievements and prospects of information systems and technologies / Proceedings of the XXI All-Ukrainian scientific and technical conference of young scientists, aspirants and students. Odessa, April 22-23, 2021 — Odessa, ONAFT Publishing House, 2021 — 229 p. —pp. 51-52.
2) Prokpov E. K.: Proof of zero-knowledge in the tasks of anonymization of financial transactions. Current issues of science, prospects and challenges. Vol. 2 / Collection ofscientific papers «SCIENTIA» with Proceedings of the I International Sci-entific and Theoretical Conference. Sydney, December 17 2021. — EuropeanScientific Platform, 2021. — 111 p. — pp. 43-44.

CHAPTER 1

ZERO-KNOWLEDGE PROOFS

Zero-knowledge proofs are a class of cryptographic protocols that allow one agent (the prover) to prove to another agent (the verifier) that a specific statement about some data is true, without disclosing any additional information about this statement (without disclosing the data itself or the source from which the prover found out about the veracity of the statement) [3]. This condition is necessary since it is usually trivial to prove that an agent has certain information by simply disclosing it. Zero-knowledge proofs are useful in cases when the prover does not have the right to disclose information to the verifier and third parties.

The protocol should take into account that the prover must only be able to convince the verifier if the provided proof is actually valid. Otherwise, it must be impossible to do this, or extremely unlikely due to the high computational complexity. With the help of zero-knowledge proofs, the verifier can make sure that the prover does indeed have some specific information, while the information is not disclosed. This allows using these algorithms for proving some statements about secret information while using insecure communication channels, without the risk of third parties revealing the information [4].

In general, the concept of proof in zero-knowledge proofs is quite similar to the classical understanding of proof in mathematics. Two main agents participate in the protocol: the prover and the verifier. In classical mathematical proofs (proofs of lemmas, theorems, etc.), the personality of the prover is somewhat transcendental. At the same time, the identity of the verifier is quite clearly defined - this is a person who reads the text of the proof and performs all its stages to make sure that it is true. In the interactivity of agents in zero-knowledge proofs, these roles are much more evident. The fundamental assumption is that the verifier initially does not trust the prover. It is assumed that the prover may try to deceive the verifier. Otherwise, if the prover is trustworthy, it is inappropriate to require him to provide any proof.

Typically, a zero-knowledge proof is an interactive cryptographic protocol.

The verifier seeks to make sure that the prover has knowledge about some secret. The verifier sends a request to the prover to solve some mathematical problem. In response to this, the prover sends the verifier the solution of the problem, which he has obtained using the secret. This solution is considered the proof. After that, the verifier checks the correctness of the received solution. If it is correct, the verifier accepts the hypothesis that the prover knows the secret. It is required that the problem has high computational complexity so that it is only possible to quickly calculate the solution if the prover knows the secret (there is no polynomial-time algorithm that would allow finding a solution without knowing the secret). The verifier should be able to easily check the correctness of the obtained solution. Also, it should not be possible to extract knowledge about the secret from the received solution (in practice it is usually required that the problem of calculating the secret from the obtained solution is sufficiently computationally difficult).

There are also zero-knowledge proof protocols that do not require interactive input data. Such protocols tend to rely on the assumption of a perfect cryptographic hash function (which means that the output of a one-way hash function cannot be predicted if the function input is unknown) [5].

Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone provide the following definition for zero-knowledge proofs [6]:

Definition 1.1. A zero-knowledge proof is an interactive protocol that allows the prover to convince the verifier that some specific statement is true. The protocol must satisfy the following three properties:

1) Completeness: If the prover and verifier are honest, the prover is likely to convince the verifier with sufficient probability. The definition of sufficient probability depends on the specific algorithm and the application, but it usually means that the probability that the verifier will reject the provided proof under the specified conditions is negligible. This means that if the statement is actually true, then the prover will convince the verifier with any predetermined probability. Here we define an honest prover as a prover who will try to convince the verifier that the statement is true only if it is actually true. An honest verifier is a verifier who admits that the statement is true only if he is provided with valid proof.

2) Soundness: If the statement is false, no dishonest prover can convince an honest verifier that the statement is true. The opposite is possible, but it is negligible.
3) Zero-knowledge: the prover must not provide any additional information about the statement other than the fact that the statement is true or false to any (honest or dishonest) verifier. Also, it is important that there does not exist a polynomial algorithm that allows the verifier to obtain any additional information about the statement from the provided proof. In addition, if the procedure of proving truthfulness of the same statement between the same prover and verifier occurs several times (in several rounds), then the proof obtained by the verifier in the previous round must not increase his chances of extracting additional information about the statement from proofs obtained in the next rounds.

Remark 1.1. Consider an observer who has access to open communication channels that are used by the prover and the verifier during the execution of the protocol. The observer witnesses the request sent from the verifier to the prover, the proof that it sends in response, and the negative or positive verdict of the verifier. However, this does not give the observer any confidence in the truth or falsity of the statement, since the verifier and the prover could have agreed in advance about the list of tasks that the verifier sends. In addition, the observer cannot be sure whether the verifier and the prover are acting honestly. This is a very important property of interactive zero-knowledge proofs: as the result of execution of the protocol, only the actively participating verifier receives new information about the statement [6].

Remark 1.2. None of the above three properties, or any combination of them, does not guarantee that the protocol is secure. A necessary condition for the security of the protocol is the fulfillment of all three of these properties and the fact that the mathematical problem that is used for building the proof (the problem on which the protocol is built) is computationally hard. [6].

When compared to classical public-key cryptographic systems, the following observations on zero-knowledge proof protocols can be made [6]:

1) No degradation with usage: The security of protocols using zero-knowledge

proofs does not suffer degradation with repetitive use, and is resistant to text selection attacks. However, many public-key systems are vulnerable to this type of attack.

2) Many zero-knowledge protocols do not explicitly use data encryption, which can be useful in some situations (for example, when applied to decentralized financial systems, it can increase user trust in the system).
3) Efficiency: often protocols based on interactive zero-knowledge proofs, due to their nature, have relatively low computational efficiency and may create a relatively high load on communication channels.
4) Depending on the problem on which they are based, some zero-knowledge protocols may use unproven assumptions (like the P versus NP problem).

Illustrative examples

Below there are some illustrative examples of zero knowledge protocols.

Two balls and the colour-blind friend

Consider two people standing in the room: V. (the verifier) and P. (the prover). On the table between them are two balls, which differ only in color: one is red, the other one is green. V. is color blind and cannot distinguish one ball from another. P. can see the color difference and can distinguish them. V. doubts that the balls are distinguishable. At the same time, P. wants to prove this to V., without revealing which of the balls is red and which is green. The proof is carried out as follows: V. takes both balls in his hands and hides them behind his back. Then he shows one of the balls to P. Then V. puts it behind his back, and then again shows only one of the two balls, choosing one of them randomly with equal probability. He asks P. if the same ball was shown for the first time, or another one. P. can distinguish the balls and tells V. the correct answer. Now V. knows that P. probably actually can distinguish the balls. Or P. just was lucky to guess, with a probability of 50%. Then all actions can be repeated once more so that the probability that P. guessed the correct answer twice in a row is 25% [4].

This is an example of an interactive zero-knowledge proof. In this case,

the following statement is being proved: P. knows the colors of the balls, and, accordingly, can distinguish them. Each round of proofing consists of several steps:

1) V. Shows P. one of the balls.
2) V. randomly selects one of the balls, shows it to P, and asks P. to say whether the same ball was shown at the first time or not.
3) As proof, P. tells V. the correct answer.
4) If the answer is incorrect, V. does not believe P. If it is correct, V. believes that P. does indeed have the required information about the statement, or asks for another round.

So, after n rounds V. will make sure that P. knows the colors of the balls, with probability $1 - \frac{1}{2^n}$. In this case, P. will not receive any additional information about how he could distinguish the balls if they were mixed randomly. At the same time, P. can never be completely sure that D. is telling the truth, but can convince of this with an arbitrarily high probability. It is easy to notice that this algorithm satisfies the three properties from definition 1.1.

The strange cave of Ali Baba

This illustrative example was first published by Jean-Jacques Quisquater and others in their paper "How to Explain Zero-Knowledge Protocols to Your Children" [7].

In this example we consider a torus-shaped cave (figure 1.1):

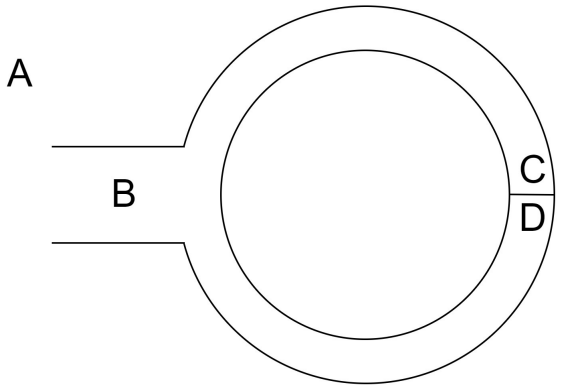

Figure 1.1. The cave scheme

The cave has an entrance at point B, and at its opposite end, between points C and D, there is a door, which can only be opened if one knows the secret password. The verifier (V.) wishes to make sure that the prover (P.) knows the password. At the same time, the prover does not want to disclose the password to the verifier. Then they act as follows:

1) V. and P. Move to the entrance to the cave, to the point A. V. waits at point A, while P. enters the cave and goes to one of the points C or D.
2) After some time, sufficient for P. to hide out of sight, V. moves to point B and shouts out loud from which side P. should go out. Suppose V. wants P. to come out from the right. Then, if P. is at point D, he just exits the cave. If he is at point C, he uses the password to open the door, moves to point D, and exits from the required side.
3) Seeing that P. has exited from the correct side, V. understands that either P. has the password, or he has guessed at which point he would be asked to show.

The chance that P. just guessed from which side he would be asked to show is 50%. Thus, repeating the procedure n times, P. can convince V. that he knows the password with probability $1 - \frac{1}{2^n}$. Obviously, this algorithm satisfies the properties of completeness, soundness and zero-knowledge.

This example illustrates Remark 1.1 well: if a third-party observer is watching the procedure, he cannot be sure that P. indeed knows the password, since P. and V. could have agreed in advance to deceive the observer. The only way for an observer to make sure that P. knows the password is to participate in the protocol himself in the role of the verifier.

Yao's Millionaires' Problem

Another illustrative example, related to economics in some way, is a discrete version of Yao's millionaires' problem.

The original problem has the following statement [8]. Consider Alice has a secret number a, and Bob has a secret number b. The goal of Alice and Bob is to find out whether the inequality $a \leq b$ is true, without revealing to each other or any third parties the values of a and b, or any other information about these numbers.

This problem is called so because, in the original setting, Alice and Bob are two millionaires who want to find out which of them is wealthier, without revealing the amounts of money they have. We will take a look at a discrete version of this problem [4].

Consider Alice has a secret number a, and Bob has a secret number b such that a and b are integers in set $\{1, 2, ...n\}$, where n is a natural number. They want to check if statement $a = b$ is true, without revealing to each other or third parties the values of these numbers or any other information about them. To solve this problem, the following procedure is proposed:

1) Bob enters the room alone, takes n lockable boxes, and marks each of them with a unique number from the set $\{1,2, ...n\}$.
2) Bob leaves the room and throws away all the keys to the boxes except one corresponding to the box marked with b. Alice makes sure that Bob has only one key left.
3) Alice enters the room. Alice slips a small piece of paper marked with the text "Yes" into a box marked with number a. In the rest of the boxes, she slips identical pieces of paper labeled "No". Alice leaves the room.
4) Bob enters the room. He opens the box marked with b, takes out a piece of paper, and leaves it on the table. He erases the marks from all the boxes and removes them.
5) Alice enters the room and also checks what is written on the piece of paper.

Now, if the piece of paper says "Yes," both Alice and Bob know that $a = b$. Otherwise, they know that $a \neq b$. An interesting thing about this example is

that in this case both Alice and Bob act as both the verifier and the prover. The algorithm satisfies the third property from definition 1.1 (zero-knowledge), since neither Alice nor Bob acquire any knowledge about the numbers a and b except for the correctness of the provided statement. On the other hand, it does not satisfy the second property, since both Alice and Bob can successfully cheat. Alice can slip a piece of paper labeled "Yes" into the wrong box. Bob can keep the key to another box, not the one marked with b.

Examples of zero-knowledge proofs used in practice

Fiat-Shamir identification protocol

Fiat-Shamir protocol is a well-known and widely used zero-knowledge protocol [6]. It is an identification protocol in which the user proves his identity by proving that he knows the value of the square root of some number t, which serves as the user's public key. The protocol is based on the fact that the task of calculating the square root of a number by some modulo is computationally hard.

The setup phase consists of several steps:

1) A trusted third party chooses two large primes p and q, computes their product $n = pq$, and communicates the value n to the users.
2) Users, using the obtained value of n, generate their public keys. Each user chooses a random number s such that $1 \leq s \leq n - 1$. s now is the user's private key. Next, each user calculates the value of his public key t: $t = s^2 \mod n$.

There also exist modifications of this algorithm that do not require a trusted third party for the setup phase [6].

The identification itself is performed in several rounds. Each round has the following structure:

1) The prover chooses a random number v such that $1 \leq v \leq n-1$, calculates the value $q = v^2 \mod n$, called the fixer, and sends it to the verifier.
2) The verifier sends the prover a random bit $r \in \{0, 1\}$.
3) The prover calculates the value $x = vs^r \mod n$ and sends it back to the

verifier.

The verifier considers the result of the round to be positive if the equality $x^2 = qt^r \mod n$ is satisfied. It is easy to show by substitution that if the prover acts honestly, these expressions are equal:

$$\begin{gathered} x^2 \mod n = qt^r \mod n \\ (vs^r)^2 \mod n = v^2 t^r \mod n \\ v^2 s^{2r} \mod n = v^2 s^{2r} \mod n \end{gathered}$$

The probability that a dishonest prover (who does not know the secret key s) in one round can pass the check is $\frac{1}{2}$ [6]. Therefore, after z rounds, the probability that a cheater could be mistaken for a user knowing the secret s is $\frac{1}{2^z}$. By setting the required number of rounds sufficiently large, it is possible to reduce the probability of cheating to any required value.

An important remark is that that the randomness of the verifier's request generated at the second step is a fundamental part of the protocol. If the request is known in advance, a potential dishonest prover can easily mislead the verifier. In each round, the dishonest prover has two possible patterns of behavior. If he expects a request to be $r = 0$, he chooses a random number v and passes the value $q = v^2 \mod n$ to the verifier. If he receives a request $r = 0$ from the verifier, then the prover sends the correct answer $x = v$. However, he cannot correctly answer if the request is $r = 1$. In the opposite case, when the expected request is $r = 1$, the dishonest prover chooses a random number v and sends the number $q = \frac{v^2}{t} \mod n$ to the verifier. If he receives a request $r = 1$ from the verifier, he sends the answer $x = v$, which will be accepted by the verifier as correct, since $qt \mod n = \frac{v^2}{t} t \mod n = v^2 \mod n = x^2 \mod n$. However, the prover will not give the correct answer to the request $r = 0$.

There is a simple modification of the protocol that allows reducing the required number of rounds to one, thereby significantly reducing the requirements for communication channels [9]. It relies on a version of the protocol that exploits the difficulty of the task of calculation of a root of a large simple degree by a simple modulus to avoid a trusted third party in the setup phase.

For public keys generation, numbers of the structure $t = s^k \mod p$ are used, where p is a prime number of a special structure: $p = Nk^2 + 1$, where k

is a prime number. Unlike the iterative protocol, this version assumes that each user has h values $t_i = s_i^k$ as a public key, where s_i are secret values, $i = 1, 2, ..., h$. This allows h one-bit requests of the iterative protocol to be combined into a single h-bit request E, and the identification process, which now takes only one round, consists of the following steps:

1) The prover chooses a random number v such that $1 \leq v \leq p-1$, calculates the value of the fixer $q = v^k \mod p$, and sends it to the verifier.
2) The verifier generates a random h-bit number $E = (e_1, e_2, ..., e_h)$ and sends it to the prover.
3) The prover calculates the value $W = v \prod_{i=1}^{h} x_i^{e_i} \mod p$ and sends it as a response to the received request.

The verifier considers the answer correct if the relation $W^k = q \prod_{i=1}^{h} t_i^{e_i} \mod p$ is true. The probability of deception is $\frac{1}{2^h}$, which is determined by the following actions of the dishonest prover trying to impersonate the owner of the open key $(t_1, t_2, ..., t_h)$. The prover generates a random request $E = (e_1, e_2, ..., e_h)$ and a random response W, and then calculates the value of $q = W^k \prod_{i=1}^{h} t_i^{-e_i} \mod p$. Then, at the first step of the protocol, he sends the received fixer value to the verifier, expecting to receive a request $E = E$, which is an event with probability $\frac{1}{2^h}$. When such an event occurs, the dishonest prover successfully passes the authentication procedure.

CHAPTER 2

USAGE OF ZERO-KNOWLEDGE PROOFS IN DISTRIBUTED FINANCIAL SYSTEMS

Distributed financial systems built using zero-knowledge proofs are of great interest to the public. They offer a significantly higher level of anonymity by hiding a lot of transaction details such as the sender and recipient addresses or the amount of the transaction. Such systems have another strong advantage over classical distributed financial systems, which store the entire transaction history publicly available. Some of them do not require storing the complete history of transactions !!!, but only its actual part. This leads to a significant reduction of the requirements to the memory size of the network node and increases the scalability of the system.

On the other hand, this is also the source of the main disadvantage of this kind of systems. Without a complete history of transactions, the user can make sure of the correctness of the system only by understanding the principle of its operation. This raises a problem since usually the protocols used in practice are quite complex. And often, the more reliability the protocol can provide, the more complex it is. This can lead to high distrust of a significant part of potential users to the system because of inability to understand the "moon math", which greatly increases the entry threshold. Furthermore, the simpler the system is, the easier it is to maintain and perform the audit. It is highly important since many projects using zero-knowledge proofs are open-source and community-driven.

Mimblewimble protocol

Mimblewimble is a blockchain format and protocol using zero-knowledge proofs. It provides relatively high scalability, privacy, and anonymity of transactions. One of the implementations of this protocol is the open-source project Grin [10].

To ensure a high level of transaction anonymity, Mimblewimble uses

elliptic curve cryptography. In this paper, we will not discuss cryptography on elliptic curves in detail, but below we will provide a description of their basic properties.

An elliptic curve defined over a certain field is a set of points [6]. Consider an elliptic curve C. For the points of the elliptic curve, the operations of addition and scalar multiplication are defined in a specific way. Consider an integer number k. Then, using scalar multiplication, it is possible to calculate $k * H$, and the product is also a point on the curve C. Elliptic curves form an abelian group with respect to the addition operation. Consider another scalar j, then it is also possible to calculate $(k + j) * H$. Addition and scalar multiplication on the Elliptic Curve satisfies the commutative and associative properties of addition and multiplication: $(k + j) * H = k * H + j * H$.

In elliptic curve cryptography, if a large scalar value of k is considered as the private key, then the product $k*H$, where H is the point of the elliptic curve, can be used as the corresponding public key. Although multiplication by a scalar in elliptic curves is trivial, division by a point of a curve is computationally difficult. Thus, even if someone knows the value of the public key $k * H$, calculating k is close to impossible.

The previous formula $(k + j) * H = k * H + j * H$, where k and j are considered to be secret private keys, shows that it is possible to obtain a public key by adding two private keys and this operation is identical to adding two corresponding public keys.

The Mimblewimble protocol achieves a high level of anonymity using the specific transaction structure. The verification of transactions relies on two basic properties:

1) Zero sums verification. The sum of transaction's inputs must be equal to the sum of transaction outputs. This proofs that transaction does not create new tokens, without revealing the actual input and output amounts.
2) Private key ownership. Like in many other distributed financial systems, the user's ownership of specific transactions outputs is guaranteed by the possession of elliptic curves private keys. However, the proof that the user owns specific private keys is not achieved by simply signing the

transaction.

Elliptic curves properties allow obscuring the values in a transaction. Consider v is an input or output value of a transaction, and H is an elliptic curve. Then instead of directly adding v into the transaction's data, v is substituted with $v * H$. This is possible because it is still possible to ensure that transaction outputs and inputs sum equals zero using the properties of the elliptic curve. Consider a transaction with two inputs v_1 and v_2 and one output v_3. Then $v_1 + v_2 = v_3 \implies v_1 * H + v_2 * H = v_3 * H$. This allows ensuring that the transaction does not create new tokens without revealing its inputs and outputs.

In practice, the set of possible values of transaction inputs or outputs is finite and quite small, so it would be possible to reveal the amount v using brute force. Moreover, knowledge of the value of a specific transaction input or output v_1 reveals output values of all further transactions that use v_1 [10]. To avoid this, consider another elliptic curve G, which is another generator of the group, generated by H. Also consider a private key r used as a blinding factor. Thus, input and output values of a transaction are encoded as $r * G + v * H$, where:

1) r is a private key used as a blinding factor. G is an elliptic curve. $r * G$ is a public key on the curve G.
2) v is the value of an input or an output of a transaction. H is another elliptic curve.

Due to elliptic curves properties, it is difficult to compute r and v. As an example, consider creating of a transaction with two inputs v_1 and v_2 and one output v_3. If it is a valid transaction, then $v_1 + v_2 = v_3$. Following the described procedure, a blinding factor is generated for every input and output. So the new equation is $(r_1 * G + v_1 * H) + (r_2 * G + v_2 * H) = (r_3 * G + v_3 * H)$, which implies that $r_1 + r_2 = r_3$. So, transaction amounts can be validated without revealing their amounts.

Another idea introduced by Mimblewimble is that the same private key r that is used for blinding the transaction value can be used to prove ownership of that value.

Consider Alice wants to send Bob v_1 tokens. To blind this amount, Bob chooses r_1 as a blinding factor. Then somewhere on the blockchain the following

unspent transaction output that should only be spendable by Bob appears: $X = r_1 * G + v_1 * H$. The addition result X is a public value, value of v_1 is known by Alice and Bob, the r_1 value is known only by Bob. To spend these v_1 tokens, the protocol requires the user to know the secret key r_1. Consider Bob now wants to transfer these v_1 tokens to Carol. Then he should create a transaction such that $X_i \implies Y$, where X_i is a transaction input that spends the output X of the previous transaction, and Y is a transaction output for Carol. There is no way to create such a transaction without knowing the secret key r_1. Of course, if Carol accepts the transaction, she has to know the value of the private key r_1 and the value v_1. The resulting transaction sum equals to zero: $Y - X_i = (r_1 * G + v_1 * H) - (r_1 * G + v_1 * H) = 0 * G + 0 * H$.

This is unacceptable because in this case the same private key r_1 must be used by Carol to spend the output of this transaction, but it is also known to Bob, so this output is spendable by Bob too. So, Carol generates another private key r_2. Then the following equation would be written in the transaction:

$$Y - X_i = (r_2 * G + v_1 * H) - (r_1 * G + v_1 * H) = (r_2 - r_1) * G + 0 * H.$$

The sum of this transaction is no longer zero, but it is still possible to validate it. There is an extra value on the curve G $(r_2 - r_1)$ which equals the sum of all blinding factors. Note that $(r_2 - r_1)$ is a valid public key for the generator point G [10]. For any scalars x and y the sum $x * G + y * H$ is a valid public key on G only if y equals zero. Therefore, the protocol needs to verify that the transacting parties can collectively produce the private key $(r_2 - r_1)$ for the resulting point $Y - X_i$. It is done in a simple way by signing the transaction with the extra value $(r_2 - r_1)$ and verifying the signature using the public key $Y - X_i$. This ensures that:

1) Transaction participants collectively know the private key, so the sender had the right to send the tokens.
2) Transaction does not create new tokens.

This signature must be checked by the validators to ensure that the transaction is valid.

In all the described formulas it is considered that the values inputs and outputs are always non-negative. If negative values would be allowed this would

allow the creation of new tokens in a transaction. For example, consider a transaction with one input with value 5 and two outputs with values 7 and -2. This still would be a valid transaction, according to the formulas. This case would be hard to detect as transaction values are blinded and point -2*H is indistinguishable from any other. The same problem would be caused by overflowing values.

To solve this problem, Mimblewimble uses range proofs [11]. It is a non-interactive zero-knowledge protocol that is used for generating the proof that some specific value belongs to a specific range (in the case of Mimblewimble the considered range is from 0 to the maximum integer value that does not overflow). Such proof can be verified by any third party without revealing the value. But range proofs are not the only way to solve this problem.

zk-SNARK protocols

General description

Another family of general-purpose non-interactive zero-knowledge proof protocols that attracts a lot of interest in recent years is zk-SNARKs (zero knowledge succinct arguments of knowledge) [12]. Zk-SNARKs are used to generate compact proofs that some computation (that often can be hard) has some particular output. These proofs can be verified quickly, while the underlying computation may be complex. This family of protocols is often used in distributed financial systems (e. g. Zcash) for increasing system scalability and anonymization of specific transactions details. Two main features of zk-SNARKs (when applied to distributed financial systems) are:

1) Scalability: in such systems verification of a newly created transaction is often a problem as this may require heavy computations. This gets even worse in systems that provide some transaction anonymization as they often use complex cryptography. Interactive protocols are not appliable in this case as exchanging messages between the verifier and the prover would create a heavy load on the communication channels. Usage of non-interactive protocols allows the prover to calculate the proof once and attach it to a transaction, without receiving any initial requests from the

verifier. After that, any verifier may check if the proof is valid.

2) Zero-knowledge: the protocols allow proving some statements about a value without revealing it.

This kind of protocol may be used for building succinct proofs for results of very heavy computations. This is useful if the computation should be verified often by a big number of verifiers, and the computations are too complicated to repeat every time. The proof is considered "succinct" if the proof size and time required for its verification asymptotically grow significantly slower than the size of computations for which it is built [13].

When building proof for a large computation, the verifier must not check each step of the computation, but he still should be able to check if the prover is honest. If the computation consists of multiple repetitive steps, the natural idea is random sampling. This idea is quite similar to one described in the interactive protocols that are discussed in the previous chapter. The verifier may check the intermediate computation result on several randomly chosen steps. The prover does not know the number of steps in advance, so if a set of required intermediate results is big enough, the probability that the prover is dishonest is negligible.

Considering that the computation consists of similar repetitive blocks, it is easy to build a non-interactive version of an interactive zero-knowledge proof protocol using the Fiat-Shamir heuristic [12]. Zk-SNARKs use Merkle trees for applying this heuristic.

A Merkle tree is a binary tree, such that its leaves contain hashes of data blocks, and its inner vertices contain hashes of combinations of values in the child vertices. The root element of the tree contains the hash of the whole data set, so Merkle three may be considered as a one-way hash function. It makes getting the hash of a combination of specific values easy and may be used for verification of the values. In general, the three may use any hash function. When a new data block is added, its hash is put into a new leaf node, and all the parent node's values are updated one by one.

The basic idea of the Fiat-Shamir heuristic is quite simple. In interactive proof protocols, random requests to the prover are generated by the verifier. To make the protocol non-interactive, the prover performs the computation and

calculates its Merkle tree. After that, he uses the root of the tree to pseudo-randomly generate a set of random indexes and provides the Merkle branches with corresponding indexes. The key idea is that the prover does not know in advance which branches he will have to disclose before the computation is completed. If the dishonest prover tries to fake part of the data after he knows the indexes, this changes the Merkle root, so the set of indexes also changes.

Another problem is that if the prover makes a mistake at some point in the computation, it is difficult to detect that during the verification. Using polynomials for encoding the data is the solution used in zk-SNARKs [14].

A polynomial of a single indeterminate is an algebraic expression form $P(x) = \sum_{i=0}^{n} c_i x^i$. A useful property of polynomials is that they contain information about a huge set of values, which is received by evaluating it for integer argument values. For example, if the equation $A(x)+B(x) = C(x)$, where $A(x)$, $B(x)$ and $C(x)$ are polynomials, is true , it is true for any x. Consider two large sets of values that should be checked for equality. Brute force comparison would require linear time with respect to the sets size. It is possible to build a Lagrange polynomial for each set and compare the coefficients of these polynomials.

It is also possible to check relations between a large number of adjacent evaluations of the same polynomial. This is achieved using the first corollary of the polynomial remainder theorem [14]. Consider a polynomial $P(x)$ that evaluates to zero for elements of some set $S = x_1, x_2, \ldots, x_k$. Then $P(x)$ can be expressed as $P(x) = Z(x)*H(x)$, where $Z(x) = (x-x_1)*(x-x_2)*\ldots*(x-x_k)$, and $H(x)$ is a polynomial. If a polynomial equals zero across some set, it is a multiple of the lowest-degree polynomial that equals zero across the same set.

A simple example is checking if a polynomial encodes the Fibonacci numbers [14]. Consider a polynomial $F(x)$. The task is to check if $F(x+2) = F(x) + F(x+1)$ for $x \in \{0,1,\ldots,98\}$. Usually this would require a hundred polynomial evaluations and comparisons, this may be a lot of computation. But it is also possible to prove that F actually satisfies the required condition by proving that the polynomial $P(x) = F(x+2) - F(x+1) - F(x)$ is zero over that range by providing the quotient $H(x) = \frac{F(x+2)-F(x+1)-F(x)}{Z(x)}$, where $Z(x) = (x-0)*(x-1)*\ldots*(x-98)$. It is easy for the verifier to compute the polynomial $Z(x)$ and check the equation. Thus, a computation that would

require a hundred polynomial evaluations is reduced to simple polynomial equation.

Polynomial commitments

Another problem is that comparing large polynomials coefficient by coefficient may be computationally hard. Another technique allows reducing the comparison time: the polynomial commitments

. A polynomial commitment is a specific way to hash the polynomials so that the size of produced hashes is less than the size of the polynomial, and there are some specific operations defined for hashes, that allow comparing the polynomials by comparing their hashes. We will denote "commitment of polynomial $C(x)$" as $com(C(x))$, or $com(C)$. The common operations defined for commitments are:

1) Addition: given $com(P)$, $com(Q)$, and $com(R)$ it is possible to check if $P + Q = R$.
2) Multiplication: given $com(P)$, $com(Q)$, and $com(R)$ it is possible to check if $P * Q = R$.
3) Evaluation: given $com(P)$, scalars w, z and a supplemental proof Q, it is possible to check if P(w) = z.

These three operations may be constructed from each other. Constructing multiplication from addition is trivial. Evaluation is more complex. To prove that $P(w) = z$ the prover can construct $Q(x) = \frac{P(x)-z}{x-w}$. Then the verifier can check if $Q(x) * (x - w) + z = P(x)$. If such a polynomial $Q(x)$ exists, then $P(x) - z = Q(x) * (x - w)$ which means that $P(x) - z$ equals zero at w as $x - w$ at w.

Construction of addition and multiplication from the evaluation is possible using the Schwartz-Zippel lemma [14]. It states that if some polynomial equations hold true at a randomly selected coordinate, then it almost certainly holds for all the other coordinates. For example, an interactive protocol for proving the equation $P(x+2) - P(x+1) - P(x) = Z(x) * H(x)$ from the previous example could consist of the following steps:

1) The prover sends the verifier the commitments of the polynomials $com(P)$ and $com(H)$.

2) The verifier randomly selects an x-coordinate value r.
3) The prover sends the verifier $z_0 = P(r)$, $z_1 = P(r+1)$, $z_2 = P(r_2)$, $z_H = H(r)$ and the corresponding proofs Q_0, Q_1, Q_2, Q_H.
4) The verifier verifies the proofs, calculates $z_z = Z(r)$ and checks that $z_2 - z_1 - z_0 = z_h * z_z$.

This may be easily converted to a non-interactive protocol using the Fiat-Shamir heuristic: the prover would use some hash function to calculate $r = hash(com(P), com(H))$.

Two systems wildly used in practice for building polynomial commitments are bulletproofs [11] and FRI. In practice, the FRI implementations use a lot of complex optimizations, but the basic scheme [14] is quite simple.

Consider a polynomial $P(x)$ with degree less then n. In FRI the commitment to P is a Merkle root of a set of evaluations to this polynomial at some pre-selected set of coordinates. As size of the set of coordinates may be bigger than n, it is necessary to prove that the set of evaluations was produced by a degree $< n$ polynomial, because a dishonest prover could simply build a Lagrange polynomial.

Consider $Q(x)$ is the polynomial only containing the even coefficients of P, and the polynomial R only contains the odd ones. So that $P(x) = Q(x^2) + x * R(x^2)$. Note that the highest possible degree of Q and R is $\frac{n}{2}$. The prover is asked to generate Merkle roots for $Q(x)$ and $R(x)$. Then a random or pseudo-random number r is generated and a prover is asked to provide a linear combination $S(x) = Q(x) + r * R(x)$. Then a relatively big set of pseudo-random coordinates is generated and the prover is asked to provide the branches of the Merkle tree for P, Q, R, and S at this coordinates. At each of the coordinates x_i the prover checks that $P(x_i) = Q(x_i^2) + x_i * R(x_i^2)$ and $S(x_i) = Q(x_i^2) + r * R(x_i^2)$.

Note that as Q and R both have degrees $< n$, the same is true for S as their linear combination. And the same works in reverse: this guarantees that the degree of P is less than n as $P(x) = Q(x^2) + x * R(x^2)$. This prevents the prover from choosing malicious Q and R with hidden high-degree coefficients. The described process may be done repeatedly, halving the maximum possible polynomial degree on each round. So it would take around $log_2(n)$ to make the

degree small enough to be checked manually.

The complete description of the protocol for generation an FRI commitment:

1) Calculate Merkle roots for P, Q and R.
2) Pseudo-randomly select r depending on the values obtained in the previous step.
3) Calculate Merkle root for S.
4) Randomly select a sufficient number k of random coordinates
5) Provide Merkle branches at selected coordinates for P, Q, R and S, so that the verifier can check that $P(x) = Q(x^2) + x * R(x^2)$ and S(x) = Q(x) + r*R(x) at all the provided coordinates.
6) Set S as new P so that the degree is reduced by 2.
7) Repeat these steps until the polynomial degree is small enough to be checked manually.

Thus a full "FRI commitment" in this simplified version of the protocol consists of:

1) The Merkle root of evaluations of P, Q, R, and S_1 obtained on the first round.
2) The pseudo-randomly selected branches of P, Q, R and S_1 to check that S_1 is correctly generated from P.
3) The Merkle roots and randomly selected branches at the following k rounds.
4) The full Merkle tree of the evaluations of S_k so it can be checked directly.

Privacy

The problem of the described algorithm is that in some tasks it is necessary to keep values of the polynomial at some specific in secret. Providing evaluations of the polynomial $P(x)$ may help the validator to recover the value of the polynomial at another specific coordinate. In general, this is unlikely as the proofs are succinct, so often they will not be big enough to leak big parts of secret information. But there is a way to neglect this probability [14].

The idea is to add some extra factors to the polynomials before calculating

their Merkle roots. Considering the polynomial $P(x)$ from the previous examples, it is possible to add a small multiple of $Z(x)$ to it to obtain $P'(x) = P(x) + Z(x) * E(x)$ for some random $E(x)$. This obscures the data but does not change the polynomial evaluations at the coordinates where "the computation is happening" as $Z(x)$ evaluates to zero at these coordinates, but this adds enough extra "noise" into commitments to make the remaining information unrecoverable.

Using zk-SNARKs for range proofs

The statement of the task in terms of zk-SNARKs is the following. Consider a polynomial $P(x)$ that evaluates at some point n to a specific secret value $P(n)$. Prove that for some scalar ub such that $ub > 0$: $0 \leq P(n) < 2^{ub+1}$ without revealing the exact value of $P(n)$. This is the common case in distributed financial systems when it is necessary to prove that the balance is non-zero and does not overflow without revealing its value [13].

The proof can be constructed using the following polynomial equations [14]:

1) $P(0) = 0$
2) $P(x + 1) = P(x) * 2 + R(x)$ for integer values of x across the range $\{0 \dots ub\}$
3) $R(x) \in \{0,1\}$ across the same range

The last two equations can be written in different form using the polynomial

$$Z(x) = (x - 0) * (x - 1) * \dots * (x - ub).$$

It evaluates to zero for all values from the set $\{0 \dots ub\}$.

1) $P(x + 1) - P(x) * 2 - R(x) = Z(x) * H_1(x)$
2) $R(x) * (1 - R(x)) = Z(x) * H_2(x)$. This is identical to the previous form of the equation as for any scalar value k the equation $k * (1 - k) = 0$ is true only if $k \in \{0,1\}$.

The idea is that evaluations of $P(i)$ build up the number bit-by-bit. In this way, any number in the range $\{0, 2^{ub}\}$ can be built in at most ub steps, and any numbers outside this range can not. For example, if $P(3) = 7$, then the

sequence of evaluations $P(0), P(1), P(2), P(3)$ would be $0, 1, 3, 7$, in binary format: $0, 1, 11, 111$.

CHAPTER 3

ZK-SNARK RANGE PROOFS

Computational experiment description

Previous chapters have discussed the Mimblewimble protocol and its weak spots. The main problem is the fact that the protocol relies on range proofs. Typically, decentralized financial systems are open source projects supported by the community. Therefore, it is very important to keep the system simple. Mimblewimble deals with bulletproofs, which are very difficult to understand even for users with some mathematical background. In the previous chapter, the zk-SNARK protocol was described along with a way to use it to prove that a value is in a specified interval without revealing that number. This is achieved by constructing a special type of polynomial and evaluating it on pseudo-randomly generated sets of coordinates. The proposal is to modify the Mimblewimble protocol to use zk-SNARK-s instead of bulletproofs so that the system becomes more transparent and easy for maintenance and auditing.

During the work, a software application was developed to simulate the operation of the proposed Mimblewimble modernization. The computational experiment was performed to simulate the work of a naive implementation of the algorithm proposed in the previous chapter. The proof computation process was modeled with different values of the input parameters (range width and the target value). The main purpose of the experiment was a heuristic checking the dependency of computation time and the resulting proof size of the input values. In order to reduce the influence of random factors on the evaluation of the performance of the algorithm, the calculation of the proof was performed multiple times for the same set of parameters, and then the arithmetic mean of the obtained values was calculated.

Analysis of the obtained results

Two main variables that may affect the algorithm performance are the upper bound ub and the target value $P(n)$. Code of the application used for

modeling the algorithm functioning may be found in the appendix A.

The computational experiments have shown that ub and $P(n)$ almost do not affect the proof size. This is an expected behaviour as the proof size mostly depends on number of rounds, which depends on the degree of the polynomial $P(x)$. A single polynomial may encode large number of points and the degree of the polynomial grows slowly with P(n) growth. On the other hand, as the proof contains large number of randomly sampled Merkle branches, it grows approximately linearly depending on the required number of samples. But this is not critical as in practice this value usually stays relatively small (less then 10000) [14].

The computational experiments for checking the algorithm performance were performed for the parameters ub and $P(n)$. During each of experiments the value of other parameter was fixed. The results of experiments for ranges of different width are represented on the figure 3.2.1.

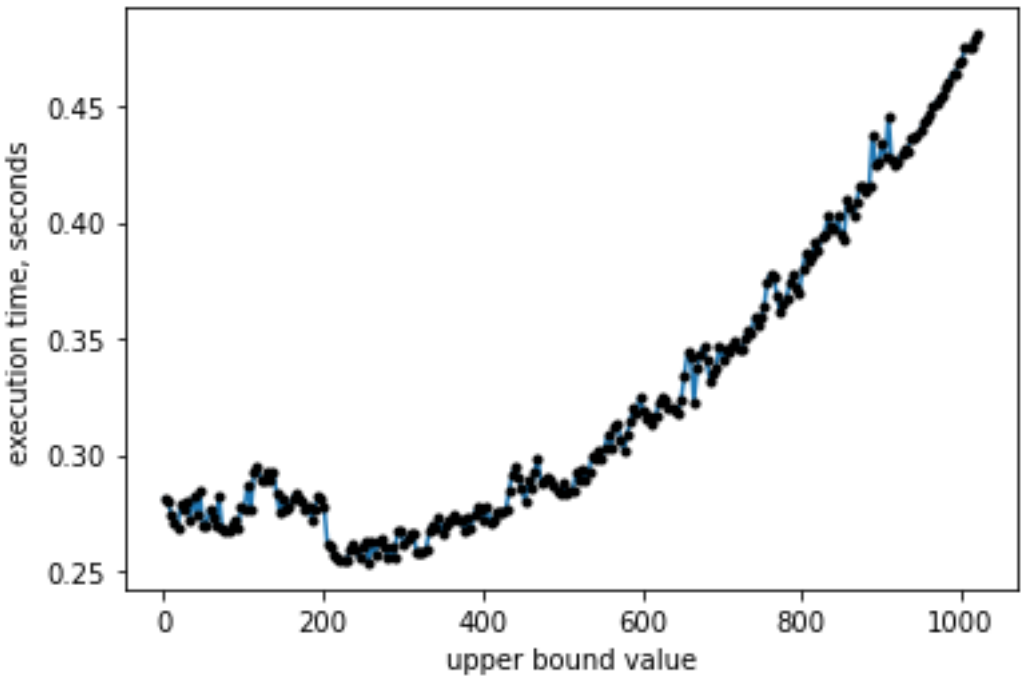

Figure 3.2.1 Results of computational experiments for ranges of different width.

The results show that the computation time grows rather quickly with an increase in the size of the required interval, while still remaining within acceptable limits.

The results of experiments for different target values are represented on the figure 3.2.2.

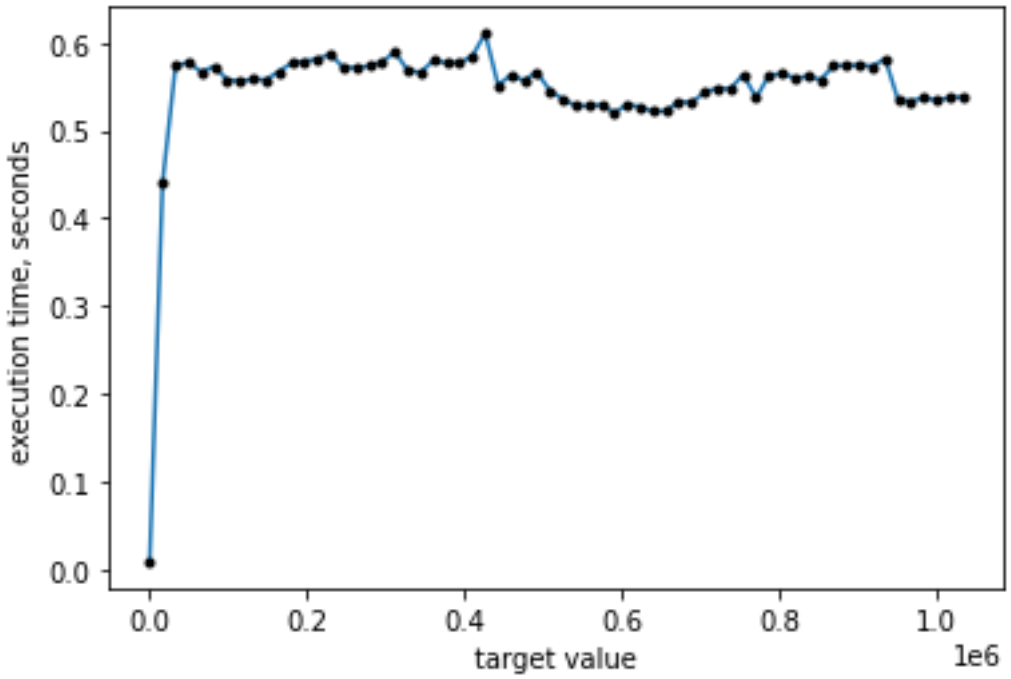

Figure 3.2.2 Results of computational experiments for different target values.

The results show that computation time is acceptable even for large target values. But $P(x)$ is a Lagrange polynomial, and number of points on which it is built depends on length of the binary representation of the target value, so for small target values the computation is performed faster.

The algorithm performance may be significantly increased using different optimizations, during the computational experiment the naive implementation was used.

Thus, the computational experiment showed that the solution is relatively scalable, proof computation takes acceptable amount of time (considering that in some popular distributed financial systems transaction verification may take up to ten minutes) [10]. The proposed modification could be used in practice.

ВИСНОВКИ

У ході виконання роботи було розглянуто докази з нульовим розголошенням та можливість їх застосування для анонімізації фінансових транзакцій у розподілених фінансових системах. Були розглянуті різні ілюстративні приклади та протоколи, що використовуються на практиці. Було розглянуто протокол розподіленої фінансової системи Mimblewimble. Були розглянуті його сильні та слабкі сторони. Була зроблена пропозиція модифікації цього протоколу. Було розглянуто клас неінтерактивних протоколів доказів із нульовим розголошенням zk-SNARK. Було розглянуто можливість їх використання у Mimblewimble для доказу того, що сума транзакції є ненульовою і не переповнюється без її розголошення, що значно підвищує анонімність транзакцій у системі порівняно з класичними системами. Було проведено обчислювальний експеримент із метою моделювання роботи запропонованого алгоритму та її продуктивності залежно від значень вхідних параметрів. Отримані дані свідчать, що він придатний для використання на практиці. Під час дослідження було зроблено дві публікації на відповідну тему.

CONCLUSION

In the course of the work, zero-knowledge proofs and the possibility of their application to anonymize financial transactions in distributed financial systems were studied. Various illustrative examples and protocols used in practice were described. The financial distributed system protocol Mimblewimble was reviewed. Its strengths and weaknesses were listed. Modification of this protocol has been proposed. The class of non-interactive zero-knowledge proof protocols zk-SNARK was described. The possibility of using them in Mimblewimble was proposed to prove that the transaction amount is non-zero and does not overflow without disclosing it, which significantly increases the anonymity of transactions in the system in comparison with classical systems. A computational experiment was performed to simulate the operation of the functioning of the proposed algorithm and its performance depending on the values of the input parameters. The obtained data indicates that it is suitable for practical usage. During the research, two publications on the relevant topic were made.

BIBLIOGRAPHY

1. Стан, досягнення та перспективи інформаційних систем і технологій / Матеріали XXI Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. Одеса, 22-23 квітня 2021 р. — Одеса, Видавництво ОНАХТ, 2021 р. — 229 с. — С. 51-52.
2. Current issues of science, prospects and challenges. Vol. 2 / Collection of scientific papers «SCIENTIA» with Proceedings of the I International Scientific and Theoretical Conference. Sydney, December 17 2021. — European Scientific Platform, 2021. — 111 p. — pp. 43-44.
3. Watrous J. Zero-Knowledge Against Quantum Attacks / John Watrous. — Waterloo, Ontario, Canada: Institute for Quantum Computing and School of Computer Science University of Waterloo, 2008. — 36 p.
4. Gowravaram N. R. Zero Knowledge Proofs and Applications to Financial Regulation [Електронний ресурс] / Gowravaram // Harvard College. — 2018. — Режим доступу до ресурсу: https://dash.harvard.edu/handle/1/38811528.
5. Santis A. D. Non-Interactive Zero-Knowledge Proof Systems / Santis A. D., Micali S., Persiano G. // Advances in Cryptology — CRYPTO '87: A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings / C. Pomerance, Lecture Notes in Computer Science — Berlin: Springer Berlin Heidelberg — 1988. — Vol. 293 — pp. 52—72.
6. Menezes A. J. Handbook of Applied Cryptography / A. J. Menezes, P. V. Oorschot, S. A. Vanstone — CRC Press, 1996. — 816 p. — pp. 405—417.
7. How to Explain Zero-Knowledge Protocols to Your Children / [J. Quisquater, L. C. Guillou, T. A. Berson and others] // Advances in Cryptology – CRYPTO '89: Proceedings / [J. Quisquater, L. C. Guillou, T. A. Berson and others]., 1990. — pp. 628–631.
8. Yao A. C. Protocols for Secure Computations / Andrew Yao., 1982. — 5 p. — (IEEE).
9. Feige U. Zero-knowledge proofs of identity / U. Feige, A. Fiat, A. Shamir, 1988. — (Journal of Cryptology). — pp. 77–94.

10. Introduction to Mimblewimble and Grin [Електронний ресурс]. — 2021. — Режим доступу до ресурсу: https://github.com/mimblewimble/grin/blob/master/doc/intro.md.
11. B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille and G. Maxwell, "Bulletproofs: Short Proofs for Confidential Transactions and More", 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2018, pp. 315-334.
12. Groth J. Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks / Jens Groth, Mary Maller. — Advances in Cryptology — CRYPTO 2017 — 37th Annual International Cryptology Conference. Santa Barbara, August 20-24 2017.— Part II.— pp. 581–612.
13. Scalable, transparent, and post-quantum secure computational integrity / E. Ben-Sasson, I. Bentov, Y. Horesh, M. Riabzev., 2018. — 83 p. — pp. 33-51.
14. Buterin V. An approximate introduction to how zk-SNARKs are possible [Електронний ресурс] / Vitalik Buterin. — 2021. — Режим доступу до ресурсу: https://vitalik.ca/general/2021/01/26/snarks.html.

Appendix A

Fragments of implementation used in the computational experiment

All the code snippets are written in Python 3

```
# Evaluate a polynomial at a specific coordinate
def eval_poly_at(p, x):
    y = 0
    power_of_x = 1
    for i, p_coeff in enumerate(p):
        y += power_of_x * p_coeff
        power_of_x = (power_of_x * x)
    return y

# Build a polynomial that returns 0 at all specified xs
def zpoly(xs):
    root = [1]
    for x in xs:
        root.insert(0, 0)
        for j in range(len(root)-1):
            root[j] -= root[j+1] * x
    return root

# Lagrange interpolations
def build_lagrange(x, y):
    return Polynomial(lagrange(x, y)).coef


def int_to_bits(n):
    return [int(digit) for digit in bin(n)[2:]]

def bits_to_int(bits):
    return int("".join(str(x) for x in bits), 2)


# Build the polynomial P
def build_p(n):
    bits = int_to_bits(n)
    y = [bits_to_int(bits[0:i+1]) for i in range(len(bits))]
```

```
    x = [i for i in range(len(y))]
    return build_lagrange(x, y)

def estimate_proof(n, ub):
    mercle_root_points = 10000
    start_time = time.time()
    proof = []

    p = build_p(n)
    print(p)
    z = zpoly([i for i in range(ub+1)])

    for i in range(0, int(math.log2(len(p)))):
        calculate_qr(p)
        proof.append(calculate_merkle_root(mercle_root_points))

    print('Time: ', time.time() - start_time)
    print('Proof size: ', np.array(proof, dtype=object).nbytes)
    return proof
```