Міністерство освіти і науки України
Харківського національного університету імені В.Н. Каразіна
Навчально-наукового інституту комп'ютерних наук та штучного інтелекту
Спеціальність 125 «Кібербезпека та захист інформації»
Освітня програма «Безпека інформаційних і комунікаційних систем»

В.о. зав. кафедрою КІСМТ
Марина ЄСІНА
«Допущено до захисту»

« »_____2024 р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Моделювання та програмна реалізація тестування низького ступеня у криптографічних протоколах доказу з нульовим розголошенням на основі швидкого перетворення Фур'є»

оцінка «_____»

Керівник:
д.т.н. професор кафедри
І.В. Лисицька_____

Голова ЕК

Лемешко О.В._____

Рецензент: к. т. н., доцент
І.В. Філіпенко_____

Виконавець: студент групи КБ-61
Аверков О.Ю._____

Харків 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра кібербезпеки інформаційних систем, мереж і технологій
Рівень вищої освіти (освітньо-кваліфікаційний рівень) магістр
Галузь знань: 12 Інформаційні технології
Спеціальність: 125 «Кібербезпека та захист інформації»
Освітня програма: «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри _____
Марина ЄСІНА

«____» _____2024_ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЕКТ)

Аверков Олег Юрійович
(прізвище, ім'я, по батькові студента)

1. Тема роботи «Моделювання та програмна реалізація тестування низького ступеня у криптографічних протоколах доказу з нульовим розголошенням на основі швидкого перетворення Фур'є/ Modelling and software implementation of low-degree testing in cryptographic zero-knowledge proof protocols based on fast Fourier transform»___

керівник роботи _Лисицька Ірина Вікторівна, д.т.н., професор_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «12» листопада 2024 року № 4101-5/3657

2. Строк подання студентом роботи_____22.11.2024_____

3. Перелік питань, які потрібно розробити
1) Визначити характеристику об'єкта та предмета дослідження, сформулювати мету і завдання дослідження;
2) Вивчити актуальність, сучасний стан та перспективність розвитку криптографічного протоколу доказу з нульовим розголошенням ZK-STARK;
3) Дати загальну характеристику криптографічного протоколу ZK-STARK з описанням його функціонування та визначенням особливостей функціонування першого етапу роботи арифметизації протоколу ZK-STARK;

4) Створити теоретичну та програмні реалізацію арифметизації першого етапу роботи протоколу ZK-STARK із застосуванням методу Гауса, методу обернених матриць та швидкого зворотного перетворення Фур'є;

5) Провести тестування програмних реалізацій першого етапу арифметизація протоколу ZK-STARK для оцінки їх часової складності;

6) Зробити порівняльний аналіз реалізації арифметизації протоколу ZK-STARK на основі швидкого перетворення Фур'є , на основі методу обернених матриць та на основі методу Гаусу;

7) Провести загальний аналіз результатів тестування першого етапу арифметизації протоколу ZK-STARK та надати рекомендації щодо найбільш ефективної роботи реалізації протоколу та щодо його впровадження до мережі Блокчейн.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1.	Визначення характеристик об'єкту та предмету дослідження, формулювання мети та завдання дослідження	01.09.2023-30.09.2023 р
2.	Аналіз сучасного стану криптографічного протоколу доказу з нульовим розголошенням ZK-STARK	01.10.2024 – 30.11.2023 р
3.	Математична обробка теоретичного матеріалу: проводиться із застосуванням Теорії інтерполювання, Теорії груп, Теорії чисел; відомостей про числа Фібоначчі; відомостей щодо функції Ейлера; щодо породжуючого елементу групи; відомостей про швидке перетворення Фур'є знань про циклічні групи; інтерполяційний поліном Лагранжа та послідовність обчислень	01.12.2023 – 31.01.2024
4.	Програмні реалізації: використовується мова програмування C++, середа розробки додатків Visual Studio 2022 та бібліотека «NTL» для розробки додатків, які використовують Теорію чисел	01.02.2024 – 31.04.2024 р
5.	Аналіз ефективності роботи програмних реалізацій протоколу ZK-STARK: за допомогою засобів середи розробки додатків Visual Studio 2022 та додатків: Notepad та Microsoft Office, зокрема Microsoft Excel	01.05.2024 – 30.06.2024 р
6.	Аналіз отриманих результатів шляхом написання висновків та рекомендацій	01.09.2024 – 30.09.2024 р
7.	Оформлення пояснювальної записки до кваліфікаційної роботи	01.10.2024 - 20.11.2024 р

5. Дата видачі завдання_____12.11.2024_____

Студент Аверков О.Ю. _____
ініціали, прізвище підпис

Керівник роботи Лисицька І.В. _____
ініціали, прізвище підпис

РЕФЕРАТ

Пояснювальна записка містить 65 сторінок, 14 рисунків, 1 таблицю, 5 додатків, 65 джерел посилань.

Метою кваліфікаційної роботи є вивчення, моделювання та тестування низького ступеня у криптографічних протоколах доказу із нульовим розголошенням шляхом програмної реалізації першого етапу арифметизація роботи протоколу ZK-STARK за допомогою швидкого перетворення Фур'є, методу Гауса та методу обернених матриць і надання рекомендацій щодо її ефективного впровадження до мережі Блокчейн.

Об'єктом дослідження кваліфікаційної роботи є процес забезпечення обчислювальної цілісності транзакції мережі Блокчейн.

Предметом розробки є ZK-STARK – криптографічний протокол, який використовує публічні докази з нульовим розголошенням.

Методи дослідження: математичні методи теорії інтерполювання (відомості про інтерполяційний поліном Лагранжа); теорії чисел (відомості про числа Фібоначчі; відомості про функцію Ойлера та її властивості); теорії груп (відомості про породжуючий елемент групи; відомості про циклічні групи).

Для програмної реалізації була обрана мова програмування C^{++},середа розробки додатків Visual Studio 2022 та бібліотека «NTL» версії 11.5 для розробки додатків, які використовують теорію чисел.

Подальше тестування цих програмних реалізацій проводилося на ПК, який має такі ключові характеристики: CPU: 13th Gen Intel(R) Core(TM) i7-13620H (16 CPUs), ~2.4GHz; RAM: - 32 Gb; об'єм диска - 750 Gb; операційна система Windows 11.

Результатами проведеної роботи. У роботі проведено моделювання та тестування низького ступеня у криптографічних протоколах доказу із нульовим розголошенням шляхом програмної реалізації першого етапу арифметизація роботи протоколу ZK-STARK.

Створено математичну модель арифметизації протоколу ZK-STARK. На її основі були написані три її програмні реалізації, у яких були використані метод Гаусу (часова складність $O(n^3)$), метод обернених матриць (часова складність $O(n^3)$), та метод швидкого зворотного перетворення Фур'є (часова складність $O(n*\log_2 n)$) для пошуку коефіцієнтів інтерполяційного поліному.

Встановлено, що практичне впровадження арифметизації на основі швидкого зворотного перетворення Фур'є має часову складність $O(n * log_2 n)$, яка у $\frac{n^3}{n*log_2 n}$ разів менша, ніж часова складність арифметизації на основі зворотних матриць та методу Гауса, що прискорює роботу арифметизації. Часова складність швидкого перетворення Фур'є дозволяє зменшити відсоток часу, який займає процес перетворення вхідних даних на поліном від часу роботи всього першого етапу із 99,99 % до 58,56% .

За результатами тестування можна зробити висновок, що найперспективнішим методом серед трьох протестованих виявився метод швидкого зворотного перетворення Фур'є. Тому рекомендується на етапі арифметизації протоколу ZK-STARK використовувати саме цей метод, бо він збільшить продуктивність роботи самого протоколу, тобто прискорить процедуру автентифікації, частиною якої і є етап арифметизація. арифметизація на основі швидкого зворотного перетворення Фур'є може прискорити етап арифметизації протоколу ZK-STARK, прискорюючи роботу самого протоколу, покращуючи масштабованість блокчейн-мережі. Тобто може збільшити кількість оброблених транзакцій за одиницю часу наприклад у сучасній криптовалюті (фінансова сфера). Це може стати новаторським шляхом до широкого впровадження для захисту цифрових активів. Також вона може бути впроваджена у системи електронного голосування та у системи потокового мовлення для покращення їхній масштабованості.

У якості подальшого дослідження задля пошуку способів прискорення роботи арифметизації протоколу ZK-STARK може бути запропонована програмна оптимізація ділення поліномів для ще більшого прискорення роботи першого етапу протоколу ZK-STARK та програмна реалізація ще швидшого

методу ніж ШЗПФ для пошуку коефіцієнтів інтерполяційного поліному.

Ключові слова: ПРОТОКОЛ, ZK-STARK, ARETHMETIZATION/АРИФМЕТИЗАЦІЯ, МОДЕЛЮВАННЯ, ПРОГРАМА, РЕАЛІЗАЦІЯ, МОВА ПРОГРАМУВАННЯ C^{++}, ТЕСТУВАННЯ, БЛОКЧЕЙН, ЕФЕКТИВНІСТЬ.

ABSTRACT

The explanatory note contains 65 pages, 14 figures, 1 tables, 5 appendices, 65 sources.

The purpose of the qualification work is to study the modeling and testing of low-degree in cryptographic protocols of proof with zero knowledge by software implementation of the first stage of Arethmetization of the ZK-STARK protocol using fast Fourier transform, Gauss method and the inverse matrix method and providing recommendations for its effective implementation in the blockchain network.

The object of study of the qualification work is the process of ensuring the computational integrity of the Blockchain network transaction.

The subject of development is ZK-STARK - a cryptographic protocol that uses public proofs with zero disclosure.

Research methods: mathematical methods from interpolation theory (information about the Lagrange interpolation polynomial); number theory (information about Fibonacci numbers; information about the Euler function and its properties); group theory (information about the generating element of a group; information about cyclic groups).

For the software implementation, the C++ programming language was chosen, along with the Visual Studio 2022 development environment and the NTL library (version 11.5) for developing applications that utilize number theory.

Subsequent testing of these software implementations was conducted on a PC with the following key specifications: CPU: 13th Gen Intel(R) Core(TM) i7-13620H (16 CPUs), ~2.4GHz; RAM: 32 GB; Disk Capacity: 750 GB; Operating System: Windows 11.

Results of the work. In this work, we have modeled and tested a low degree of proof in cryptographic protocols with zero knowledge by software implementation of the first stage «Arithmetization» of the ZK-STARK protocol.

A mathematical model of Arithmetization of ZK-STARK protocol was created.

On its basis, three of its software implementations were written, which used the Gaussian method (time complexity $O(n^3)$), the method of inverse matrices (time complexity $O(n^3)$), and the method of fast inverse Fourier transform (time complexity $O(n*\log_2 n)$ to find the coefficients of the interpolation polynomial. The time complexity of the fast Fourier transform reduces the percentage of time that the process of converting input data to a polynomial takes from 99.99% to 58.56% of the time of the entire first stage.

Testing has shown that the practical implementation of the Arithmetization based on the inverse fast Fourier transform has a time complexity $O(n*\log_2 n)$, that is in $\frac{n^3}{n*log_2 n}$ times less than the time complexity of the Arithmetization based on inverse matrices method and Gaussian method for interpolation, that speeds up the work of Arithmetization of the ZK-STARK protocol.

Testing has shown that the most promising method among the three tested was the IFFT method, so it is recommended to use this method at the Arithmetization stage of the ZK-STARK protocol, as it will increase the effectiveness of the protocol and will speed up the authentication procedure, of which the Arithmetization stage is a part. Arithmetization based on the inverse fast Fourier transform (IFFT) can accelerate the Arithmetization phase of the ZK-STARK protocol, thereby enhancing the performance of the protocol itself and improving the scalability of blockchain networks. This means it can increase the number of transactions processed per unit of time, for example, in modern cryptocurrencies (financial sector), making it a pioneering approach for widespread adoption in the protection of digital assets. Additionally, it can be implemented in electronic voting systems and streaming systems to improve their scalability.

As a direction for further research aimed at accelerating the Arithmetization phase of the ZK-STARK protocol, potential approaches include software optimization of polynomial division to further enhance the speed of the first phase of the ZK-STARK protocol, as well as the development and implementation of a faster method than the inverse fast Fourier transform (IFFT) for finding the coefficients of the interpolation polynomial.

Keywords: PROTOCOL, ZK-STARK, ARETHMETIZATION/ARITHMETIZATION, MODELING, PROGRAM, IMPLEMENTATION, C^{++} PROGRAMMING LANGUAGE, TESTING, BLOCKCHAIN, EFFICIENCY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА СИМВОЛІВ

КПМАП-ZKSTARK	-	Консольна програма для моделювання арифметизації протоколу ZK-STARK
AZkS	-	Arithmetization ZK-STARK (скорочена назва програми)
СЛАР	-	Система лінійних алгебраїчних рівнянь
ZK-STARK	-	Zero-Knowledge Scalable Transparent Argument of Knowledge
ZK-SNARK	-	Zero-Knowledge Succinct Non-interactive Argument of Knowledge
Г	-	Метод Гаусу
ОМ	-	Метод обернених матриць
ШЗПФ	-	Метод швидкого зворотного перетворення Фур’є

ВСТУП

Сучасний світ, зі стрімким розвитком Інтернету, швидко змінюється, і ці зміни продовжують прискорюватися. Це робить проблему збереження та обробки даних ще актуальнішою. Розглянемо сучасний стан цієї проблеми: раніше зберігання даних ґрунтувалося на довірі до власника серверів, який відповідає за незмінність, конфіденційність та доступність даних. Цей підхід можна назвати «традиційною моделлю», яка покладається на централізовані бази даних. Сьогодні стає очевидним, що централізовані системи не завжди забезпечують повну довіру. Користувачі прагнуть не тільки довіряти, а й мати можливість перевіряти цілісність своїх даних, особливо коли це стосується фінансових активів у банках, які насправді контролюються самими банками, а не клієнтами [1-3]. Наприклад, фінансова криза 2008 року підштовхнула людей до переосмислення своєї довіри до фінансових установ [4]. Блокчейн став рішенням цієї проблеми, забезпечуючи децентралізоване збереження та обробку даних без потреби у довірі до централізованих структур. Завдяки криптографії він створює систему, яка є незалежною від центральних органів управління. Так з'явилися криптовалюти, як Біткоїн та Ефіріум, що дозволяють зберігати активи безпосередньо у користувачів. Це означає, що активи тепер повністю належать користувачам, а відповідальність за їх конфіденційність, цілісність та доступність лежить на самих користувачах та їхньому обладнанні, а не на банках чи інших інституціях. Безпека таких транзакцій забезпечується сучасними криптографічними протоколами [1].

Ця нова модель, відома як модель «нового світу», втілена в інклюзивних блокчейнах і базується на девізі: «Не вір, а перевіряй». Її сучасна реалізація, наприклад в криптовалюті Ефіріум, відбувається через використання протоколу ZK-SNARK [1, 5]. Обчислювальна цілісність транзакцій у такому блокчейні забезпечується за рахунок підзвітності, коли будь-який вузол із базовою обчислювальною потужністю (наприклад, ноутбук з доступом до Інтернету) може

перевіряти всі транзакції в мережі. У централізованих системах перевірка транзакцій зазвичай потребує повного відтворення обчислень, що може призвести до втрати конфіденційності та обмежень у масштабованості. Щоб оптимізувати цей процес, блокчейн використовує докази з нульовим розголошенням, які є визнаним інструментом для підвищення конфіденційності та масштабованості. Цей підхід реалізовано в протоколах, таких як ZK-SNARK, ZK-STARK та інших рішеннях з нульовим знанням [1].

Сьогодні STARK є новою перспективною системою доказів, яка стійка до квантових атак, забезпечує масштабованість, прозорість та конфіденційність транзакцій і базується на спеціальній криптографії [1, 6]. Завдяки своїм властивостям STARK може швидко та без залучення третіх сторін перевіряти обчислювальну цілісність транзакцій. Основна мета протоколу STARK полягає у прозорій і лаконічній перевірці обчислювальної цілісності транзакцій [1]. Перший етап роботи STARK називається арифметизацією. На цьому етапі створюється слід виконання транзакції, будується поліном на основі цього сліду, а потім перевіряється його низький ступінь. Поліном буде мати низький ступінь лише тоді, коли слід виконання є правильним, а дані, що використовувалися, є коректними [7].

Практично вирішені задачі та застосування. У 2018 році Елі Бен-Сассон, Іддо Бентов, Інон Хореші та Михайло Рябзєв опублікували перші роботи, що описували протокол STARK [8]. На сьогодні єдина відома утиліта ZK-STARK розробляється компанією StarkWare, яку створили автори протоколу. Метою є створення тестового шару для впровадження технології в блокчейн, децентралізовані біржі та інші сфери. На даний момент STARK не пройшов широкомасштабне тестування та не отримав повного впровадження в реальні виробничі системи, зокрема й у сфері криптовалют [6]. Проте компанія StarkWare вже випустила альфа-версію мережі другого рівня Layer 2 Rollup Network для Ethereum, що використовує ZK-STARK [8].

Основна мета застосування систем нульового знання, таких як ZK-STARK, полягає у створенні високо захищених та конфіденційних систем. Ці системи

забезпечують повну децентралізацію даних, доступ до яких можливий лише при виконанні ряду суворо визначених умов. Досягнення таких умов значно ускладнене нетрадиційними методами, зокрема зломом [6].

Системи з децентралізацією інформації, такі як криптовалюти, зокрема Ефіріум, забезпечують не тільки безпеку мережі, але й захист користувачів, надаючи конфіденційність та анонімність залежно від умов [6]. Наприклад, у цьому контексті ZK-SNARK є особливо ефективним, оскільки він оптимально підходить для забезпечення приватності та анонімності без розкриття даних, водночас дозволяючи однозначно верифікувати транзакцію. Аналогічно, наприклад, реалізація Layer 2 Rollup Network для Ефіріум – StarkNet (платформа для розробки децентралізованих додатків (dApps) [9]) на базі ZK-STARK не розкриває інформацію (наприклад, приватну транзакцію), яку шифрує, але дозволяє перевірити її достовірність за будь-яких обставин [5, 10].

Одне з можливих застосувань ZK-STARK — підвищення масштабованості блокчейну шляхом зменшення розміру криптографічних доказів. У криптовалютах, таких як, наприклад, Біткоїн, де розмір блоку обмежує кількість транзакцій, що можуть оброблятися за секунду, це має критичне значення. Зменшення розміру доказів дозволяє зменшити обсяг транзакцій, що зберігаються в блоці, тим самим збільшуючи їхню кількість у кожному блоці. Особливо це стає помітним при обробці тисяч транзакцій, що покращує загальну масштабованість [6]. Сьогодні, ZK-STARK підвищює ефективність обробки транзакцій у Layer 2 Rollup Network для Ethereum, де є можливим групувати та перевіряти великі обсяги транзакцій за допомогою одного криптографічного доказу [11]. Однак це лише часткове вирішення проблеми масштабованості, оскільки навіть мінімальні криптографічні докази не можуть значно підвищити продуктивність блокчейну.

Системи такого типу можуть також бути використані, наприклад, для повністю зашифрованих та захищених систем потокового мовлення з захистом авторських прав, де традиційні методи шифрування на основі симетричної криптографії вже не будуть потрібні. Електронні системи голосування також отримують значні переваги від таких технологій, адже вони дозволяють виборцю

анонімно віддавати голос, який може бути перевірений, не розкриваючи особу виборця [6].

Також ZK-STARK є можливим інтегрувати в фінансові сервіси для забезпечення анонімності транзакцій та підвищення захисту даних, що особливо актуально у зв’язку зі зростанням популярності децентралізованих фінансів (DeFi) [12, 13]. Крім того, технологія ZK-STARK може стати основою для власності на цифрові активи в NFT-ринках (наприклад, Immutable X), забезпечуючи конфіденційність покупців і продавців без розкриття їхньої особистої інформації [14].Таким чином, потенціал використання ZK-STARK є значним і може знайти застосування в багатьох галузях.

Протокол STARK є новою технологією, яка поки не отримала широкого практичного застосування. Водночас, наразі вже виявлені деякі труднощі та проблеми, зокрема обчислювальна складність, можливі труднощі з інтеграцією з іншими протоколами та забезпечення стійкості до квантових атак [3]. У зв'язку з цим, у нашій роботі ми зосередили увагу на вирішенні проблеми обчислювальної складності та запропонували можливі шляхи її подолання.

Актуальність роботи. У протоколі STARK Прувер і Верифікатор демонструють найвищу швидкість у своєму класі, перевершуючи всі числові конструкції, і досягають значно більшої масштабованості порівняно з конкурентами (наприклад, SNARK), покладаючись на меншу кількість криптографічних припущень, які є більш безпечними. Вони також постквантово безпечні та прозорі, що означає відсутність використання небезпечних «криптографічних токсичних відходів» на етапі налаштування і відсутність «люків», які можуть загрожувати безпеці [15].

Протокол ZK-STARK є новітньою та складною у впровадженні технологією, яка має значний потенціал до вирішення головних проблем сучасних систем захисту інформації — масштабованість, обчислювальну цілісність та стійкість до квантових атак. Однак його ефективне застосування вимагає додаткових досліджень та вдосконалень. Поетапне вивчення і моделювання протоколу є важливим завданням для його впровадження та поширення, що має

на меті забезпечення обчислювальної цілісності транзакцій у сучасних блокчейн-мережах.

Мета роботи полягає у тестуванні трьох програмних реалізацій першого етапу арифметизації протоколу ZK-STARK (на основі методу Гаусу, методу обернених матриць та на основі швидкого зворотного перетворення Фур'є), розроблених на мові C^{++} з ціллю виявлення найбільш ефективної з них і надання рекомендацій щодо її практичного застосування.

Протокол ZK-STARK є новою, складною у впровадженні технологією, яка має потенціал вирішити основні проблеми сучасних протоколів безпеки — обчислювальна складність, масштабованість та стійкість до квантових атак. Однак його повноцінне впровадження потребує додаткових досліджень та вдосконалень. Тому поетапне вивчення та моделювання протоколу та етапів його роботи, зокрема арифметизації, є важливим завданням для забезпечення обчислювальної цілісності транзакцій у блокчейн-мережах.

Ця робота надає важливі знання про протокол нульового знання ZK-STARK, його принципи роботи та ефективність впровадження. Вона сприяє впровадженню технології доказів з нульовим знанням у блокчейн-мережі, зокрема технології ZK-STARK, демонструючи її ефективність на прикладі реалізацій арифметизації мовою програмування C^{++}.

Особливості програмної реалізації арифметизації протоколу ZK-STARK та результати тестування арифметизації на ефективність, які представлені в роботі, дозволяють запрограмований за допомогою швидкого зворотного перетворення Фур'є етап арифметизації рекомендувати для застосування у блокчейні, який сьогодні набув великої популярності у фінансовій сфері.

Слід відмітити роботи [16-18], у яких надано програмні реалізції протоколу ZK-STARK. Виконана робота відрізняється від робіт [16-18] наступним. По-перше, вона має відмінну від цих робіт авторську реалізацію арифметизації протоколу ZK-STARK, у якій інтерполювання запрограмовано трьома методами: Гаусу, обернених матриць та методом швидкого зворотного перетворення Фур'є. По-друге, в даній роботі виконано тетстування цих трьох програмних реалізацій

арифметизації та проведено порівняльний аналіз ефективновсті цих програмних реалізацій. По-третє, на основі результатів тестування надані практичні рекомендації щодо впровадження арифметизації протоколу ZK-STARK у блокчейн.

Об'єктом дослідження кваліфікаційної роботи є процес забезпечення обчислювальної цілісності транзакції мережі Блокчейн.

Предметом розробки є ZK-STARK – криптографічний протокол, який використовує публічні докази з нульовим розголошенням.

Метою кваліфікаційної роботи є вивчення, моделювання та тестування низького ступеня у криптографічних протоколах доказу із нульовим розголошенням шляхом програмної реалізації першого етапу арифметизація роботи протоколу ZK-STARK за допомогою швидкого перетворення Фур'є, методу Гауса та методу обернених матриць і надання рекомендацій щодо його ефективного впровадження до мережі Блокчейн.

Перелік завдань, які потрібно розробити:

1) Визначити актуальність, сучасний стан та перспективність розвитку криптографічного протоколу доказу з нульовим розголошенням ZK-STARK.

2) Дати загальну характеристику криптографічного протоколу ZK-STARK з описанням його функціонування та визначенням особливостей функціонування першого етапу роботи арифметизації протоколу ZK-STARK;

3) Створити теоретичну та програмні реалізацію арифметизації першого етапу роботи протоколу ZK-STARK із застосуванням методу Гауса, методу обернених матриць та швидкого зворотного перетворення Фур'є;

4) Провести тестування програмних реалізацій першого етапу арифметизація протоколу ZK-STARK для оцінки їх часової складності.

5) Зробити порівняльний аналіз реалізації арифметизації протоколу ZK-STARK на основі швидкого перетворення Фур'є , на основі методу обернених матриць та на основі методу Гаусу.

6) Провести загальний аналіз результатів тестування першого етапу арифметизації протоколу ZK-STARK та надати рекомендації щодо найбільш

ефективної роботи реалізації протоколу та щодо його впровадження до мережі Блокчейн.

1 ОГЛЯД ЛІТЕРАТУРИ

Історія розвитку протоколу ZK-STARK розпочалася 2 липня 2018 року, коли Ethereum Foundation надав компанії StarkWare дворічний грант на розробку STARK-оптимізованої хеш-функції (SFH). Ця функція була призначена для використання у прозорих та постквантових системах доказів у блокчейн-мережі [19]. З того часу актуальним залишається питання програмного впровадження ZK-STARK, чим компанія StarkWare [20] активно займається і сьогодні. Використання STARK вважається перевагою порівняно з іншими протоколами, що забезпечують обчислювальну цілісність транзакцій у блокчейні, завдяки численним перевагам у сфері захисту інформації, продемонстрованим фахівцями StarkWare [6].

На сьогоднішній день блокчейн [21, 22] став першою платформою для реалізації протоколу ZK-STARK у практичному використанні. Блокчейн можна описати як технологію обробки та зберігання даних, а також ідентифікації користувачів. Сам термін «блокчейн» (blockchain) походить від англійської назви «ланцюжок блоків», і технологія була вперше представлена у 2008 році особою або групою осіб під псевдонімом Сатоші Накамото [23, 24].

Блокчейн має кілька ключових характеристик [23, 24].

1) Прозорість – усі транзакції зберігаються в мережі та доступні для перегляду.

2) Стабільність – неможливо видалити або змінити дані заднім числом, лише додати нові транзакції.

3) Децентралізація – дані зберігаються на багатьох вузлах мережі, а не на одному сервері.

Таким чином, блокчейн можна вважати базою даних, яка реалізує такі основні ідеї [23, 24]:

1) Транзакції

2) Незмінні реєстри

3) Децентралізовані вузли

4) Процеси шифрування

5) Механізм консенсусу

6) Смарт-контракти (за необхідності).

Серед основних методів захисту інформації в блокчейні виділяють три типи [1, 25, 26]:

1) Шифрування даних

2) Алгоритм консенсусу

3) Алгоритми перевірки достовірності транзакцій.

Якщо, коротко, шифрування - процеси криптографічного перетворення інформації, що складаються із взаємо - однозначних процесів зашифровування, в результаті виконання якого відкрита інформація відображається в шифротекст, та розшифровування, в результаті виконання якого шифротекст відображається у відкриту інформацію [27]. В мережі блокчейн шифрування захищає інформацію в середині транзакціїї мережі. Наприклад, у Bitcoin використовується асиметричний криптографічний алгоритм ECDSA, що використовується для створення цифрових підписів у Bitcoin, SHA-256 для створення криптографічних хешів, які забезпечують цілісність блоків і транзакцій та RIPEMD-160 поєднанні із SHA-256 для створення Bitcoin-адрес [28].

Щоб забезпечити автентичність транзакцій, блокчейн застосовує консенсусні моделі, які дозволяють групам користувачів, які не довіряють один одному, співпрацювати. Згідно з дослідженнями фахівців [29-31], консенсус є процесом прийняття рішень, метою якого є досягнення згоди серед усіх учасників щодо поточного стану мережі після додавання нових даних, блоку або транзакцій. Консенсус-протокол забезпечує правильність ланцюга, мотивує учасників залишатися чесними та запобігає централізації контролю системи. Він також гарантує дотримання правил мережі та підтверджує автентичність транзакцій через механізм, за яким більшість вузлів підтверджують достовірність конкретної транзакції, перш ніж додати її до блокчейну.

Для забезпечення конфіденційності, цілісності, доступності разом із

властивістю нульового розголошення транзакцій в мережі блокчейн обов'язковими засобами верифікації транзакцій або обчислень, паралельно із алгоритмом досягнення консенсусу, в мережі блокчейн використовуються протоколи перевірки автентичності транзакції, такі як ZK-STARK або, наприклад, ZK-SNARK, що можуть бути використовуваними паралельно із алгоритмом досягнення консенсусу, наприклад «PoWeight» та «PoB», задля того, щоб учасники не потребували знання всіх даних, що лежать в основі кожної транзакції [32, 33]. Наприклад, Zcash - це перше доступне застосування ZK-SNARKs в мережі блокчейн [34]. В роботі висвітлюється саме спосіб веріфікації транзакції на достовірність за допомогою етапу арифметизації протоколу ZK-STARK, який при перевірці транзакції не розкриває суть самої транзакції.

Сама абревіатура STARK в назві протоколу ZK-STARK дослівно означає «масштабований прозорий аргумент знання, що масштабується». Масштабованість означає, що при збільшенні кількості даних або розміру завдання, час, необхідний для створення доказу, збільшується квазілінійно, а час верифікації доказу збільшується полілагорифмічно [35]. Термін «прозорий» означає, що всі повідомлення Веріфікатор є випадковими числами, які генеруються публічно [35]. Це означає, що сутності протоколу ZK-STARK – Прувер та Веріфікатор можуть бачити процес перевірки доказу, який використовує випадкові числа, які теж їм відомі [35]. Термін «аргумент знання» позначає криптографічний доказ, що підтверджує автентичність або право власності на дані чи транзакцію в блокчейн-мережі. Це механізм, за допомогою якого одна зі сторін у мережі демонструє знання певної інформації без її розкриття. Іншими словами, цей підхід базується на технології «нульового знання» або принципі «нульового розголошення».

До структури всіх протоколів нульового розголошення відноситься наявність сутностей «Прувер» та «Веріфікатор», які у процесі взаємодії доводять, чи спростовують доказ. У протоколі Zero-Knowledge STARK (ZK-STARK) Прувер та Верифікатор виконують різні ролі [35-37]:

«Прувер» - це сторона, яка намагається підтвердити коректність певного

твердження або обчислення, не розкриваючи при цьому конфіденційну інформацію. У нього є конкретний набір даних або обчислень, які він хоче довести, і його мета - переконати перевіряльника у правильності свого доказу. Прувер виконує обчислення та генерує доказ, який надсилає перевіряльнику.

«Веріфікатор» - це сторона, яка аналізує доказ, отриманий від Прувера. Він не має доступу до захищеної інформації, яку доводчик прагне зберегти, але здатен визначити коректність доказу. Мета Веріфікатора - встановити достовірність результатів, представлених Прувером. Для цього він застосовує математичні алгоритми та правила, щоб переконатися, що Прувер не обманює і дійсно надає правдиву інформацію.

Протоколи з нульовим розголошенням (позначені префіксом ZK у назві ZK-STARK) є криптографічними методами, які дозволяють Пруверу переконати Веріфікатора у тому, що він володіє певними знаннями, не розкриваючи ці знання. Формальніше, якщо задано відношення R, то ZK-протокол для цього відношення дозволяє Пруверу підтвердити Веріфікатор, що він знає свідка w для певного твердження s, так що (s; w) є R, не розкриваючи сам свідок w. Наприклад, Прувер може довести, що s є хешем коректно сформованих даних w (тобто, у випадку, коли відношення R складається з пар (s; w), де s = hash(w) і w є добре сформованими даними) [38].

ZK-протоколи є ключовими елементами в криптографії, оскільки вони дозволяють забезпечити чесну поведінку навіть у середовищі з потенційно зловмисними учасниками. Наприклад, використання ZK-протоколу може гарантувати, що певна сторона має право виконувати визначені дії або отримувати доступ до конфіденційної інформації [38].

Безпека ZK-протоколів визначається через властивості повноти, достовірності, доказу знання та нульового знання. Повнота забезпечує коректне виконання протоколу, якщо як Прувер, так і Веріфікатор діють згідно з правилами. Достовірність гарантує, що Прувер не зможе переконати Веріфікатора у правильності хибного твердження без свідка, окрім випадків з низькою ймовірністю. Доказ знання означає, що Прувер, який успішно переконує

Веріфікатора, дійсно володіє свідком. Нульове знання передбачає, що зловмисний Прувер не може отримати інформацію про свідка під час виконання протоколу [38].

Якщо сумувати наведені дані [8, 39-43], то характеристики протоколу є можливим зобразити у вигляді таблиці 1.1.

Таблиця 1.1 - Головні характеристики протоколу ZK-STARK

Головні риси	Значення риси протоколу ZK-STARK
Складність алгоритму Прувера	$O(n * poly\log(n))$
Складність алгоритму Веріфікатора	$O(poly\log(n))$
Комунікаційна складність	$O(poly\log(n))$
Оцінка розміру одиничної транзакції	45 КБ
Оцінка розміру десяти тисяч транзакцій	135 КБ
Вартість газу, який потребує віртуальна машина Ефіріуму	$\sim$2,5 мілліони (не залежить від конкретного алгоритму)
Потрібність у довіреному наборі	ні
Пост-квантова стійкість	так
Крипто-допущення	стійкий до геш-колізій
Масштабованість	більш масштабований ніж SNARK
Розмір доказу	45 КБ – 200 КБ, тобто $O(poly\log(n))$
Час доказу	1,6 с
Час перевірки доказу	16 мс
Тип криптографії	геш-функції

На даний момент визначені головні переваги та недоліки алгоритму ZK-STARK [43-50].

Якщо порівнювати ZK-STARK і ZK-SNARK, перший забезпечує кращу масштабованість, прозорість та безпеку в блокчейнах. На відміну від більшості

SNARK, STARK використовує геш-функції, які є квантово-стійкими, що надає додаткову захищеність від можливих атак квантових комп'ютерів. Також ZK-STARK не потребує «довіреного налаштування», що підвищує загальний рівень безпеки. Однак, STARK має більший розмір доказів і потребує більше часу на їх перевірку в порівнянні зі SNARK [8, 42-44]. ZK-SNARK, у свою чергу, має більшу підтримку серед розробників завдяки більш ранньому впровадженню та нижчій вартості використання. Для транзакцій SNARK використовує лише близько 24% від обсягу газу, необхідного для STARK, що робить SNARK більш економічно вигідним для кінцевих користувачів [8]. Крім того, SNARK були розроблені приблизно на шість років раніше, що сприяло їх швидшому впровадженню в екосистемі блокчейну [43]. ZK-STARK все ще знаходиться на ранніх етапах розвитку та вимагає додаткових зусиль для демонстрації своєї ефективності серед розробників та інших зацікавлених сторін в блокчейн-екосистемі [8, 49]. На сьогодні, існують такі перші початкові імплементації ZK-STARK у проектах другого рівня (Layer 2) [49, 51]:

1) StarkEx від StarkWare використовує технологію zk-STARK для підвищення ефективності транзакцій і зниження комісій за газ в Ethereum.

2) dYdX використовує zk-STARKs від StarkEx для приватної, економічно ефективної торгівлі з використанням кредитного плеча та інших фінансових інструментів.

3) Immutable X використовує zk-STARK для миттєвих, масштабованих і безгазових NFT-транзакцій, зберігаючи конфіденційність користувачів.

4) Блокчейн Ethereum вирішую проблему масштабованості за допомогою ZK-STARK від Starknet.

Взагалі, ZK-STARK поки-що масово не використовується, але у світі блокчейн мереж, наприклад, криптовалют, цей протокол має великий потенціал і може стати новаторським шляхом до широкого впровадження [8, 49].

Варто зазначити, що деякі системи для верифікації автентичності транзакцій можуть мати властивості як SNARK, так і STARK. Наприклад, неінтерактивні STARK можуть набувати характеристик SNARK, а прозорі SNARK з мінімальним

налаштуванням можуть функціонувати як STARK. Існують також інші системи доказів, які не є ні SNARK, ні STARK, але використовують властивість нульового знання, тобто не розкривають суть транзакції під час верифікації її істинності. Одним із прикладів є Bulletproofs [35].

Ще один приклад - Plonky2, який є рекурсивним SNARK. Plonky2 поєднує властивості STARK, зокрема швидкі докази та відсутність потреби у довіреному налаштуванні, із властивостями SNARK, такими як підтримка рекурсії та низька вартість верифікації в Ethereum [54]. Сьогодні вже існує Plonky2 Goldibear - це найновіша версія доведення Plonky2, що підтримує генерацію доказів в доменах Goldilocks і BabyBear. Також Telos Foundation також працює над апаратним прискоренням для Plonky2 Goldibear і прагне впровадити верифікатор доказів Polygon Hermez zkEVM [53, 54].

Отже, перший етап реалізації протоколів ZK-STARK називається арифметизацією [55] і складається з двох основних кроків. Спершу генерується слід виконання та поліноміальні обмеження для транзакції, створеної користувачем. Потім, ці об’єкти перетворюються на один поліном низького ступеня [55], який буде таким лише за умови, що слід виконання та вхідні дані є правильними [55]. Процес арифметизації в ZK-STARK реалізується методом AIR [33, 56]. Завдяки своїй стійкості до квантових атак, ZK-STARK може стати надійною альтернативою іншим протоколам, які можуть бути вразливими до квантових обчислень, як ZK-SNARK.

Загалом, протокол ZK-STARK є складним у впровадженні, але має значний потенціал для вирішення головних проблем - масштабованість, обчислювальну цілісність та стійкість до квантових атак. Це робить подальше дослідження, моделювання та поетапне вивчення ZK-STARK важливим для забезпечення обчислювальної цілісності транзакцій у блокчейн-мережах.

2 ТЕОРЕТИЧНА РЕАЛІЗАЦІЯ АРИФМЕТИЗАЦІЇ ПРОТОКОЛУ ZK-STARK

2.1 Поля, що використані для розрахунків

Полем називається множина з двома визначеними на ній операціями «+» і «•» при цьому виконуються такі аксіоми [57]:

1) Щодо операції + - множина є абелевою групою.

2) Поле замкнуте відносно операції «•» і множина ненульових елементів утворюють абелеву групу відносно операції «•».

3) Виконується закон дистрибутивності:

Для всіх a, b, c ∈ H:$(a + b) \bullet c = a \bullet c + b \bullet c$

4) Одиничним елементом відносно операції + є «0» ($e_+=0$). Зворотний елемент позначається як «-a». Для операції «•» одиничний елемент «1» ($e_\bullet=1$). При цьому зворотний елемент позначається як a^{-1}.

Поле з q (де q – просте число) елементами називається скінченним полем або полем Галуа і позначається GF(q). Потужність множини елементів називається порядком поля (кількість елементів поля). Якщо множина скінченна, тоді кажуть, що і поле скінченне, це поле Галуа і q є порядком поля. Усі операції у полі проходять за модулем [57].

Для тестування було обране поле GF(257), яке має підгрупи порядку 2^i для усіх i = 1,2..8, тобто на вхід у модель арифметизації можна подати 2^i чисел Фібоначчі для усіх i = 1,2..8. Також було використане поле GF(503), яке містить підгрупу порядку 502 та поле GF(1013), яке містить підгрупу порядку 1012.

Полем Золотоволоски (Goldilocks field) є кінцеве поле GF(p) визначене для простого числа $p = 2^{64} – 2^{32} + 1$, тобто поле GF($2^{64} – 2^{32} + 1$) [58]. Це поле містить нетривіальні підгрупи порядку 2^i для усіх i = 1,2..32 [58]. Це дозволяє реалізовувати швидке перетворення Фур'є для векторів, які мають довжину 2^i , для усіх i = 1,2..32 [2]. У нашій роботі в якості першообразного корню був обраний корінь g=7 [58].

2.2 Відомості про методи пошуку коефіцієнтів інтерполяційного полінома

Для розв'язання послідовностей розрахунків в арифметизації необхідно спочатку розрахувати коефіцієнти інтерполяційного поліному $f(g^0x)$, де g^i – спеціальний коефіцієнт та $i \in [0;3)$ [7]. Задля цього, наприклад, є можливим використати наступну формулу для пошуку коефіцієнтів інтерполяційного поліному Лагранжа [59], але за умови, що розрахунки проводяться у конкретному полі:

$$L_n(x) = \sum_{i=0}^{n} f(x_i) \frac{\omega_n(x)}{(x - x_i) * \omega'_n(x_i)},$$

де x – аргумент інтерполяційного поліному, та функції f(x), та функції $\omega_n(x) = (x - x_0)(x - x_1)\ldots(x - x_n)$.

Після знайдення коефіцієнтів інтерполяційного поліному $f(g^0x)$ потрібно розрахувати також коефіцієнти поліномів для випадків $f(g^1x)$ та $f(g^2x)$.

У цій роботі буде запрограмована та протестована арифметизація на онові трьох методів пошуку коефіцієнтів інтерполяційного поліному:

1) за допомогою методу Гаусу з вибором головного елементу по стовпцю;

2) за допомогою методу Обернених матриць із пошуком оберненої матриці методом Гаусу (використовується для переведення матриці до верхньотрикутникового вигляду);

3) за допомогою Швидкого перетворення Фур'є.

Перший варіант [60] еквівалентний застосуванню звичайного методу Гауса до системи, в якій на кожному кроці виключення проводиться відповідна перенумерація рівнянь:

$$\left|a_{rk}^{(k)}\right| = max_i \left|a_{ik}^{(k)}\right|, \; k \le i \le n.$$

Проте головний елемент можна вибирати і серед всіх елементів неперетвореної частини матриці (рис. 2.1, б).

$$\left|a_{rs}{}^{(k)}\right| = max_{i,j}\left|a_{ij}{}^{(k)}\right|, \qquad k \le i, \qquad j < n$$

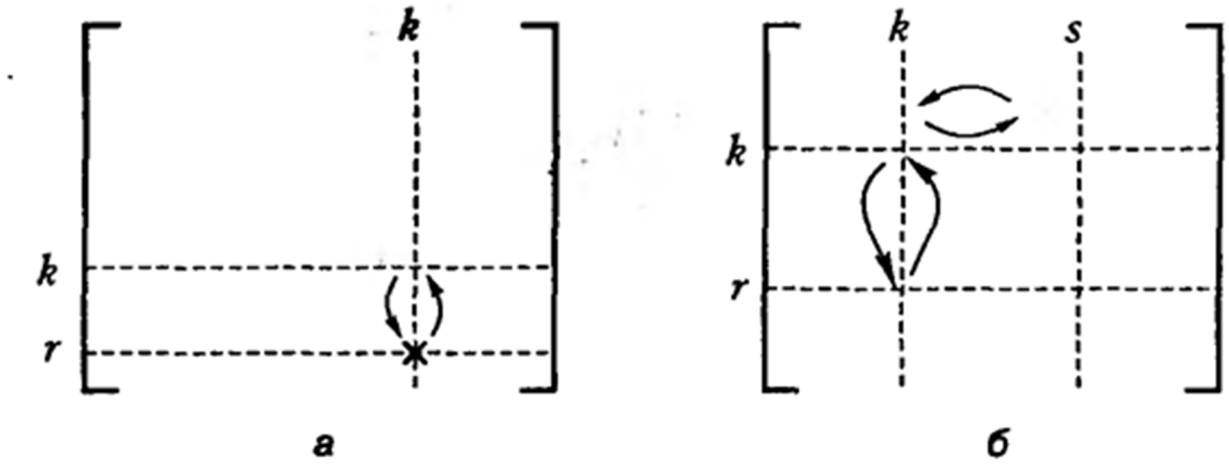

Рисунок 2.1 - Вибір головного елемента:
а – серед елементів стовпця матриці; б – серед елементів не перетвореної частини матриці

Складність алгоритму (кількість арифметичних операцій) оцінюється величиною

$$f_A(n) = \frac{n^3}{3} + n^2 = O(n^3).$$

Алгоритм буде модифікований задля його використання в модульній арифметиці.

Розглянемо другий варіант, у якому для пошуку коефіцієнтів інтерполяційного поліному застосовується метод обернених матриць із пошуком оберненої матриці методом Гаусу [61]. Він використовується для переведення матриці до верхньотрикутникового вигляду.

Нехай дана система:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
$$\cdots$$

$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n$$

Або в матричній формі

$$A*X = B, \quad (2.1)$$

де A = (a_{ij}) - матриця із коефіцієнтами при невідомих, а B і X – вектор стовпців, що складається із вільних членів і із невідомих.

Якщо матриця A невироджена, тобто визначник системи $\Delta = |A| \neq 0$, то помноживши обидві частини рівняння (2.1) на матрицю A-1 зліва, отримаємо рішення системи в матричній формі: $X = B*A^{-1}$.

Часова складність методу: $O(n^3)$

Третій варіант пошуку коефіцієнтів інтерполяційного поліному ми проводимо за допомогою Швидкого перетворення Фур'є [62]. Цей метод базується на звичайному дискретному перетворенні Фур'є (ДПФ) та зворотному дискретному перетворенні Фур'є (ЗДПФ).

Дискретне перетворення Фур'є (за англійською абревіатурою, DFT Discrete - Fourier Transform) - це математичний інструмент, який використовується для перетворення дискретних, звичайно обмежених послідовностей в іншу область - зазвичай із часової області в частотну [62].

ДПФ визначено таким чином [62].

Для послідовності чисел $v_0, v_{1,\ldots}\ v_{n-1}$, його ДПФ це інша послідовність чисел $V_0, V_{1,\ldots}\ V_{n-1}$, яка задана таким чином [62]:

$$V_j = \sum_{k=0}^{n-1} v_k e^{-\frac{2\pi i}{n} jk}, \text{ j} = 0,1,\ldots\text{n-1}, \quad (2.2)$$

де i – мнима одиниця ($i^2 = -1$), а e – це основа натурального логарифма.

Формула Ейлера пов'язує тригонометричні функції з комплексною експонентною [62]:

$$e^{ix} = \cos(x) + i\sin(x)$$

Використовуючи формулу Ейлера вираз (2.2) можемо записати у вигляді [62]:

$$V_j = \sum_{k=0}^{n-1} v_k e^{-\frac{2\pi i}{n} jk} = \sum_{k=0}^{n-1} v_k * (\cos\left(\frac{2\pi jk}{n}\right) - i * \sin\left(\frac{2\pi jk}{n}\right)).$$

ДПФ являє собою комплексне перетворення, тобто кожен елемент вихідної послідовності V = (V_0,$V_{1,...}$ V_{n-1}) є комплексним числом [62]:

$$V_j = \mathrm{Re}(V_j) + i\mathrm{Im}(V_j)$$

Модуль цього комплексного числа зазвичай використовується для представлення амплітуди частоти [62]:

$$A_j = \frac{|X_j|}{n} = \frac{\sqrt{\mathrm{Re}(V_j)^2 - \mathrm{Im}(V_j)^2}}{n},$$

А фаза (кут) використовується для представлення фази сигналу на цій частоті [62]:

$$\varphi_i = \arg(V_i) = \arctan\left(\frac{\mathrm{Im}(V_j)}{\mathrm{Re}(V_j)}\right) = \arcsin(\frac{\mathrm{Im}(V_j)}{A_j}) = \frac{2\pi}{n} jk = w_j t_k,$$

- де j – індекс частоти;
- частота j-го синусоїдного сигналу дорівнює $f_i = \frac{j}{T}$, де T - період часу протягом якого бралися вхідні дані;
- $w_j = 2\pi f_i = \frac{2\pi j}{T}$ – кругова частота (радіальна частота, частота обертів);

$$w_j t_k = \frac{2\pi j}{T} * \frac{k}{n} T = \frac{2\pi}{n} * jk$$

Зворотне дискретне перетворення Фур'є (ОДПФ) являє собою операцію, зворотну ДПФ. Вона дає змогу перейти від подання в частотній області назад у

тимчасову область. ЗДПФ для послідовності чисел V = (V_0,$V_{1,...}$ V_{n-1}) визначається таким чином [62]:

$$v_k = \frac{1}{n}\sum_{k=0}^{n-1} V_j e^{-\frac{2\pi i}{n}jk} = \frac{1}{n}\sum_{j=0}^{n-1} V_j * (\cos\left(\frac{2\pi jk}{n}\right) - i * \sin\left(\frac{2\pi jk}{n}\right)),\ k=0,1\ldots,n-1$$

Класичне дискретне перетворення Фур'є (ДПФ або англіською DFT) вимагає складності O (n^2), де n - це розмір вхідної послідовності. Швидке перетворення Фур'є (ШПФ або англійською FFT) скорочує цю складність до $n * log_2(n)$, що робить його набагато ефективнішим для великих послідовностей [62]. Основне припущення для більшості алгоритмів FFT - це те, що n є ступенем двійки: $n = 2^m$, $m \in N$. Інтерполяція проходить із використанням методу Кулі-Тьюкі для розрахунку Оберненого швидкого перетворення Фур'є [62]. Він розбиває DFT на два менші завдання DFT (заснований на принципі «розділяй і володарюй»), кожне з яких має половину розміру вихідного завдання [62].

Один із найвідоміших алгоритмів FFT - це алгоритм Кулі-Тьюкі, який заснований на принципі «розділяй і володарюй». Він розбиває DFT на дві менші завдання DFT, кожна з яких має половину розміру вихідної завдання.

Нехай у нас є послідовність чисел v = (v_0, v_1,… v_{n-1}) та ми бажаємо обчислити її DFT V = (V_0, V_1,… V_{n-1}). Замість прямого використання алгоритму DFT, алгоритм Кулі-Тьюкі розбиває його на дві частини одну для парних індексів та одну для непарних індексів [62]:

$$V_j = E_j + e^{-1*\left(\frac{2*\pi*i}{n}\right)*j} * O_j,\ j = 0,1..,\ n-1$$

де E_j – DFT для парних індексів:

$$E_j = \sum_{k=0}^{n/2-1} v_{2k} e^{-\frac{2\pi i}{n/2}jk} = \sum_{k=0}^{n/2-1} v_{2k}\varepsilon_{-2jk},$$

а O_j – DFT для непарних індексів:

$$O_j = \sum_{k=0}^{n/2-1} v_{2k+1} e^{-\frac{2\pi i}{n/2}jk} = \sum_{k=0}^{n/2-1} v_{2k+1}\varepsilon_{-2jk},$$

Тепер замість обчислення одного DFT розміру n, обчислюються два DFT розміру n/2. Цей процес можна повторити рекурсивно, розбиваючи DFT на більш малі частини поки розмір не стане маленьким для прямого обчислення [62]. Також завдяки властивостям комплексних чисел є можливим вдруге використовувати результати обчислень E_j та O_j для обчислення V_j.

Слід звернути увагу, що завдяки властивостям комплексних експонент $\varepsilon_{-j} = \varepsilon_{-n/2-j}$ є можливим вдруге використати результати обчислень для E_j та O_jу межах j=0,1…, n/2 -1 для обчислень V_j умежах j= n/2,…, n -1:

$$E_{n/2+j} = \sum_{k=0}^{n/2-1} v_{2k}\varepsilon_{-2(n/2+j)k} = \sum_{k=0}^{n/2-1} v_{2k}\varepsilon_{-nk-2jk} = \sum_{k=0}^{n/2-1} v_{2k}\varepsilon_{-2jk} = E_j,$$

$$O_{n/2+j} = \sum_{k=0}^{n/2-1} v_{2k+1}\varepsilon_{-2(n/2+j)k} = \sum_{k=0}^{n/2-1} v_{2k+1}\varepsilon_{-nk-2jk} = \sum_{k=0}^{n/2-1} v_{2k+1}\varepsilon_{-2jk} = \\ = O_j\ ,$$

$$V_{n/2+j} = E_{n/2+j} + e^{-1*\left(\frac{2*\pi*i}{n}\right)*(n/2+j)} * O_{n/2+j} = E_j + \varepsilon_{-n/2-j} * O_j = E_j - O_j$$

де, j = 0,1.., n/2 – 1.

Це дозволяє нам уникнути обчислення DFT для кожного j незалежно, що призводить до значного зменшення обчислювальних витрат [62].

Таким чином, алгоритм FFT дає змогу ефективно обчислювати DFT, скорочуючи час обчислень для великих послідовностей [62]. Це робить його дуже важливим інструментом у багатьох галузях, включно з обробкою сигналів, обробку зображень, розв'язання диференціальних рівнянь і багато інші [62].

Дискретне перетворення Фур'є можна також інтерпретувати в термінах оцінки многочленів [62].

Для вхідного вектору v = (v_0,$v_{1,...}$ v_{n-1}) є можливим асоційовати поліном ступеню n-1 [62]:

$$P(x) = v_0 + v_1 x + v_2 x^2 \dots v_{n-1} x^{n-1} = \sum_{k=0}^{n-1} v_k x^k$$

Тоді DFT вектору v = (v_0,$v_{1,\dots}$ v_{n-1}) дорівнює оцінці поліному P(x) у комплексних коренях з одиниці $x = \varepsilon_{-j} = e^{-\frac{2\pi i}{n} j}$ для j=0,1…, n -1[62]:

$$V_j = P\left(\varepsilon_{-j}\right) = \sum_{k=0}^{n-1} x_k {\varepsilon_{-j}}^k = v_0 + v_1 \varepsilon_{-j} + v_2 {\varepsilon_{-j}}^2 \dots v_{n-1} {\varepsilon_{-j}}^{n-1}$$

Це слідує із означення DFT[62]:

$$V_j = \sum_{k=0}^{n-1} x_k e^{-\frac{2\pi i}{n} jk} = \sum_{k=0}^{n-1} v_k {\varepsilon_{-j}}^k = P\left(\varepsilon_{-j}\right)$$

Алгоритм FFT, зокрема метод Кулі-Тьюкі, використовує цю інтерпретацію, щоб розбити завдання оцінки многочлена на менші завдання[62]. Ми можемо розкласти многочлен P(x) на многочлени, що включають тільки парні та непарні ступені x, тобто [62]:

$$P(x) = E(x^2) + x * O(x^2),$$

де

$$E(x) = v_0 + v_2 x + v_4 x^2 \dots v_{n-2} x^{(n/2)-1} = \sum_{k=0}^{(n/2)-1} v_{2n} x^k$$

та

$$O(x) = v_1 + v_3 x + v_5 x^2 \dots v_{n-1} x^{(n/2)-1} = \sum_{k=0}^{(n/2)-1} v_{2n+1} x^k$$

Тепер оцінка поліному P(x) на одиничній окружності зводиться до оцінки поліномів E(x) та O(x) на квадратних коренях з одиниці [62]:

$$V_j = \sum_{k=0}^{\left(\frac{n}{2}\right)-1} v_{2k} e^{-\frac{2\pi i}{\frac{n}{2}}jk} + e^{-\frac{2\pi i}{n}j} * \sum_{k=0}^{\left(\frac{n}{2}\right)-1} v_{2k+1} e^{-\frac{2\pi i}{\frac{n}{2}}jk} =$$

$$= E\left({\varepsilon_{-j}}^2\right) + \varepsilon_{-j} * O\left({\varepsilon_{-j}}^2\right) =$$

$$= E\left(\varepsilon_{-2jmodn}\right) + \varepsilon_{-j} * O\left(\varepsilon_{-2jmodn}\right) = P\left(\varepsilon_{-j}\right)$$

Використаємо рівність [62]:

$$\varepsilon_{-j} = -\varepsilon_{-n/2-j},$$

Звідки маємо [62]:

$$E\left({\varepsilon_{-n/2-j}}^2\right) = E\left(\left(-\varepsilon_{-j}\right)^2\right) = E\left(\left(\varepsilon_j\right)^2\right)$$

$$O\left({\varepsilon_{-n/2-j}}^2\right) = O\left(\left(-\varepsilon_{-j}\right)^2\right) =) \left(\left(\varepsilon_j\right)^2\right)$$

Це дає можливість вдруге використати результати обчислені для $E\left({\varepsilon_j}^2\right)$ та $O\left({\varepsilon_j}^2\right)$ у межах j=0,1…, n/2 -1 для обчислення V_j у межах j= n/2,…, n-1 [62]:

$$V_j = E\left({\varepsilon_{-n/2-j}}^2\right) + \varepsilon_{-n/2-j} * O\left({\varepsilon_{-\frac{n}{2}-j}}^2\right) = E\left({\varepsilon_{-j}}^2\right) + \varepsilon_{-j} * O\left({\varepsilon_{-j}}^2\right), j=0,1\ldots,n/2-1$$

Таким чином, розбиття поліномів на менші поліноми є ключовим моментом, який дозволяє алгоритму FFT скоротити кількість обчислень з $O(n^2)$ (для звичайного DFT) до $n * log_2(n)$. До того ж, це розбиття можна застосовувати рекурсивно, продовжуючи розділяти поліноми, доки розмір перетворення не стане достатньо малий, щоб його можна було обчислити безпосередньо [62].

Для того, щоб використати цей метод для пошуку коефіцієнтів інтерполяційного поліному, потрібно змінити напрямок обходу комплексної площини, домножаючи ступінь числа e на мінус один та домножити само

отримане число $e^{-1*(\frac{2*\pi*i}{n})}$ на $\frac{1}{n}$, де n – розмір вхідної послідовності. Цей метод на фоні двох інших виглядає найперспективнішим через те, що його часова складність дорівнює $n * log_2(n)$ [63].

Цей метод буде використаний для розрахунків в полі: алгоритм для розрахунків в області комплексних чисел відрізняється від алгоритму для розрахунків в полі лише тим, що число $e^{-1*(\frac{2*\pi*i}{n})}$ замінюється на елемент α^i групи, що утворює циклічну підгрупу мультиплікативної групи [64].

2.3 Відомості про забезпечення результатами комп'ютерного експерименту заданих точності та достовірності (вірогідності)

Для забезпечення результатами комп'ютерного експерименту заданих точності та достовірності (вірогідності) слід використати наступні поняття [65]:

Точністю оцінки характеристики Θ^* називають величину ε відносно

$$|\Theta^* - M\lfloor\Theta\rfloor| < \varepsilon,$$

де $M\lfloor\Theta\rfloor -$ математичне очікування випадкової величини.

Величина ε являє собою абсолютне значення помилки у визначенні значення шуканої характеристики [65].

Достовірність оцінки характеристики Θ^* називають імовірність α того, що задана точність досягається [65]:

$$P(|\Theta^* - M\lfloor\Theta\rfloor| < \varepsilon) = \alpha$$

де $M\lfloor\Theta\rfloor -$ математичне очікування випадкової величини [65].

Для розрахунку дисперсії генеральної сукупності (є невідома до початку тестування) була використана формула [65]:

$$S^2[a] = \frac{\sum_{i=1}^{N}(a_i - a)^2}{N}, \quad (2.3)$$

де a_i – незалежні значення показника ефективності при N вимірювань (прогонах), а a – середньоарифметичне значення за N вимірювань.

Ця формула відповідає функції excel ДИСП.Г.

Потім ця формула була використана для розрахунку довірчого інтервалу генеральної сукупності за формулою:

$$\mathrm{L} = (1 - 0{,}95) * \frac{\sqrt{S^2[a]}}{\sqrt{N}}, \tag{2.4}$$

Ця формула відповідає функції excel ДОВЕРИТ.НОРМ.

Тобто, це формула точності оцінки (у секундах), яка потім використана у формулі:

$$\frac{L}{t_{\text{ср час}}} * 100\%,$$

де $t_{\text{ср час}}$ – середній час генеральної сукупності.

За замовчуванням, було прийнято рішення, що достовірність оцінки буде дорівнювати 95% (або 0,95 у долях одиниці), а точність не буде перевищувати 5% (або 0,05 у долях одиниці).

Дані про середній час роботи програми збиралися у таблиці excel (тобто збиралися a_i), потім засобами excel розраховувався середній час роботи програми (тобто a), а потім вже були використані вищеописані функції excel (2.3, 2.4) задля того, щоб при заданій достовірності 95% досягалася точність не вище 5%.

Далі ці поняття будуть використані у роботі з ціллю провести тестування трьох програмних реалізацій арифметизації із заданою достовірністю не більше 95% та точність не більше 5%.

2.4 Формулювання задачі та приклад її вирішення

Нехай за умовою задачі нам дано:

Поле Z_{13} = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} і воно складається з $|Z_{13}| = 13$ чисел зі складанням і множенням за модулем 13 (вхідні дані в програму). Це поле має мультиплікативну підгрупу G, таку, що |G| =6 (вхідні дані в програму) і

породжувальний елемент групи g = 4 (вхідні дані в програму). Існування такої підгрупи гарантоване, оскільки 6 ділить розмір цієї групи (який дорівнює 12) без залишку [7].

Припустімо, що розглянуте твердження про необхідність перевірки обчислювальної цілісності транзакції звучить так: «Перевіряючий має послідовність із 6 чисел, усі з яких це числа Фібоначчі».

Цю послідовність чисел Фібоначчі необхідно перевірити, прочитавши істотно менше 6 чисел [7]. Ця задача має такий розв'язок:

Послідовність чисел Фібоначчі формально визначається так (це вхідні дані в програму):

$a_0 = 1$;

$a_1 = 1$;

$a_{n+2} = (a_{n+1} + a_n) \bmod 13$.

Можливо створити слід виконання, записавши поспіль усі 6 чисел Фібоначчі: 1, 1, 2, 3, 5, 8. Поліноміальні обмеження можуть мати вигляд [7, 55]:

$$\begin{cases} A_0 - 1 = 0 \text{ та } A_1 - 1 = 0, \\ \forall\ 0 \le i < 4\text{: } A_{i+2} - A_{i+1} - A_i = 0, \\ A_5 - 8 = 0. \end{cases}$$

Тепер приведемо поліноміальні обмеження до поліноміального вигляду:

Рекурентне співвідношення Фібоначчі втілює набір обмежень на весь слід виконання, і його можна альтернативно інтерпретувати так [7]:

$$\forall\ 0 \le i < 4\text{: } f\left(g^{i+2}\right) - f\left(g^{i+1}\right) - f\left(g^{i}\right) = 0\ ,$$

Тепер Верифікатор може створити поліном композицію за формулою [7]:

$$q(x) = \frac{f\left(g^{i+2}\right) - f\left(g^{i+1}\right) - f\left(g^{i}\right)}{\prod_{i=0}^{3}(x - g^{i})}\ ,$$

Обчислення цього виразу для особливого випадку, коли ступені g утворюють підгрупу, може бути виконано так [7]:

$$x^{|G|} - 1 = \prod_{g \in G} (x - g),$$

Ця рівність є правильною, оскільки обидві сторони є многочленами ступеня |G|, корені яких у точності є елементами G [7]. А фактичний знаменник розглянутого композиційного полінома Фібоначчі можна отримати, переписавши його у вигляді [7]:

$$\frac{f(g^{i+2}) - f(g^{i+1}) - f(g^i)}{\prod_{i=0}^{3}(x - g^i)} = \frac{(w - g^4) * (w - g^5) * [f(g^2 * w) - f(g^1 * w) - f(w)]}{w^6 - 1},$$

де $w \in \{1, g^1, g^2, g^3, g^4, g^5\}$.

Ця послідовність обчислень арифметизації запрограмована на C^{++} та використана для тестування першого етапу роботи протоколу ZK-STARK.

3 ПРИКЛАДИ ПРОГРАМНОЇ РЕАЛІЗАЦІЯ ЕТАПУ ПРОТОКОЛУ ZK-STARK «АРИФМЕТИЗАЦІЯ»

3.1 Програмна реалізація на основі методу Гаусу

3.1.1 Загальні відомості

Найменування програми: «Консольна програма для моделювання арифметизації протоколу ZK-STARK».

Позначення програми – КПМАП-ZKSTARK-01, або, скорочено, AZkS1.

Позначення: КПМАП-ZKSTARK-01 версії 0.2.0.

Розробник: Аверков О.Ю.

Тел. 0989198104

E-mail: olegaverkov072@gmail.com

Програмне забезпечення необхідне для роботи програми.

Програма AZkS1 функціонує під керуванням операційної системи не нижче Windows 10. Додатково потрібне встановлення середовища розроблення не нижче Visual Studio 2022, створення проекту та підключення до проекту бібліотеки NTL версії 11.5.

Мова програмування програми.

Програму AZkS1 розроблено та реалізовано мовою програмування C^{++} у середовищі розробки Visual Studio 2022.

3.1.2 Функціональне призначення програми

«Консольна програма для моделювання арифметизації протоколу ZK-STARK» призначена для створення та візуалізації роботи протоколу STARK за умови, що сутність Прувер не намагається обманути сутність Веріфікатор.

Ключові функції програми:

1) Введення числа, що позначає основи поля і має сенс модуля, за яким проходять обчислення в програмі та позначає кількість вхідних чисел Фібоначчі.
2) Введення числа, що позначає первісний корінь групи.
3) Введення числа, що позначає довжину підгрупи.

4) Проведення послідовності розрахунків арифметизації.

5) Відображення полінома ступеня менше двох.

6) Вихід із програми.

Відомості про функціональні обмеження програми на застосуванні:

- Обсяг пам'яті, який займає програма під час функціонування, залежить від довжини підгрупи. Що більшою є довжина підгрупи, то більшою є квадратна матриця для пошуку коренів інтерполяційного полінома (залежність обсягу пам'яті, який займає програма, від довжини підгрупи має нелінійний характер). Програма AZkS1 потребує для функціонування щонайменше 10 Мега байтів вільної оперативної пам'яті, за меншої кількості вільної пам'яті програма може не функціонувати.

- Програма AZkS1 вимагає введення тільки натуральних цілих чисел, можливе введення також нуля. Діапазон обмежується можливостями бібліотеки NTL.

3.1.3 Опис логічної структури програми

Програма написана і виконується з текстом програми мовою C^{++}.

Алгоритм програми та використовувані методи.

Програма обробляє одержувані дані через меню керування командами в консолі користувача. Усі параметри вводиться в тіло програми.

Методи, що використовуються, ґрунтуються на можливостях апаратних модулів технічного засобу, на якому запущено Програму.

Алгоритм:

1) Запуск програми

2) Створення компонентів (змінні, вектори, масиви)

3) Ініціалізація основних змінних користувачем

4) Обробка введених даних

5) Візуалізація даних.

Ключові методи:

1) Створення послідовності чисел Фібоначчі у вигляді масиву (послідовність вхідних чисел)

2) Створення матриці трикутникового вигляду для вирішення СЛАР

3) Зворотна підстановка для розв'язання трикутної системи

4) Вирішенні СЛАР (знаходження коефіцієнтів інтерполяційного поліному f(x))

5) Виконання послідовності розрахунків етапу арифметизація протоколу ZK-STARK (помноження інтерполяційного поліному на спеціальний коефіцієнт g^i, створення рекурентного співвідношення, помноження поліному із рекурентного співвідношення на спеціальний поліном, а потім ділення поліномів за умовами арифметизації).

6) Виведення поліному-результату на консоль.

Структурний опис програми із її складовими можливо подати у вигляді рисунка 3.1.

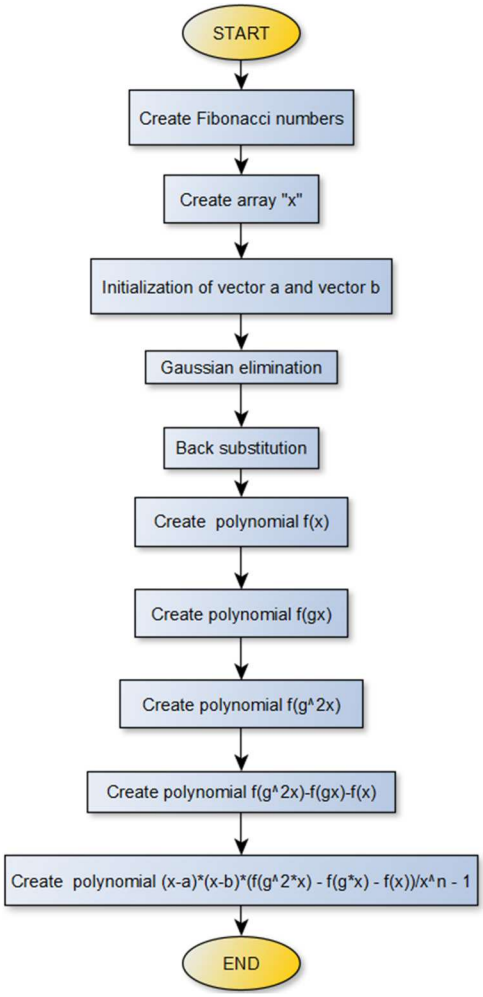

Рисунок 3.1 – Структура програми AZkS1

Кожна функція, яка показана на рисунку 3.1 виконується тільки за умови виконання попередньої функції, тобто всі функції виконуються послідовно.

Зв'язки програми з іншими програмами.

Програма не пов'язана з іншими програмами. Програмі не потрібен доступ в інтернет.

3.1.4 Технічні засоби, що використовуються

Програма встановлюється на обладнанні з такими мінімальними характеристиками:

- CPU: 13th Gen Intel(R) Core(TM) i7-13620H (16 CPUs), ~2.4GHz;

- RAM: - 32 Gb;

- об'єм диска - 750 Gb (залежить від планованої кількості АТТ);

- операційна система Windows 11.

3.1.5 Виклик і завантаження

Після встановлення на комп'ютер, запуск програми AZkS1 проводиться вручну після завантаження операційної системи. Спочатку з архіву відкривається проект, потім відкривається файл, прикріплений до проекту. Бажано відкривати файл проекту за допомогою Visual Studio 2022. Після відкриття проекту, обов’язково потрібно вимкнути засіб діагностики ПЗ через те, що він значно уповільнює час виконання програми.

3.1.6 Опис вхідних і вихідних даних

Вхідні дані. Вхідними даними для Програми є набір текстових рядків англійською мовою в кодуванні Windows-1251. Текстові рядки розділяються символами кінця рядка. Попередньої підготовки команд інтерфейсу командного рядка не потрібно.

Вихідні дані. Вихідними даними для Програми є:

- текстова інформація, що виводиться на консолі (результати роботи програми);

-файли в формату .csv із інформацією про час роботи програми.

Повний код програми наведено в додатку (Додаток А).

3.2 Програмна реалізація на основі методу обернених матриць

3.2.1 Загальні відомості

Позначення та найменування програми.

Найменування програми: «Консольна програма для моделювання арифметизації протоколу ZK-STARK».

Позначення програми - КПМАП-ZKSTARK-02, або, скорочено, AZkS2.

Позначення: КПМАП-ZKSTARK-02 версії 0.6.0.

Розробник: Аверков О.Ю.

Тел. 0989198104

E-mail: olegaverkov072@gmail.com

Програмне забезпечення необхідне для роботи програми.

Програма AZkS2 функціонує під керуванням операційної системи не нижче Windows 10. Додатково потрібне встановлення середовища розробки не нижче Visual Studio 2022, створення проекту і підключення до проекту бібліотеки NTL версії 11.5.

Мова програмування програми.

Програму AZkS2 розроблено та реалізовано мовою програмування C^{++} у середовищі розробки Visual Studio 2022.

3.2.2 Функціональне призначення програми

«Консольна програма для моделювання арифметизації протоколу ZK-STARK» призначена для моделювання етапу арифметизації роботи протоколу ZK-STARK за умови, що сутність Прувер не намагається обманути сутність Веріфікатор.

Ключові функції програми:

1) Введення числа, що позначає основи поля і має сенс модуля, за яким проходять обчислення в програмі та позначає кількість вхідних чисел Фібоначчі.
2) Введення числа, що позначає первісний корінь групи.
3) Введення числа, що позначає довжину підгрупи.
4) Проведення послідовності розрахунків арифметизації.
5) Відображення полінома ступеня менше двох.
6) Вихід із програми.

Відомості про функціональні обмеження програми на застосуванні:

- Обсяг пам'яті, який займає програма під час функціонування, залежить від довжини підгрупи. Що більшою є довжина підгрупи, то більшою є квадратна матриця для пошуку коефіцієнтів інтерполяційного полінома (залежність обсягу пам'яті, який займає програма, від довжини підгрупи має нелінійний характер). Програма AZkS2 потребує для функціонування щонайменше 10 Мега байтів вільної оперативної пам'яті, за меншої кількості вільної пам'яті програма може не функціонувати.

- Програма AZkS2 вимагає введення тільки натуральних цілих чисел, можливе введення також нуля. Діапазон обмежується можливостями бібліотеки NTL.

3.2.3 Опис логічної структури програми

Програма написана і виконується з текстом програми мовою C^{++}.

Алгоритм програми та використовувані методи.

Програма обробляє одержувані дані через меню керування командами в консолі користувача. Усі параметри вводиться в тіло програми.

Методи, що використовуються, ґрунтуються на можливостях апаратних модулів технічного засобу, на якому запущено Програму.

Алгоритм:

1) Запуск програми
2) Створення компонентів (змінні, вектори, масиви)
3) Ініціалізація основних змінних користувачем
4) Обробка введених даних
5) Візуалізація даних.

Ключові методи:

1) Створення послідовності чисел Фібоначчі у вигляді масиву (послідовність вхідних чисел)
2) Створення матриці для вирішення СЛАР
3) Створення оберненої матриці для вирішення СЛАР
4) Вирішенні СЛАР (знаходження коефіцієнтів інтерполяційного поліному

f(x))

5) Виконання послідовності розрахунків етапу арифметизація протоколу ZK-STARK (помноження інтерполяційного поліному на спеціальний коефіцієнт g^i, створення рекурентного співвідношення, помноження поліному із рекурентного співвідношення на спеціальний поліном, а потім ділення поліномів за умовами арифметизації).

6) Виведення поліному-результату на консоль.

Структурний опис програми можливо подати у вигляді рисунка 3.2.

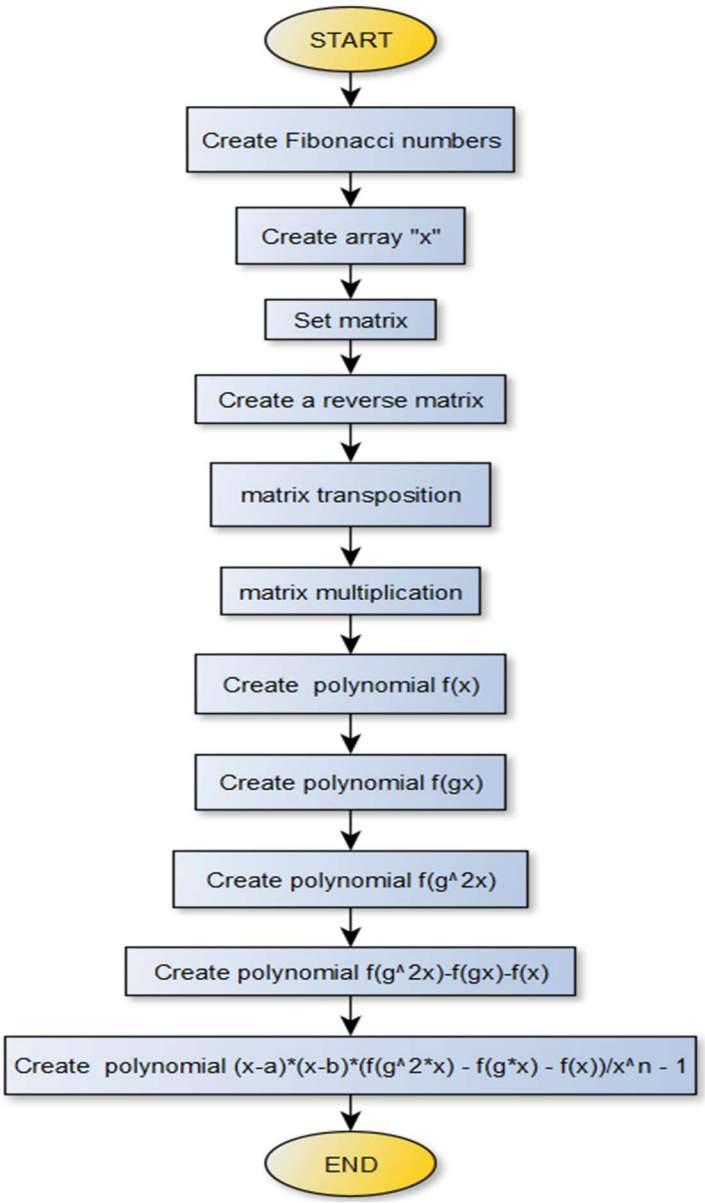

Рисунок 3.2 – Структура програми AZkS2

Кожна функція, яка показана на рисунку 3.2 виконується тільки за умови виконання попередньої функції, тобто всі функції виконуються послідовно.

Зв'язки програми з іншими програмами.

Програма не пов'язана з іншими програмами. Програмі не потрібен доступ в інтернет.

3.2.4 Технічні засоби, що використовуються

Програма встановлюється на обладнанні з такими мінімальними характеристиками:

- CPU: 13th Gen Intel(R) Core(TM) i7-13620H (16 CPUs), ~2.4GHz;
- RAM: - 32 Gb;
- об'єм диска - 750 Gb (залежить від планованої кількості АТТ);
- операційна система Windows 11.

3.2.5 Виклик і завантаження

Після встановлення на комп'ютер, запуск програми AZkS2 проводиться вручну після завантаження операційної системи. Спочатку з архіву відкривається проект, потім відкривається файл, прикріплений до проекту. Бажано відкривати файл проекту за допомогою Visual Studio 2022. Після відкриття проекту, обов’язково потрібно вимкнути засіб діагностики ПЗ через те, що він значно уповільнює час виконання програми.

3.2.6 Опис вхідних і вихідних даних

Вхідні дані. Вхідними даними для Програми є набір текстових рядків англійською мовою в кодуванні Windows-1251. Текстові рядки розділяються символами кінця рядка. Попередньої підготовки команд інтерфейсу командного рядка не потрібно.

Вихідні дані. Вихідними даними для Програми є:

- текстова інформація, що виводиться на консолі (результати роботи програми);

-файли в формату .csv із інформацією про час роботи програми.

Повний код програми наведено в додатку (Додаток Б).

3.3 Програмна реалізація на основі методу швидкого перетворення Фур’є

3.3.1 Загальні відомості

Найменування програми: «Консольна програма для моделювання арифметизації протоколу ZK-STARK».

Позначення програми - КПМАП-ZKSTARK-03, або, скорочено, AZkS3.

Позначення: КПМАП-ZKSTARK-03 версії 0.2.1.

Розробник: Аверков О.Ю.

Тел. 0989198104

E-mail: olegaverkov072@gmail.com

Програмне забезпечення необхідне для роботи програми.

Програма AZkS3 функціонує під керуванням операційної системи не нижче Windows 10. Додатково потрібне встановлення середовища розроблення не нижче Visual Studio 2022, створення проекту та підключення до проекту бібліотеки NTL версії 11.5.

Мова програмування програми.

Програму AZkS3 розроблено та реалізовано мовою програмування C^{++} у середовищі розробки Visual Studio 2022.

3.3.2 Функціональне призначення програми

«Консольна програма для моделювання арифметизації протоколу ZK-STARK» призначена для створення та візуалізації роботи протоколу STARK за умови, що сутність Прувер не намагається обманути сутність Веріфікатор.

Ключові функції програми:

1) Введення числа, що позначає основи поля і має сенс модуля, за яким проходять обчислення в програмі та позначає кількість вхідних чисел Фібоначчі.
2) Введення числа, що позначає первісний корінь групи.
3) Введення числа, що позначає довжину підгрупи.
4) Проведення послідовності розрахунків арифметизації.
5) Відображення полінома ступеня менше двох.
6) Вихід із програми.

Відомості про функціональні обмеження програми на застосуванні:

- Обсяг пам'яті, який займає програма під час функціонування, залежить від

довжини підгрупи. Що більшою є довжина підгрупи, то більшою є квадратна матриця для пошуку коренів інтерполяційного полінома (залежність обсягу пам'яті, який займає програма, від довжини підгрупи має нелінійний характер). Програма AZkS3 потребує для функціонування щонайменше 10 Мега байтів вільної оперативної пам'яті, за меншої кількості вільної пам'яті програма може не функціонувати.

- Програма AZkS3 вимагає введення тільки натуральних цілих чисел, можливе введення також нуля. Діапазон обмежується можливостями бібліотеки NTL.

3.3.3 Опис логічної структури програми

Програма написана і виконується з текстом програми мовою C^{++}.

Алгоритм програми та використовувані методи.

Програма обробляє одержувані дані через меню керування командами в консолі користувача. Усі параметри вводиться в тіло програми.

Методи, що використовуються, ґрунтуються на можливостях апаратних модулів технічного засобу, на якому запущено Програму.

Алгоритм:

1) Запуск програми
2) Створення компонентів (змінні, вектори, масиви)
3) Ініціалізація основних змінних користувачем
4) Обробка введених даних
5) Візуалізація даних.

Ключові методи:

1) Створення послідовності чисел Фібоначчі у вигляді масиву (послідовність вхідних чисел)

2) Вирішенні СЛАР за допомогою швидкого зворотного перетворення Фур’є (знаходження коефіцієнтів інтерполяційного поліному f(x))

3) Виконання послідовності розрахунків етапу арифметизація протоколу ZK-STARK (помноження інтерполяційного поліному на спеціальний коефіцієнт g^i, створення рекурентного співвідношення, помноження поліному із

рекурентного співвідношення на спеціальний поліном, а потім ділення поліномів за умовами арифметизації).

4) Виведення поліному-результату на консоль.

Структурний опис програми із її складовими можливо подати у вигляді рисунка 3.3.

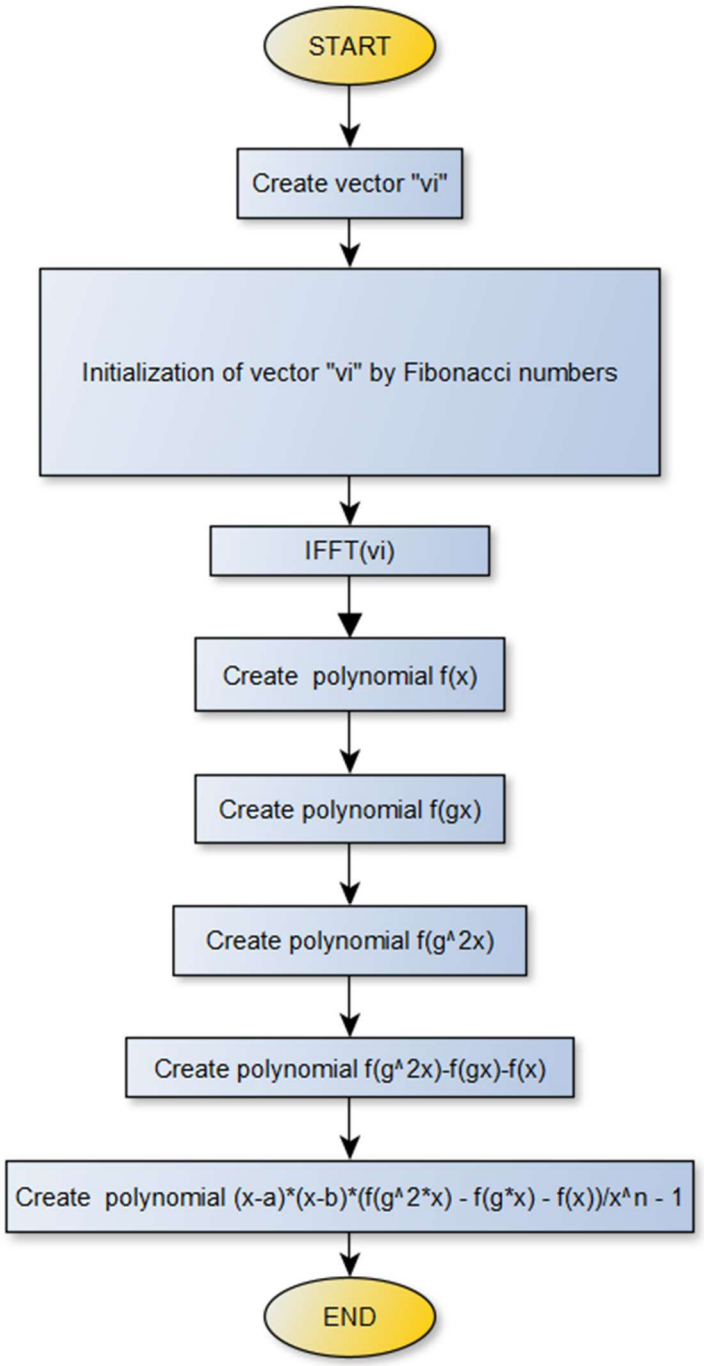

Рисунок 3.3 – Структура програми AZkS3

Кожна функція, яка показана на рисунку 3.3 виконується тільки за умови виконання попередньої функції, тобто всі функції виконуються послідовно.

Зв'язки програми з іншими програмами.

Програма не пов'язана з іншими програмами. Програмі не потрібен доступ в інтернет.

3.3.4 Технічні засоби, що використовуються

Програма встановлюється на обладнанні з такими мінімальними характеристиками:

- CPU: 13th Gen Intel(R) Core(TM) i7-13620H (16 CPUs), ~2.4GHz;

- RAM: - 32 Gb;

- об'єм диска - 750 Gb (залежить від планованої кількості АТТ);

- операційна система Windows 11.

3.3.5 Виклик і завантаження

Після встановлення на комп'ютер, запуск програми AZkS3 проводиться вручну після завантаження операційної системи. Спочатку з архіву відкривається проект, потім відкривається файл, прикріплений до проекту. Бажано відкривати файл проекту за допомогою Visual Studio 2022. Після відкриття проекту, обов’язково потрібно вимкнути засіб діагностики ПЗ через те, що він значно уповільнює час виконання програми.

3.3.6 Опис вхідних і вихідних даних

Вхідні дані. Вхідними даними для Програми є набір текстових рядків англійською мовою в кодуванні Windows-1251. Текстові рядки розділяються символами кінця рядка. Попередньої підготовки команд інтерфейсу командного рядка не потрібно.

Вихідні дані. Вихідними даними для Програми є:

- текстова інформація, що виводиться на консолі (результати роботи програми);

-файли в формату .csv із інформацією про час роботи програми.

Повний код програми наведено в додатку (Додаток В).

3.4 Оцінка ефективності роботи програмних реалізацій першого етапу роботи протоколу ZK-STARK на домашньому комп’ютері

Оцінку часової складності арифметизації на основі методів обернених матриць, методу Гаусу та зворотного перетворення Фур’є проводили за результатами тестування їх програмних реалізацій

Згідно результатів тестування, було визначено, що метод швидкого зворотного перетворення Фур’є (ШЗПФ) є найефективнішим для вирішення СЛАР з великою кількістю невідомих та рівнянь, тобто цей метод здатній вирішувати СЛАР з кількістю невідомих та рівнянь, яка майже в 16578 разів більша ніж кількість невідомих та рівнянь у методу Гаусу (Г) та методу Зворотних матриць (ЗМ). При використанні методу ШЗПФ для проведення процедури інтерполяції є можливим подати на вхід в 16578 чисел Фібоначчі більше (максимально 2^{24} чисел), ніж можна подати на вхід при використанні методів Г та ЗМ для вирішення СЛАР (максимально 1012), при умові, що СЛАР вирішуються за «розумний час».

Найбільшу ефективність серед трьох методу ШЗПФ можна проілюструвати за допомогою графіку, що поданий на рисунку 3.4.

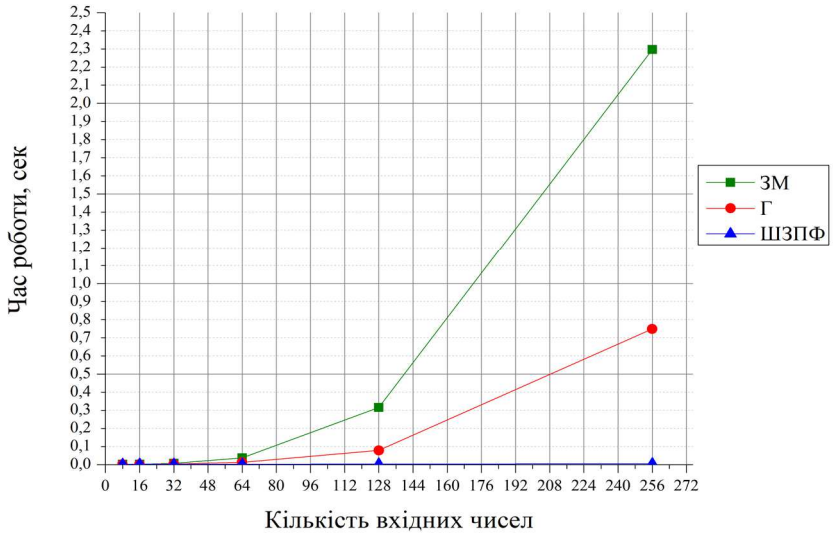

Рисунок 3.4 – Залежність часу роботи арифметизації на основі методів ЗМ, Г, ШЗПФ від кількості вхідних чисел Фібоначчі

На графіку вісь Ox означає середній за 1000 вимірювань час роботи програми (при цьому довірча вірогідність (достовірність) становить 95%, а точність не перевищує 3% [65]) , вісь Oy означає кількість вхідних в програму чисел Фібоначчі, синім кольором позначено час роботи варіанту арифметизації, який використовує метод ЗМ для проведення процедури інтерполяції, зеленим кольором позначено час роботи варіанту арифметизації, який використовує метод Г для проведення процедури інтерполяції та червоним кольором позначено час роботи варіанту арифметизації, який використовує метод ШЗПФ для проведення процедури інтерполяції. При цьому, розрахунки у варіантах арифметизації, які використовують метод Г та ЗМ для інтерполяції проходять у полі GF (257), а розрахунки у варіанту арифметизації, який використовує метод ШЗПФ проходять у полі Златовласки, тобто у GF ($2^{64} - 2^{32} + 1$). Тобто розрахунки за допомогою ШЗПФ для 256 вхідних чисел Фібоначчі (256 невідомих та 256 рівнянь) проходять, як мінімум в 176 разів швидше при умові, що величини невідомих чисел в полі Златовласки більші в $\frac{2^{64} - 2^{32} + 1}{257}$ разів ніж в полі GF (257).

Із тестів було визначено, що, якщо варіанту арифметизації, який використовує метод Г або ЗМ подати на вхід 8 числа Фібоначчі, але розрахунки проводити у полі Златовласки, тобто за модулем $P = 2^{64} - 2^{32} + 1$, то цей варіант працював 40 хвилин без результату, тобто це в 1 545 946 разів довше, ніж варіант з результатом з ШЗПФ.

Тестування показало, що максимальна кількість чисел Фібоначчі, яку можна подати на вхід становить 2^{24} чисел, тобто за допомогою методу ШЗПФ можливо вирішувати СЛАР розміром 2^{24} невідомих та 2^{24} рівнянь за середній за 100 вимірювань час 177,45 секунди, при цьому достовірність становить 95%, а точність не перевищує 3% [65]. Більшу кількість вхідних чисел подати на вхід не вдалося через обмежені можливості бібліотеки NTL.

За допомогою методу лінійної регресії (рис. 3.5) було показано, що запрограмований варіант арифметизації, який використовує метод ШЗПФ для інтерполяції дійсно має часову складність $O(n * log_2 n)$.

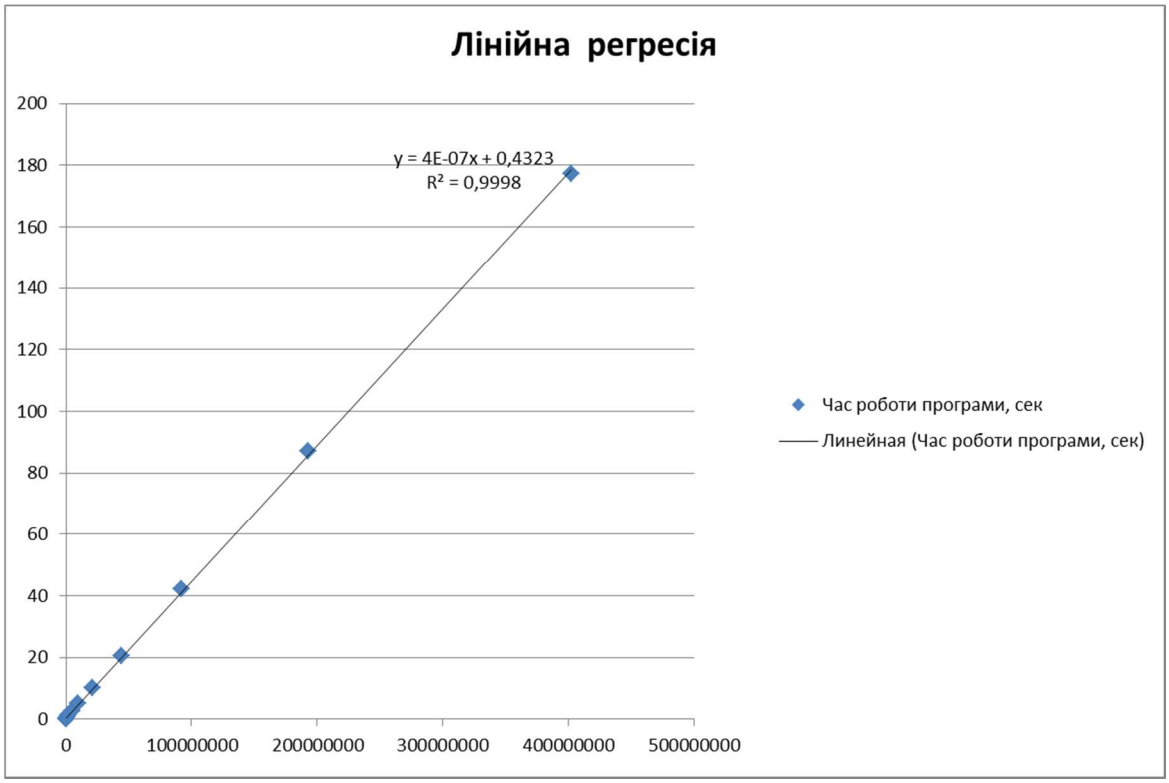

Рисунок 3.5 – Лінійна регресія арифметизації, яка використовує метод ШЗПФ для інтерполяції

На осі Ox показані числа $n * log_2 n$ (де n – кількість вхідних чисел Фібоначчі), а на осі Oy показан загальний час роботи етапу Арефметизації. Тобто тестування підтверджує, що використання методу ШЗПФ призводить до прискорення роботи етапу арифметизації у порівнянні з варіантами, які використовують методи Г та ЗМ для інтерполяції. Тобто арифметизація із ШЗПФ має часову складність майже O($n * log_2 n$).

На відміну від попереднього випадку, де було використано ШЗПФ для проведення інтерполяції, за допомогою поліноміальної регресії за результатами тестування було виявлено, що часова складність арифметизації, яка використовує метод обернених матриць та метод Гаусу для інтерполяції, становить O(n^3).

Це можна проілюструвати двома графіками, що подані на рисунках 3.6 та 3.7.

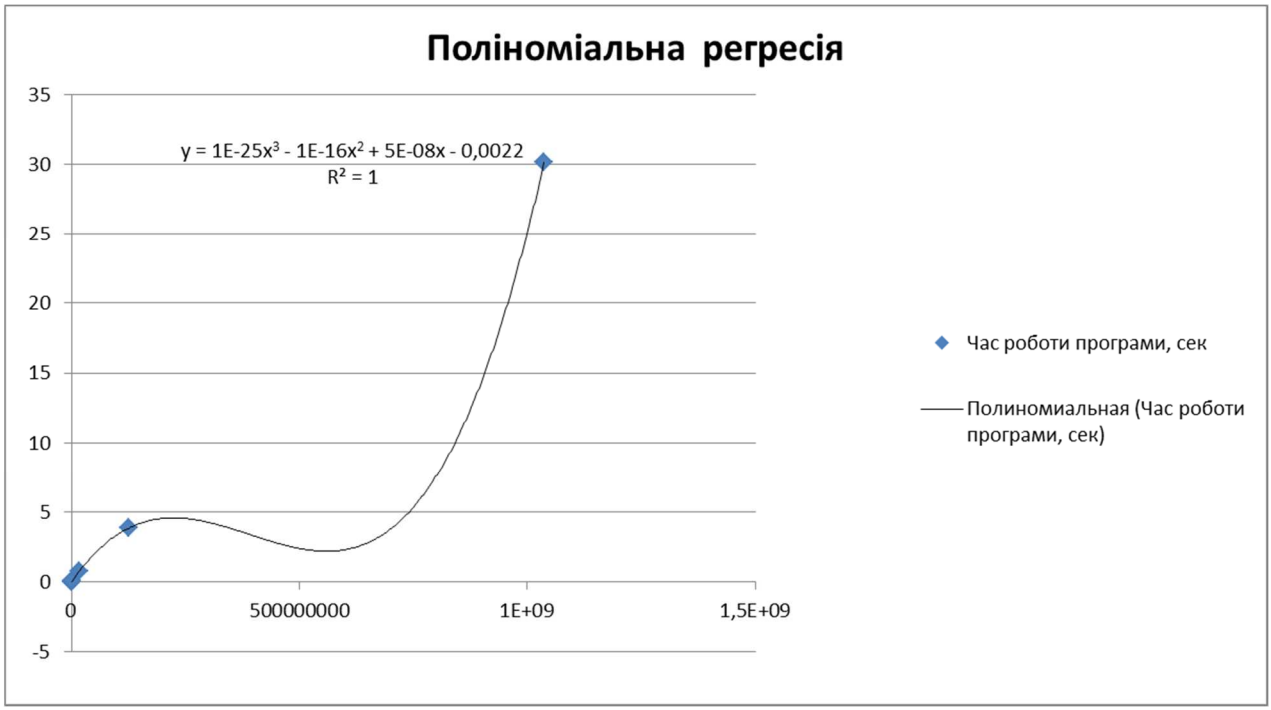

Рисунок 3.6 – Поліноміальна регресія арифметизації, яка використовує метод Гаусу для інтерполяції

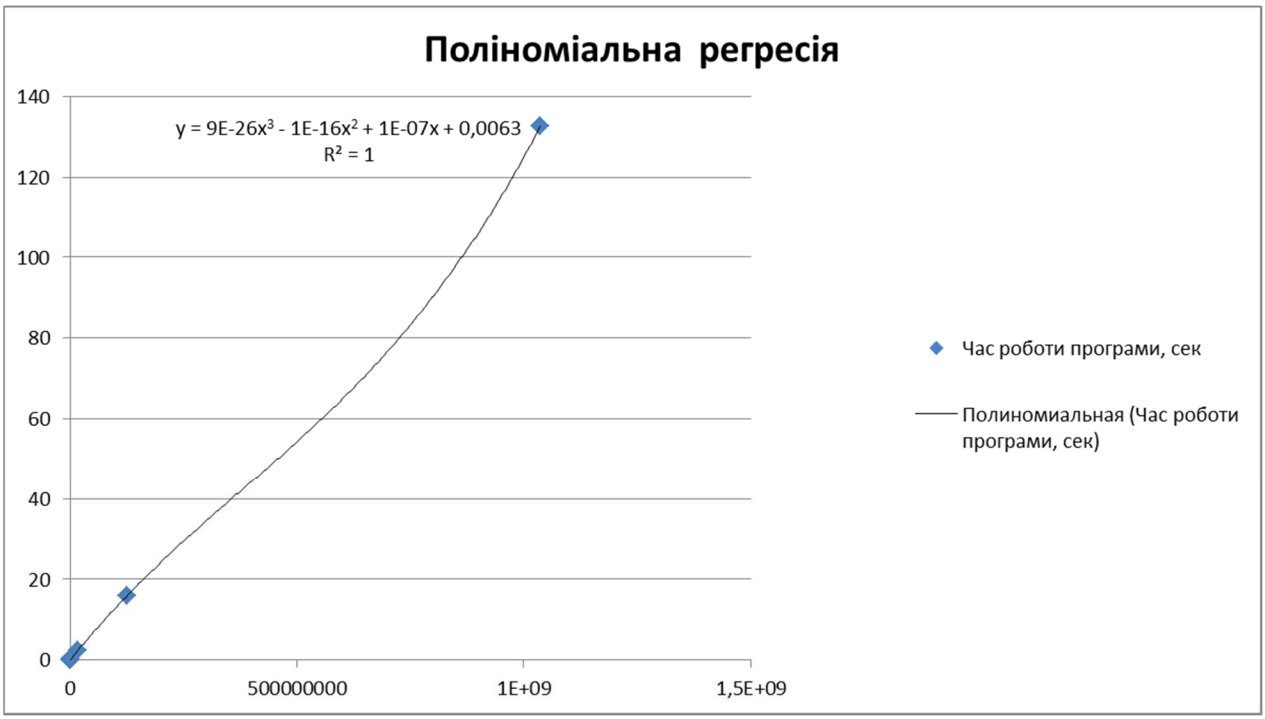

Рисунок 3.7 – Поліноміальна регресія арифметизації, яка використовує метод обернених матриць для інтерполяції для інтерполяції

Було виявлено, що математична часова складність алгоритмів ШЗПФ, Г та ОМ співпадає із часовою складністю, яка була утримана через аналіз результатів тестування моделей арифметизації, які використовують вищеописані методи для проведення процедури інтерполяції.

Результати перевірки фактичної часової складності моделей арифметизації свідчать про те, що використовуючи метод ШЗПФ для прискорення роботи Прувера в арифметизації, можна прискорити в $\frac{n^3}{n*log_2 n}$ разів час перевірки на правильність транзакції мережі блокчейн та зекономити часові та апаратні ресурси мережі.

За допомогою ШЗПФ вдалося зменшити час перетворення вхідних даних у поліном та ще зменшити відсоток часу, який займає процес перетворення вхідних даних на поліном (процедура Інтерполяції) від часу роботи всього першого етапу, тобто завдяки ШЗПФ відсоток часу зменшився із 99,99 % (для 1012 вхідних чисел Фібоначчі та Г та ОМ) до 58,56% (для 2^{24} чисел Фібоначчі та ШЗПФ).

Це можна проілюструвати графіком, що подано на рисунку 3.8.

Рисунок 3.8 – Залежність відсотка часу, який займає процедура Інтерполяції за допомогою ШЗПФ від кількості вхідних чисел Фібоначчі

За графіком, можна констатувати, що спочатку при збільшенні кількості вхідних чисел Фібоначчі, відсоток часу, який займає процедура Інтерполяції за допомогою ШЗПФ, збільшується, а потім виходить на «плато», не перевищуючи відмітку в 60%.

Для випадку, коли арифметизація використовує методи Гаусу для інтерполяції цей графік має наступний вигляд (рис. 3.9).

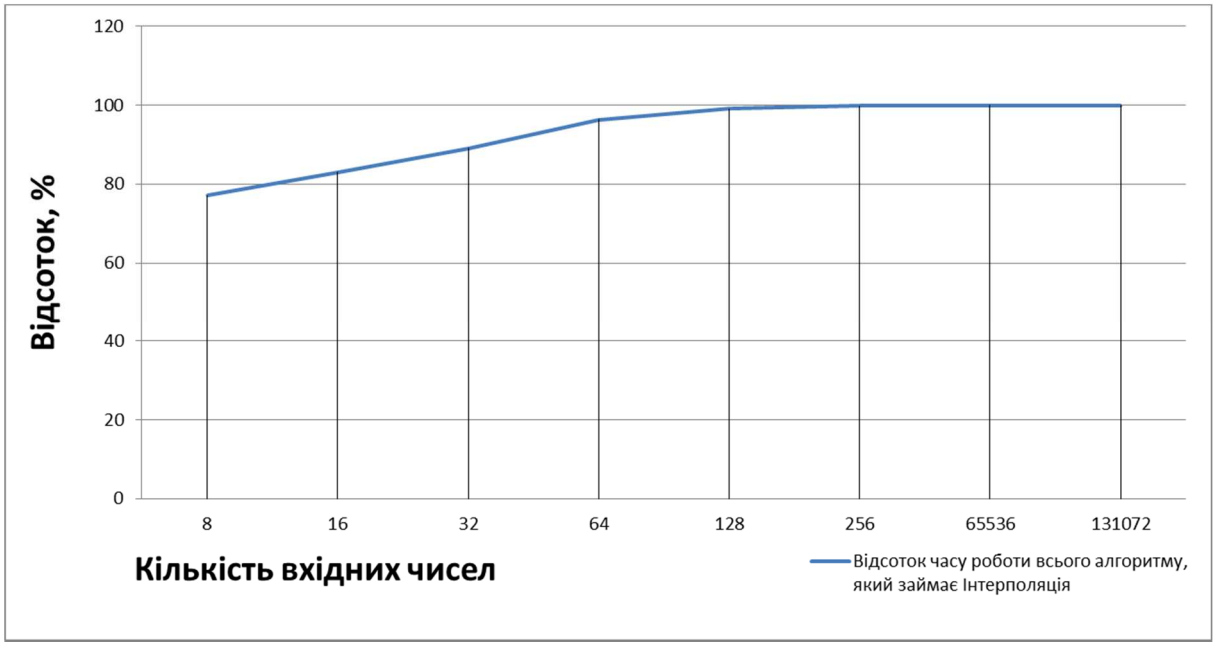

Рисунок 3.9 – Залежність відсотка часу, який займає процедура Інтерполяції за допомогою метода Гаусу від кількості вхідних чисел Фібоначчі

За графіком, можна зробити висновок, що при збільшенні кількості вхідних чисел Фібоначчі, відсоток часу, який займає процедура Інтерполяції за допомогою методу Гаусу, збільшується, а потім виходить на «плато», наближуючись майже до 100%.

Це свідчить про те, що інтерполяція за допомогою метода Г проходить довше у відсотковому відношенні ніж за допомогою ШЗПФ.

Для випадку, коли арифметизація використовує метод обернених матриць для інтерполяції цей графік має наступний вигляд (рис. 3.10).

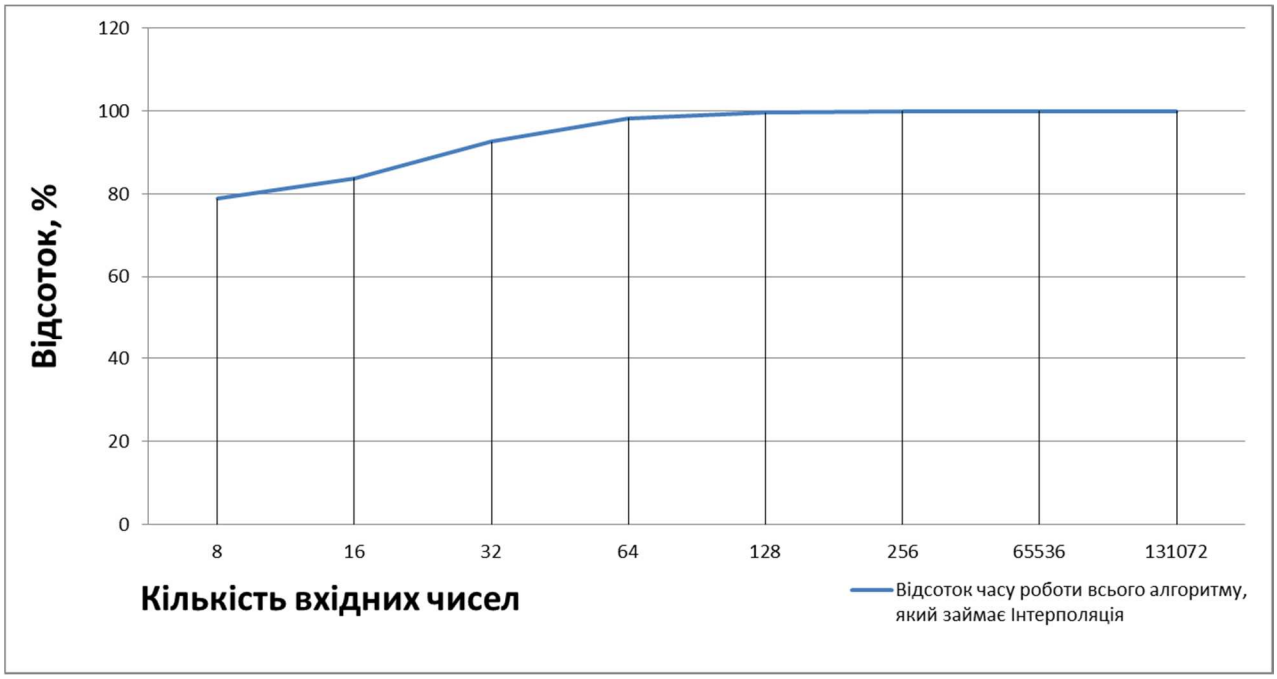

Рисунок 3.10 – Залежність відсотка часу, який займає процедура Інтерполяції за допомогою метода методу обернених матриць від кількості вхідних чисел Фібоначчі

За графіком, можна зробити висновок, що при збільшенні кількості вхідних чисел Фібоначчі, відсоток часу, який займає процедура Інтерполяції за допомогою обернених матриць, збільшується, а потім виходить на «плато», наближуючись майже до 100%.

Тобто, якщо при використанні для Інтерполяції методу ОМ діаграма, що показує відсоток часу, який займає процедура інтерполяції (діяльність сутності Прувер) та який займає процедура веріфікації (діяльність сутності Веріфікатор) від часу роботи всієї програми виглядає таким чином (рис. 3.11).

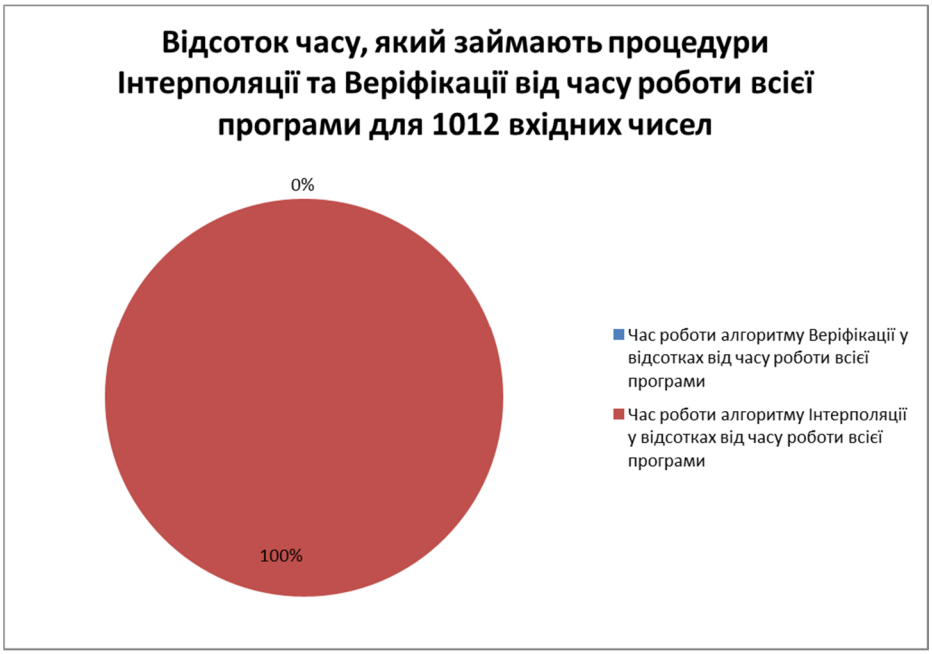

Рисунок 3.11 – Відсоток часу, який займають процедури Інтерполяції та Веріфікації від часу роботи всієї програми для 1012 вхідних чисел (максимальної кількості) при використанні методу ОМ для Інтерполяції

Ця ж діаграма виглядає майже таким чином і для випадку арифметизації на основі методу Гаусу (рис. 3.12).

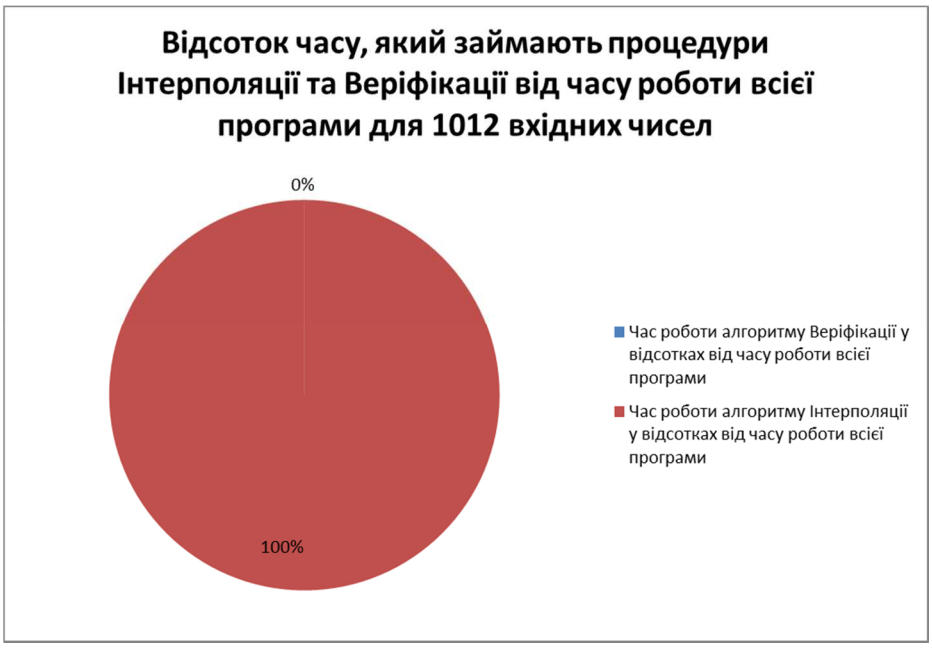

Рисунок 3.12 – Відсоток часу, який займають процедури Інтерполяції та Веріфікації від часу роботи всієї програми для 1012 вхідних чисел (максимальної кількості) при використанні методу Г для Інтерполяції

А ця ж діаграма для інтерполяції на основі ШЗПФ має наступний вигляд (рис. 3.13).

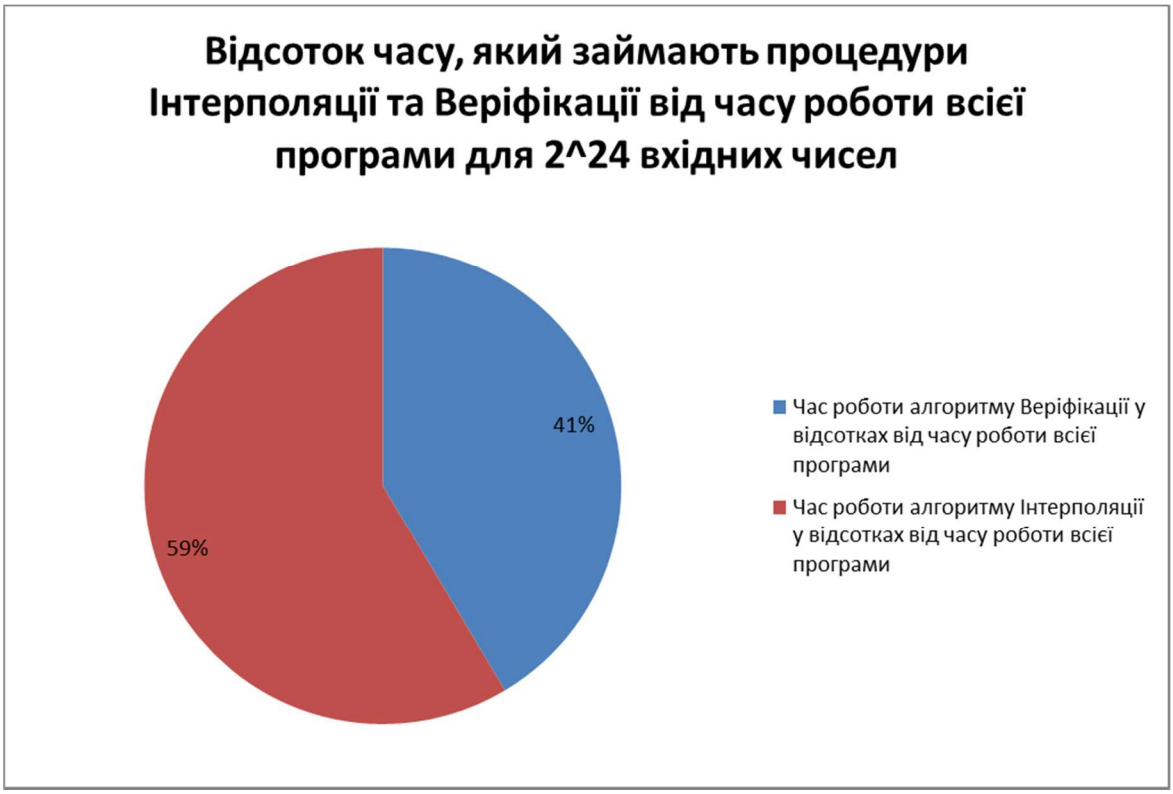

Рисунок 3.13 – Відсоток часу, який займають процедури Інтерполяції та Веріфікації від часу роботи всієї програми для 2^{24} вхідних чисел (максимальної кількості) при використанні методу ШЗПФ для Інтерполяції

Тобто, використовуючи метод ШЗПФ для Інтерполяції є можливим зменшити відсоток часу, який займає процедура Інтерполяції із 99,99% до 58,56% від часу роботи всього етапу арифметизації.

ВИСНОВКИ

1) Визначено актуальність, сучасний стан та перспективність розвитку криптографічного протоколу доказу з нульовим розголошенням ZK-STARK.

Актуальність протоколу ZK-STARK полягає у стійкості протоколу до атак з боку квантових комп'ютерів, досягає найбільшої масштабованості серед своїх конкурентів та є прозорим.

Аналіз сучасного стану протоколу ZK-STARK виявив, що вже існують такі перші початкові імплементації ZK-STARK у проектах другого рівня (Layer 2):

- StarkEx від StarkWare використовує технологію zk-STARK для підвищення ефективності транзакцій і зниження комісій за газ в Ethereum.

- dYdX використовує zk-STARKs від StarkEx для приватної, економічно ефективної торгівлі з використанням кредитного плеча та інших фінансових інструментів.

- Immutable X використовує zk-STARK для миттєвих, масштабованих і безгазових NFT-транзакцій, зберігаючи конфіденційність користувачів.

- Блокчейн Ethereum вирішую проблему масштабованості за допомогою ZK-STARK від Starknet.

Найбільш перспективним шляхом є використання протоколу ZK-STARK у світі блокчейн мереж, наприклад, криптовалют, де цей протокол має великий потенціал і може стати новаторським шляхом до широкого впровадження.

2) Надано загальну характеристику протоколу ZK-STARK, висвітлино функціонування протоколу ZK-STARK та показані особливості функціонування його першого етапу арифметизації.

А саме, показано, що, головними особливістями протоколу ZK-STARK є базування на характерних поняттях: масштабованість, прозорість та аргумент знання. До того масштабованість цього протоколу означає, що при збільшенні кількості даних або розміру завдання, час, необхідний для створення доказу, збільшується квазілінійно, а час веріфікації доказу збільшується

полілагорифмічно. Прозорість означає, що всі повідомлення Веріфікатора є випадковими числами, які генеруються публічно. Аргумент знання позначає криптографічний доказ, що підтверджує автентичність або право власності на дані чи транзакцію в блокчейн-мережі.

Висвітлено, що особливістю функціонування цього протоколу нульового розголошення є використання сутності «Прувер» та «Веріфікатор», які у процесі взаємодії доводять, чи спростовують доказ. У протоколі ZK-STARK Прувер та Веріфікатор виконують різні ролі з метою спростувати чи доказати достовірність транзікції блокчейн.

Показано, що перший етап реалізації протоколів ZK-STARK арифметизація складається з двох основних кроків. Визначено особливості функціонування першого етапу роботи протоколу ZK-STARK - етапу арифметизації: він починається із перетворення вхідних даних (наприклад, слід виконання транзакції Блокчейн) у поліном за допомогою інтерполяційного поліному. Потім цей поліном тестується на низьку ступінь. Якщо поліном має ступінь менше ніж два, то він проходить перевірку і вважається достовірним, і відповідно, є достовірними і вхідні дані (тобто транзакція, що перевіряється на достовірність). Потім, якщо у ролі вхідних даних виступає слід виконання транзакції, то достовірна транзакція приєднується до мережі Блокчейн.

3) Створині теоретична та програмні реалізації арифметизації першого етапу роботи протоколу ZK-STARK із застосуванням методу Гауса, методу обернених матриць та швидкого зворотного перетворення Фур'є.

Був створений приклад теоретичної реалізації першого етапу роботи протоколу ZK-STARK арифметизація, у якому пошук коефіцієнтів інтерполяційного поліному проводився за допомогою інтерполяційного поліному Лагранжа, що має часову складність $O(n^2)$, на основі послідовності математичних розрахунків, які виконуються під час арифметизації у протоколі ZK-STARK.

Потім на основі теоретичної реалізації були розроблені три програмні реалізації першого етапу протоколу арифметизація. Перша програмна реалізація - КПМАП-ZKSTARK-01, де пошук коефіцієнтів інтерполяційного поліному був

запрограмований за допомогою методу Гаусу (часова складність $O(n^3)$). Друга програмна реалізація КПМАП-ZKSTARK-02 використовує метод обернених матриць (часова складність $O(n^3)$ для пошуку коефіцієнтів інтерполяційного поліному. Третя програмна реалізація КПМАП-ZKSTARK-03 для пошуку коефіцієнтів інтерполяційного поліному використовує швидке зворотне перетворення Фур'є (часова складність $O(n * log_2 n)$.

4) Проведено тестування програмних реалізацій першого етапу арифметизація протоколу ZK-STARK та оцінені їх часові складності.

Аналіз часової складності коду C^{++} програмних реалізацій арифметизації на основі методу Гаусу та методу обернених матриць відповідає часовій складності цих програмних реалізацій, яка була отримана при використанні результатів тестування програмних реалізацій та при використання методу поліноміальної регресії і відповідає значенню $O(n^3)$. Це значення часової складності також відповідає математичним часовим складностям методів Гаусу та методу обернених матриць, тобто значенню $O(n^3)$. Аналіз часової складності коду C^{++} програмної реалізації арифметизації на основі методу швидкого зворотного перетворення Фур'є відповідає часової складності цієї програмної реалізації, яка була отримана при використанні результатів тестування програмної реалізації та при використанні методу лінійної регресії і відповідає значенню $O(n * log_2 n)$. Це значення часової складності також відповідає математичній часовій складності методу швидкого зворотного перетворення Фур'є, тобто значенню $O(n * log_2 n)$.

5) Зроблено порівняльний аналіз реалізації арифметизації протоколу ZK-STARK на основі швидкого перетворення Фур'є , на основі методу обернених матриць та на основі методу Гаусу.

За допомогою швидкого зворотного перетворення Фур'є можливо вирішувати системи лінійних алгебраїчних рівнянь у полі Златовласки $GF(2^{64}-2^{32}+1)$, у яких максимальна кількість невідомих становить 2^{24} та рівнянь становить 2^{24} (максимально можлива кількість за «розумний час»), що у 16578 разів більше ніж у методів Гауса та зворотного перетворення Фур'є, за допомогою яких є можливим вирішувати системи лінійних алгебраїчних рівнянь у полі GF(257), які

мають 1012 невідомих та 1012 рівнянь (максимально можлива кількість за «розумний час»).

Було встановлено, що за допомогою швидкого перетворення Фур'є можливо вирішувати систему лінійних алгебраїчних рівнянь у яких максимальна кількість невідомих становить 2^{24} та рівнянь становить 2^{24} у полі Златовласки GF(2^{64}- 2^{32}+1) за 177,45 секунди. Водночас, системи лінійних алгебраїчних рівнянь у полі GF(257), які вирішуються за допомогою метод Гаусу та методу обернених матриць, можливо вирішувати, як мінімум, за 30,11 секунди.

Під час тестування було виявлено, що більша основа P поля, то більші числа воно містить і то довше проходить пошук коефіцієнтів інтерполяційного полінома. Із тестів було визначено, що, якщо варіанту арифметизації, який використовує метод Гаусу або зворотних матриць для пошуку коефіцієнтів інтерполяційного поліному подати на вхід 8 числа Фібоначчі, але розрахунки проводити у полі Златовласки, тобто за модулем P = 2^{64}- 2^{32}+1, то цей варіант працював 40 хвилин без результату, тобто це в 1 545 946 разів довше, ніж варіант з швидким зворотним перетворенням Фур'є.

Що більший час роботи програми (час прогону), то менше необхідно зробити прогонів для визначення середнього часу роботи програми з довірчою ймовірністю $\Upsilon \geq 0,9$ та точністю $\varepsilon \leq 0,1$.

Що більша довжина циклічної підгрупи (що більше вхідних чисел Фібоначчі) для заданого модуля і первісного кореня, то меншу кількість прогонів програми треба зробити для визначення середнього часу роботи програми з довірчою $\Upsilon \geq 0,9$ та точністю $\varepsilon \leq 0,1$.

6) Проведений загальний аналіз результатів тестування першого етапу арифметизації протоколу ZK-STARK та надні рекомендації щодо найбільш ефективної роботи реалізації протоколу та щодо його впровадження до мережі Блокчейн. Під час тестування було виявлено, що час роботи арифметизації залежить від методу, за допомогою якого знаходяться коефіцієнти інтерполяційного полінома, від кількості вхідних чисел Фібоначчі та від поля, яке використовується для пошуку коефіцієнтів інтерполяційного полінома.

Приймаючи до уваги, що поле Златовласки GF(2^{64}- 2^{32}+1) у $\frac{2^{64}-2^{32}+1}{257}$ разів більше, ніж поле GF(257) та, що у промисловості використовуються поля, розміром з полем Златовласки і більше, можна стверджувати, що метод швидкого перетворення Фур'є набагато більш практичний і має набагато більший потенціал для розв'язку систем алгебраїчних рівнянь (наприклад, під час пошуку коефіцієнтів інтерполяційного поліному у протоколі ZK-STARK), ніж метод Гаусу і метод обернених матриць.

Тестування показало, що практичне впровадження арифметизації на основі швидкого зворотного перетворення Фур'є має часову складність $O(n * log_2 n)$, яка у $\frac{n^3}{n*log_2 n}$ разів менша, ніж часова складність арифметизації на основі зворотних матриць та методу Гауса, що прискорює роботу арифметизації.

За допомогою швидкого зворотного перетворення Фур'є вдалося зменшити час перетворення вхідних даних у поліном та ще зменшити відсоток часу, який займає процес перетворення вхідних даних на поліном (процедура Інтерполяції) від часу роботи всього першого етапу, тобто завдяки ШЗПФ відсоток часу зменшився із 99,99 % (для 1012 вхідних чисел Фібоначчі та методи Гаусу та Обернених матриць для інтерполяції) до 58,56% (для 2^{24} чисел Фібоначчі та Швидкого зворотного перетворення Фур'є).

Найперспективнішим методом серед трьох протестованих виявився метод швидкого зворотного перетворення Фур'є, тому рекомендується на етапі арифметизації протоколу ZK-STARK використовувати саме цей метод, бо він збільшить продуктивність роботи самого протоколу, тобто прискорить процедуру автентифікації, частиною якої і є етап арифметизація.

Вирішення систем лінійних алгебраїчних рівнянь з великою кількістю невідомих та рівнянь – це дуже складна обчислювальна задача. Тому, якщо діяльність протоколу ZK-STARK зумовлює рішення систем лінійних алгебраїчних рівнянь, розміру 2^{24}, чи більшого, то, наразі, має сенс розподілити обчислення під час процедури автентифікації у протоколі ZK-STARK між комп'ютерами обчислювальної мережі Блокчейн задля прискорення роботи

першого етапу роботи протоколу або збільшити потужність окремих вузлів мережі, або проводити обчислення на квантовому комп’ютері, бо кількість даних, які потрібно оброблювати комп’ютерним мережам кожен рік зростає.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. П. Кравченко, Б. Скрябін, О. Дубініна. Блокчейн і децентралізовані системи : навч. посібник для студ. закладів вищ. освіти : в 3 частинах. Ч. 1 / П. Кравченко, Б. Скрябін, О. Дубініна. – Харків : ПРОМАРТ, 2019. – 452 с.
2. STARK Math: The Journey Begins [Електронний ресурс] / StarkWare // Medium. – Режим доступу: https://medium.com/starkware/stark-math-the-journey-begins-51bd2b063c71 (дата звернення: 10.11.2024).
3. Extreme Integrity in Decentralized World [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/starkware/extreme-integrity-in-decentralized-world-9e66cdf24d8b (дата звернення: 10.11.2024).
4. Mastering Blockchain [Електронний ресурс] / Lorne Lantz, Daniel Cawrey // Errata. – Режим доступу: http://oreilly.com/catalog/errata.csp?isbn=9781492054702 (дата звернення: 10.11.2024).
5. Introduction to zk-SNARKs [Електронний ресурс] // CONSENSYS Blog. – Режим доступу: https://consensys.net/blog/developers/introduction-to-zk-snarks/ (дата звернення: 10.11.2024).
6. What are zk-STARKs? [Електронний ресурс] // bit2me ACADEMY. – Режим доступу: https://academy.bit2me.com/en/what-are-zk-stark/ (дата звернення: 10.11.2024).
7. Arithmetization II [Електронний ресурс] / StarkWare // Medium. – Режим доступу: https://medium.com/starkware/arithmetization-ii-403c3b3f4355 (дата звернення: 10.11.2024).
8. Zero-Knowledge Proofs: STARKs vs SNARKs [Електронний ресурс] / CONSENSYS // Blockchain Explained. – Режим доступу: https://consensys.net/blog/blockchain-explained/zero-knowledge-proofs-starks-vs-snarks/ (дата звернення: 10.11.2024).
9. Polygon Launches a zk-STARK Scaling Solution for Dapp Deployment

[Електронний ресурс] // Cointelegraph. – Режим доступу: https://cointelegraph.com/news/polygon-launches-a-zk-stark-scaling-solution-for-dapp-deployment (дата звернення: 10.11.2024).

10. What is StarkNet? [Електронний ресурс] // StarkNet. – Режим доступу: https://www.starknet.io/what-is-starknet/ (дата звернення: 10.11.2024).
11. Top Ethereum zk-Rollup Projects [Електронний ресурс]. – Режим доступу: https://www.kucoin.com/ru/learn/crypto/top-ethereum-zk-rollup-projects (дата звернення: 10.11.2024).
12. Why zk-Technology is the Key to Unlocking Cryptocurrency's Full Potential [Електронний ресурс]. – Режим доступу: https://www.noncustodial.finance/news/why-zk-technology-is-the-key-to-unlocking-cryptocurrencys-full-potential (дата звернення: 10.11.2024).
13. Understanding zk-SNARK vs zk-STARK [Електронний ресурс]. – Режим доступу: https://metaschool.so/articles/understanding-zk-snark-vs-zkstark/ (дата звернення: 10.11.2024).
14. What is Immutable X (IMX) [Електронний ресурс]. – Режим доступу: https://academy.binance.com/uk/articles/what-is-immutable-x-imx (дата звернення: 10.11.2024).
15. STARK Math – A Very Short Primer [Електронний ресурс] / StarkWare. – Режим доступу: https://starkware.co/stark-math-a-very-short-primer/ (дата звернення: 10.11.2024).
16. GitHub Repository for libSTARK [Електронний ресурс]. – Режим доступу: https://github.com/elibensasson/libSTARK (дата звернення: 10.11.2024).
17. GitHub Repository for zkSTARKs [Електронний ресурс]. – Режим доступу: https://github.com/actuallyachraf/zkstarks (дата звернення: 10.11.2024).
18. Proposal: AIR Assembly [Електронний ресурс] // zkProof Documentation. – Режим доступу: https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-airAssembly.pdf (дата звернення: 10.11.2024).
19. StarkWare Team. ethSTARK Documentation Version 1.1 [Електронний ресурс] // IACR. – Режим доступу: https://eprint.iacr.org/2021/582.pdf

20. Polygon Launches a zk-STARK Scaling Solution for Dapp Deployment [Електронний ресурс] // Cointelegraph. – Режим доступу: https://www.cryptotimes.io/2021/11/18/polygon-launches-a-zk-stark-scaling-solution-to-develop-dapp /
21. Blockchain Technology and Applications [Електронний ресурс] / Pethuru Raj, Kavita Saini, Chellammal Surianarayanan. – Режим доступу: https://www.amazon.com/Blockchain-Technology-Applications-Pethuru-Raj-ebook-dp-B08DWBSPGW/dp/B08DWBSPGW/ref=mt_other?_encoding=UTF8&me=&qid (дата звернення: 10.11.2024).
22. Beginning Blockchain. A Beginner's Guide to Building Blockchain Solutions [Електронний ресурс] / Bikramaditya Singhal, Gautam Dhameja, Priyansu Sekhar Panda. – Режим доступу: https://doi.org/10.1007/978-1-4842-3444-0 (дата звернення: 10.11.2024).
23. Key Features of Blockchain [Електронний ресурс]. – Режим доступу: https://thefintechway.com/6-key-features-of-blockchain/(дата звернення: 10.11.2024).
24. Features of Blockchain [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/features-of-blockchain/ (дата звернення: 10.11.2024).
25. П. Кравченко, Б. Скрябін, О. Дубініна. Блокчейн і децентралізовані системи: навч. посібник для студ. закладів вищ. освіти: в 3 частинах. Ч. 2 / П. Кравченко, Б. Скрябін, О. Курбатов, О. Дубініна. - Харків, 2019. – 412 с.
26. П. Кравченко, Б. Скрябін, О. Дубініна. Блокчейн і децентралізовані системи: навч. посібник для студ. закладів вищ. освіти: в 3 частинах. Ч. 3 / П. Кравченко, Б. Скрябін, О. Курбатов, О. Дубініна. - Харків, 2022. – 380 с.
27. Горбенко І.Д., Горбенко Ю.І. Прикладна криптологія. Теорія. Практика. Застосування: Підручник для вищих навчальних закладів. – Харків: «Форт», 2013. – 880с.

28. О. О. Кузнецов М. О. Полуяненко. Огляд технології блокчейн
Лекція 3. Адреси та управління ключами: методичні рекомендації з дисципліни «Технології блокчейн»[для студентів вищих навчальних закладів, які навчаються за напрямами підготовки «Безпека інформаційних і комунікаційних систем»]. Х. : ХНУ імені В.Н. Каразіна, 2022. – 25 с.
29. Кузнецов О., Полуяненко М. Основні поняття та визначення в моделюванні консенсусу децентралізованих систем. Протоколи консенсусу відповідно до моделі «Доказ виконаної роботи»: методичні рекомендації з дисципліни «Технології блокчейн»[для студентів вищих навчальних закладів, які навчаються за напрямами підготовки «Безпека інформаційних і комунікаційних систем»]. Х. : ХНУ імені В.Н. Каразіна, 2019. – 45 с.
30. Consensus Algorithms: The Root Of Blockchain Technology [Електронний ресурс] / Guides. // 101Blockchains. – Режим доступу: https://101blockchains.com/consensus-algorithms-blockchain/ (дата звернення: 10.11.2024).
31. Comparison Among The Consensus Algorithms [Електронний ресурс] / Rutu Manhandi. // LinkedIn. – Режим доступу: https://www.linkedin.com/pulse/comparison-among-consensus-algorithms-rutu-mandhani (дата звернення: 10.11.2024).
32. ZK-SNARKs and zk-STARKs Explained [Електронний ресурс] // Binance Academy. – Режим доступу: https://academy.binance.com/en/articles/zk-snarks-and-zk-starks-explained (дата звернення: 10.11.2024).
33. SNARKs vs. STARKs [Електронний ресурс] / ZK Whiteboard Sessions. // ZK Hack. – Режим доступу: https://zkhack.dev/whiteboard/module-four/ (дата звернення: 10.11.2024).
34. ZK-SNARKs: A Gentle Introduction [Електронний ресурс]. // ENS. – Режим доступу: https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf (дата звернення: 10.11.2024).
35. Theoretical and Practical Introduction to ZK-SNARKs and ZK-STARKs [Електронний ресурс] // MUNI. – Режим доступу:

https://is.muni.cz/th/ovl3c/SNARKs_STARKs_introduction_Archive.pdf (дата звернення: 10.11.2024).

36. A Brief History of the Development of Zero-Knowledge Proof Technology [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/@zkpedia33/a-brief-history-of-the-development-of-zero-knowledge-proof-technology-explained-in-one-article-3b7f64f74047 (дата звернення: 10.11.2024).
37. Proofs, Arguments, and Zero-Knowledge [Електронний ресурс] / Justin Thaler. – Режим доступу: https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf (дата звернення: 10.11.2024).
38. Zero-Knowledge in EasyCrypt [Електронний ресурс] / Denis Firsov, Dominique Unruh // Guardtime, Tallinn University of Technology, Tartu University. – Режим доступу: https://m.guardtime.com/files/Zero-Knowledge%20in%20EasyCrypt.pdf (дата звернення: 10.11.2024).
39. ZK-STARKs vs ZK-SNARKs: Differences in Zero-Knowledge Technologies [Електронний ресурс] // Panther Academy. – Режим доступу: https://blog.pantherprotocol.io/zk-snarks-vs-zk-starks-differences-in-zero-knowledge-technologies/ (дата звернення: 10.11.2024).
40. The Zero Knowledge Frontier: On SNARKs, STARKs, and Future Applications [Електронний ресурс] // TheTie. – Режим доступу: https://research.thetie.io/zero-knowledge-starks-snarks/ (дата звернення: 10.11.2024).
41. ZK-STARKs vs. ZK-SNARKs Explained [Електронний ресурс]. // Cointelegraph: The Future of the Money. – Режим доступу: https://cointelegraph.com/explained/zk-starks-vs-zk-snarks-explained (дата звернення: 10.11.2024).
42. Understanding the Difference Between zk-SNARKs and zk-STARKs [Електронний ресурс] // Chainlink Blog. – Режим доступу: https://blog.chain.link/zk-snarks-vs-zk-starks/ (дата звернення: 10.11.2024).

43. ZK-STARKs vs ZK-SNARKs: Exploring Key Differences [Електронний ресурс] // Hacken. – Режим доступу: https://hacken.io/discover/zk-snark-vs-zk-stark/ (дата звернення: 10.11.2024).
44. Differences Between zk-SNARKs and zk-STARKs [Електронний ресурс] / Chainlink Education Hub. – Режим доступу: https://chain.link/education-hub/zk-snarks-vs-zk-starks (дата звернення: 10.11.2024).
45. SNARKs vs. STARKs: A Comparative Analysis [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/@ramsesfv/starks-vs-snarks-d2e09c4e6069 (дата звернення: 10.11.2024).
46. Decoding zk-STARK vs zk-SNARK: An In-Depth Comparative Analysis [Електронний ресурс] // Antier Solutions. – Режим доступу: https://www.antiersolutions.com/decoding-zk-stark-vs-zk-snark-an-in-depth-comparative-analysis/ (дата звернення: 10.11.2024).
47. zk-SNARKs vs zk-STARKs: Comparing the Main ZKP Technologies [Електронний ресурс] // Halborn. – Режим доступу: https://www.halborn.com/blog/post/zk-snarks-vs-zk-starks-comparing-the-main-zkps-in-blockchain (дата звернення: 10.11.2024).
48. Differences Between zk-SNARKs and zk-STARKs [Електронний ресурс] // Extrimian Blog. – Режим доступу: https://extrimian.io/differences-between-zk-snarks-and-zk-starks/ (дата звернення: 10.11.2024).
49. ZK-STARKs vs ZK-SNARKs [Електронний ресурс] // QuillAudits. – Режим доступу: https://www.quillaudits.com/blog/ethereum/zk-starks-vs-zk-snarks (дата звернення: 10.11.2024).
50. A Full Comparison: What Are zk-SNARKs and zk-STARKs [Електронний ресурс]. – Режим доступу: https://www.cyfrin.io/blog/a-full-comparison-what-are-zk-snarks-and-zk-starks (дата звернення: 10.11.2024).
51. What is StarkNet? [Електронний ресурс] // StarkNet. – Режим доступу: https://www.starknet.io/what-is-starknet/ (дата звернення: 10.11.2024).
52. Introducing Plonky2 [Електронний ресурс] / Polygon Labs. – Режим доступу: https://polygon.technology/blog/introducing-plonky2 (дата звернення:

10.11.2024).

53. Telos Foundation Unveils Plonky2 Goldibear for Enhanced Cryptographic Proofs [Електронний ресурс]. – Режим доступу: https://www.binance.com/ru-UA/square/post/2024-10-10-telos-foundation-unveils-plonky2-goldibear-for-enhanced-cryptographic-proofs-14685499416073 (дата звернення: 10.11.2024).
54. Revealing Plonky2x: The Fastest Proof Aggregator Over the BabyBear Field [Електронний ресурс]. – Режим доступу: https://www.telos.net/post/revealing-plonky2x-the-fastest-proof-aggregator-over-the-babybear-field (дата звернення: 10.11.2024).
55. Arithmetization I [Електронний ресурс] / StarkWare // Medium. – Режим доступу: https://medium.com/starkware/arithmetization-i-15c046390862 (дата звернення: 10.11.2024).
56. Scalable, Transparent, and Post-Quantum Secure Computational Integrity [Електронний ресурс]. // IACR. – Режим доступу: https://eprint.iacr.org/2018/046.pdf (дата звернення: 10.11.2024).
57. Кузнецов О.О. ЛЕКЦІЯ № 16 – 17 з дисципліни “ТЕОРІЯ ГРУП, ПОЛІВ, КІЛЕЦЬ” Розділ 1. Арифметика та основні властивості груп, полів, кілець Тема лекції. Основні поняття та визначення теорії груп, полів, кілець. Х. : ХНУ імені В. Н. Каразіна, 2020. – 20 с.
58. О. О. Кузнецов, "Конспект лекцій по математичним основам криптографічних доказів з нульовим розголошенням. Лекція №23: Дискретне перетворення Фур'є в кінцевих полях," PROXIMA LABS, 1501 Larkin Street, Suite 300, San Francisco, USA, 13 с.
59. H. P. William, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, Numerical Recipes in C++. The Art of Scientific Computing, 2nd ed: Cambridge University Press, Cambridge, 2003, 1004 с.
60. Державний університет "Житомирська політехніка". Освітній портал [Електронний ресурс]. – Режим доступу: https://learn.ztu.edu.ua (дата звернення: 10.11.2024).
61. Літнарович Р.М. Алгебра матриць: Курс лекцій [Електронний ресурс]. –

Рівне: МЕГУ, 2007. – 112 с. – Режим доступу: https://core.ac.uk/download/pdf/14034615.pdf (дата звернення: 10.11.2024).

62. Кузнецов О.О. Конспект лекцій по математичним основам криптографічних доказів з нульовим розголошенням. Лекція №22: Дискретне перетворення Фур'є: PROXIMA LABS, 1501 Larkin Street, Suite 300, San Francisco, USA, 13 с.
63. The Fast Fourier Transform (FFT): Most Ingenious Algorithm Ever? [Електронний ресурс]. // YouTube. – Режим доступу: https://www.youtube.com/watch?v=h7apO7q16V0 (дата звернення: 10.11.2024).
64. Кузнецов О.О. Конспект лекцій по математичним основам криптографічних доказів з нульовим розголошенням. Лекція №23: Дискретне перетворення Фур'є в кінцевих полях: PROXIMA LABS, 1501 Larkin Street, Suite 300, San Francisco, USA, 13 с.
65. Кузнецов О.О. ЛЕКЦІЯ 10-11 «Тема 7. Планування експериментів при дослідженні ефективності технічних систем» з дисципліни “Моделювання та оцінка ефективності засобів захисту інформації”. Х. : ХНУ імені В. Н. Каразіна, 2017. – 15 с.

ДОДАТОК А

Лістинг програми КПМАП-ZKSTARK-01

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cmath>
#include <vector>
#include <iostream>
#include <vector>
#include <chrono>
#include <iomanip>

#include "NTL\ZZ.h"

using namespace std;
using namespace NTL;
using namespace std;
using namespace std;
using namespace chrono;

template <typename T> void FreeMem(T** matr, int n);
void gaussian_elimination(int n);
vector<ZZ> back_substitution(int n);
////Тест 3
//const int Marray = 502;
//ZZ g = conv < ZZ>("5");
//ZZ Mod = conv<ZZ>("503");

//const int Marray = 1012;
//ZZ g = conv < ZZ>("3");
//ZZ Mod = conv<ZZ>("1013");



//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("4");
//
//const int Marray = 8;

//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("2");
//
//const int Marray = 16;

//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("15");
```

```
//
//const int Marray = 32;
//с точностью 1 наносекда

//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("11");
//
//const int Marray = 64;

//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("9");
//
//const int Marray = 128;
//
//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("3");
//
//const int Marray = 256;
//const int Marray = 502;
//ZZ g = conv < ZZ>("5");
//ZZ Mod = conv<ZZ>("503");

//const int Marray = 1012;
//ZZ g = conv < ZZ>("3");
//ZZ Mod = conv<ZZ>("1013");

const int Marray = 6;
ZZ g = conv < ZZ>("4");
ZZ Mod = conv<ZZ>("13");

ZZ* argX = new ZZ[Marray];

ZZ number00 = conv < ZZ>("0");
ZZ number01 = conv < ZZ>("1");
ZZ number02 = conv < ZZ>("2");
vector<vector<ZZ>> a;
vector<ZZ> b;
//static_assert to check whether high_resolution_clock is actually aliased to a real clock
//https://stackoverflow.com/questions/38252022/does-standard-c11-guarantee-that-high-resolution-clock-measure-real-time-non
static_assert(
   std::is_same<high_resolution_clock, steady_clock>::value
   || std::is_same<high_resolution_clock, system_clock>::value,
   "high_resolution_clock IS NOT aliased to one of the other standard clocks!");
using maxres_sys_or_steady =
std::conditional<
   system_clock::period::den <= steady_clock::period::den,
   system_clock, steady_clock
>::type;
using maxres_nonsleeping_clock =
std::conditional<
```

```
    high_resolution_clock::is_steady,
    high_resolution_clock, maxres_sys_or_steady
>::type;


void main()
{
    cout << "================ Interpolation by Gauss method ================" << endl;
    locale loc("ru_RU.UTF-8");
    auto start1 = high_resolution_clock::now();//Начальная (первая) точка отчета времени
    ofstream inOut1, inOut2, inOut3, inOut4;
    inOut1.open("Time_mesurement_STARK_RevMatr Interpolation time.csv", ios::app);
    inOut2.open("Time_mesurement_STARK_RevMatr Verification time.csv", ios::app);
    inOut3.open("Time_mesurement_STARK_RevMatr TOTAL time.csv", ios::app);
    inOut4.open("Mesurements checking.csv", ios::app);
    inOut1.imbue(loc);
    inOut2.imbue(loc);
    inOut3.imbue(loc);
    inOut4.imbue(loc);
    if (!inOut1.is_open() || !inOut2.is_open() || !inOut3.is_open() || !inOut4.is_open()) {
        std::cerr << " File was not opened successfully!" << std::endl;
        return;
    }
    inOut1 << " " << endl; // Используем табуляцию
    inOut2 << " " << endl; // Используем табуляцию
    inOut3 << " " << endl; // Используем табуляцию
    inOut4 << " " << endl;

    /*double time_spent = 0.0;
    clock_t begin = clock();

    setlocale(0, "");*/
    int n = 1;

    const int size01 = 1;
    cout << "Enter the size of the matrix: ";
    n = Marray;
    cout << n << endl;
    ZZ* Fibonacci = new ZZ[n];
    //============================================================================
        // Формирование чисел Фибоначчи
    Fibonacci[0] = conv < ZZ>("1");
    Fibonacci[1] = conv < ZZ>("1");
    for (int j = 2; j < Marray; j++)
    {
        Fibonacci[j] = Fibonacci[j - 1] + Fibonacci[j - 2];
        // cout << Fibonacci[j] << endl;
    }
    //============================================================================

    cout << "Fibonacci numbers - done" << endl;
    //============================================================================
        // Поиск значий циклической группы по заданному первообразному корню g
```

```
    for (int j = 0; j < Marray; j++)
    {
        argX[j] = PowerMod(g, j, Mod);//PowerMod(g, j, Mod)

    }
    ZZ tmp01[3]{};
    tmp01[0] = 1;//при x^2
    tmp01[1] = (-(argX[Marray - 1] + argX[Marray - 2]));//при x^1
    tmp01[2] = ((argX[Marray - 1] * argX[Marray - 2]));//при x^0
    cout << "Args X - done" << endl;
    //=============================================================================

    /*for (int i = 0; i < 3; i++) {
        cout << tmp01[i]<< endl;

    }*/
    cout << endl;

    a.assign(Marray, vector<ZZ>(Marray));
    b.assign(Marray, number00);


    for (int i = 0; i < a.size(); i++) {
        for (int j = 0; j < a.size(); j++)
            a[i][j] = PowerMod(argX[i], j, Mod);
    }

    for (int i = 0; i < Marray; i++)
    {
        b[i] = Fibonacci[i];
    }

    cout << "The matrix is:" << endl;
    //for (int i = 0; i < a.size(); i++) {
    //    for (int j = 0; j < a.size(); j++)
    //     // cout << a[i][j];
    //    cout << endl;
    //}
    //cout << endl;

    delete[] argX;

    cout << "The matrix was created " << endl;

//*****************************************************************************************************************************
*****
    //*************************************************Решение СЛАУ методом
Гаусса************************************************

    //auto start = chrono::high_resolution_clock::now();

    gaussian_elimination(Marray);
    auto x = back_substitution(Marray);
```

```
    auto stop1 = high_resolution_clock::now();//вторая точка отчета времени
    auto start2 = high_resolution_clock::now();//третья точка отчета времени
    auto duration1 = duration_cast<duration<double>>(stop1 - start1); // Преобразуем в секунды с двойной точностью

    //auto end2 = chrono::high_resolution_clock::now();
    //double time_taken =
    //    chrono::duration_cast<chrono::milliseconds>(end2 - start).count();

    //cout << "Solution is ready: ";
    ///* for (int i = 0; i < Marray; ++i)
    //     cout << "x[" << i << "] = " << x[i] << "\n";*/

    //cout << "Time taken: " << time_taken / 1000 << " seconds\n";

//*****************************************************************************************************************************
*****

//*****************************************************************************************************************************
*****
    for (int i = 0; i < n / 2; i++) {
        ZZ temp = x[n - i - 1];
        x[n - i - 1] = x[i];
        x[i] = temp;
    }

    cout << "Polynom f(x): " << endl;
    /*for (int i = 0; i < Marray; i++)
    {

        cout << x[i] << " ";
    }*/

    ZZ* f_gx_ = new ZZ[Marray];
    ZZ* f_g2x_ = new ZZ[Marray];
    ZZ* subtractionArr = new ZZ[Marray];

    cout << endl;
    cout << "Polynom f(g*x):" << endl;
    int counter02 = Marray - 1;
    for (int i1 = 0;i1 < Marray;i1++) {
        f_gx_[i1] = x[i1] * PowerMod(g, counter02, Mod);
        f_gx_[i1] = f_gx_[i1] % Mod;
        /*cout << "f_gx_ [" << i1 << "] = " << f_gx_[i1] << endl;*/
       // cout << f_gx_[i1] << " ";
        counter02--;
    }
  //  cout << endl;


    cout << "Polynom f(g^2*x):" << endl;
    ZZ number03 = PowerMod(g, number02, Mod);
    int counter03 = Marray - 1;
```

```
for (int i = 0;i < Marray;i++) {
    f_g2x_[i] = x[i] * PowerMod(number03, counter03, Mod);
    f_g2x_[i] = f_g2x_[i] % Mod;
    /*cout << "f_g2x_ [" << i << "] = " << f_g2x_[i] << endl;*/
    //cout << f_g2x_[i] << " ";
    counter03--;
}
// cout << endl;
cout << "Polynom f(g^2*x) - f(g*x) - f(x):" << endl;
for (int i = 0; i < Marray; i++) {
    subtractionArr[i] = f_g2x_[i] - f_gx_[i] - x[i];
    subtractionArr[i] = subtractionArr[i] % Mod;
   /*cout << "subtractionArr [" << i << "] = " << subtractionArr[i] << endl;*/
    //cout << subtractionArr[i] << " ";
}
cout << endl;
delete[]f_gx_;
delete[]f_g2x_;
//delete[]x;

cout << "Polynom (x-a)*(x-b)*(f(g^2*x) - f(g*x) - f(x))" << endl;
const int NewSize = Marray + 2;
ZZ* mul = new ZZ[NewSize];


for (int i = Marray - 1; i >= 0; i--) {
    for (int j = 2; j >= 0; j--) {
        mul[i + j] += subtractionArr[i] * tmp01[j];//tmp02+
        mul[i + j] = mul[i + j] % Mod;

    }
}
delete[]subtractionArr;
int i10 = 0;
//for (i10 = 0; i10 < NewSize; i10++) {
//   /* cout << "mul [" << i10 << "]  =" << mul[i10] << endl;*/
//    cout << mul[i10] << " ";
//}
cout << endl;
cout << "Polynom (x-a)*(x-b)*(f(g^2*x) - f(g*x) - f(x))/(x^n - 1)" << endl;
//=============================================================================================
//=============================================================================================
//=============================================================================================
//
    //Polynom temp = p1;
const int degreeOfNumerator = NewSize;//degree = 7
const int degreeOfDivider = Marray;//degree = 6
int rdeg = degreeOfNumerator - degreeOfDivider;

ZZ* res = new ZZ[NewSize - Marray + 1]{};
ZZ* PolynomialDivider = new ZZ[degreeOfNumerator]{};

for (int i10 = 0; i10 < degreeOfNumerator; i10++) {
```

```
    PolynomialDivider[i10] = 0;
}

PolynomialDivider[1] = 1;
PolynomialDivider[degreeOfNumerator - 1] = 12;

//for (i10 = 0; i10 < degreeOfNumerator; i10++) {
//   /* cout << "PolynomialDivider [" << i10 << "]  =" << PolynomialDivider[i10] << endl;*/
//    cout << PolynomialDivider[i10] << " ";
//}

//**************************************************Рабочее**************************************************
//cout << "degreeOfNumerator = " << degreeOfNumerator << endl;
//cout << "rdeg = " << rdeg << endl;
int counter04 = 1;

for (int i11 = 0; i11 < rdeg; i11++) {
    res[rdeg - i11 - 1] = mul[0 + i11] / PolynomialDivider[1];
    //cout << "res [" << i11 << "]  =" << (res[rdeg - i11 - 1]) << endl;
    //cout << (res[rdeg - i11 - 1]) << " ";

    for (int j4 = 0; j4 <= degreeOfNumerator; j4++) {
        mul[0 + j4 + i11] -= PolynomialDivider[counter04] * res[rdeg - i11 - 1];
       // cout << "mul [" << 0 + j4 + i11 << "]  =" << mul[0 + j4 + i11] % Mod << endl;
        counter04++;
        if (counter04 == NewSize) {
            break;
        }

    }

    counter04 = 1;

}
    delete[]res;
    delete[]mul;
    delete[]PolynomialDivider;
 //*************************************************************************************************************
//*************<Time>*****************************************************************
//*************************************************************************************************************
    //clock_t end = clock();
    //// рассчитать прошедшее время, найдя разницу (end - begin) и
    //    // деление разницы на CLOCKS_PER_SEC для перевода в секунды
    //time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
    //cout << endl;
    //cout << "The elapsed time is %f" << time_spent << " seconds" << endl;
    auto stop2 = high_resolution_clock::now();//четвертая точка отчета времени
    auto duration2 = duration_cast<duration<double>>(stop2 - start2);
    auto duration3 = duration_cast<duration<double>>(stop2 - start1);
    auto duration4 = duration3 - duration2 - duration1;
    inOut1 << fixed << setprecision(9) << duration1.count() << "" << "";;
    inOut2 << fixed << setprecision(9) << duration2.count() << "" << "";;
    inOut3 << fixed << setprecision(9) << duration3.count() << "" << "";;
```

```
    inOut4 << fixed << setprecision(20) << duration4.count() << "" << "";;

    inOut1.close();
    inOut2.close();
    inOut3.close();
    inOut4.close();
} // end of main


//*****************************************************************************************************
//                    Дополнительные функции
//*****************************************************************************************************
//Функция освобождения памяти
template <typename T> void FreeMem(T** matr, int n)
{
  for (int i = 0; i < n; i++)
    delete[] matr[i];
  delete[] matr;
}

// Gaussian elimination to transform matrix to triangular form
void gaussian_elimination(int n) {
  for (int row = 0; row < n; ++row) {
    // Find row with maximum value in the column
    int max_row = row;
    for (int i = row + 1; i < n; ++i) {
      if (abs(a[i][row]) > abs(a[max_row][row]))
        max_row = i;
    }
    // Swap rows
    swap(a[row], a[max_row]);
    swap(b[row], b[max_row]);
    // Eliminate column
    //if (abs(a[row][row]) < EPS)
    //    continue;
    ZZ invElofGroup = conv<ZZ>("1");
    ZZ temp = a[row][row];
    for (int ii = 1; ii < Mod; ii++)
    {
      ZZ antiDivider = temp * ii;
      ZZ antiDivider1 = antiDivider % Mod;
      if (antiDivider1 == 1)
      {
        invElofGroup = ii;
      }
    }
    ZZ inv_diag = (number01 * invElofGroup) % Mod;
    for (int i = row + 1; i < n; ++i) {
      ZZ k = (a[i][row] * inv_diag) % Mod;
      for (int j = row + 1; j < n; ++j)
        a[i][j] -= (k * a[row][j]) % Mod;
      b[i] -= (k * b[row]) % Mod;
      a[i][row] = number00;
```

```
        }
    }
}

// Back substitution to solve triangular system
vector<ZZ> back_substitution(int n) {
    vector<ZZ> x(n);
    for (int i = n - 1; i >= 0; --i) {
        ZZ sum = number00;
        for (int j = i + 1; j < n; ++j)
            sum += (a[i][j] * x[j]) % Mod;

        ZZ invElofGroup = conv<ZZ>("1");
        ZZ temp = a[i][i];
        for (int ii = 1; ii < Mod; ii++)
        {
            ZZ antiDivider = temp * ii;
            ZZ antiDivider1 = antiDivider % Mod;
            if (antiDivider1 == 1)
            {
                invElofGroup = ii;
            }
        }

        x[i] = ((b[i] - sum) * invElofGroup) % Mod;

    }
    return x;
}
```

ДОДАТОК Б

Лістинг програми КПМАП-ZKSTARK-02

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cmath>
#include "NTL\ZZ.h"
#include <iomanip>

using namespace std;
using namespace NTL;
using namespace std;
using namespace std;
using namespace chrono;


template <typename T> void FreeMem(T** matr, int n);
template <typename T> void SetMtx(T** matr, int n);
template <typename T> void TransponMtx(T** matr, T** tMatr, int n);
void inversion(ZZ** A, int N);

//Тест 1
const int Marray = 6;
ZZ g = conv < ZZ>("4");
ZZ Mod = conv<ZZ>("13");
//Тест 2
//const int Marray = 112;
//ZZ g = conv < ZZ>("3");
//ZZ Mod = conv<ZZ>("113");
////Тест 3
//const int Marray = 502;
//ZZ g = conv < ZZ>("5");
//ZZ Mod = conv<ZZ>("503");

//const int Marray = 1012;
//ZZ g = conv < ZZ>("3");
//ZZ Mod = conv<ZZ>("1013");


//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("4");
//
//const int Marray = 8;кл

//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("2");
//
//const int Marray = 16;
```

```
//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("15");
//
//const int Marray = 32;
//с точностью 1 наносекда

//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("11");
//
//const int Marray = 64;

//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("9");
//
//const int Marray = 128;

//ZZ Mod = conv<ZZ>("257");
//
//ZZ g = conv < ZZ>("3");
//
//const int Marray = 256;


//const int Marray = 222;
//ZZ g = conv < ZZ>("3");
//ZZ Mod = conv<ZZ>("223");

ZZ* argX = new ZZ[Marray];
ZZ* f_gx_ = new ZZ[Marray];
ZZ* f_g2x_ = new ZZ[Marray];
ZZ* subtractionArr = new ZZ[Marray];
ZZ* Coef = new ZZ[Marray];
//static_assert to check whether high_resolution_clock is actually aliased to a real clock.
//https://stackoverflow.com/questions/38252022/does-standard-c11-guarantee-that-high-resolution-clock-measure-real-time-non
static_assert(
   std::is_same<high_resolution_clock, steady_clock>::value
   || std::is_same<high_resolution_clock, system_clock>::value,
   "high_resolution_clock IS NOT aliased to one of the other standard clocks!");
using maxres_sys_or_steady =
std::conditional<
   system_clock::period::den <= steady_clock::period::den,
   system_clock, steady_clock
>::type;
using maxres_nonsleeping_clock =
std::conditional<
   high_resolution_clock::is_steady,
   high_resolution_clock, maxres_sys_or_steady
>::type;
```

```
void main()
{
   locale loc("ru_RU.UTF-8");
   auto start1 = high_resolution_clock::now();//Начальная (первая) точка отчета времени
   ofstream inOut, inOut1, inOut2, inOut3, inOut4;
   inOut.open("Matrix-01.txt");
   inOut1.open("Time_mesurement_STARK_RevMatr Interpolation time.csv", ios::app);
   inOut2.open("Time_mesurement_STARK_RevMatr Verification time.csv", ios::app);
   inOut3.open("Time_mesurement_STARK_RevMatr TOTAL time.csv", ios::app);
   inOut4.open("Mesurements checking.csv", ios::app);
   inOut1.imbue(loc);
   inOut2.imbue(loc);
   inOut3.imbue(loc);
   inOut4.imbue(loc);
   if (!inOut1.is_open()|| !inOut2.is_open()|| !inOut3.is_open()|| !inOut4.is_open()) {
      std::cerr << " File was not opened successfully!" << std::endl;
      return;
   }
   inOut1 << " " << endl; // Используем табуляцию
   inOut2 << " " << endl; // Используем табуляцию
   inOut3 << " " << endl; // Используем табуляцию
   inOut4 << " " << endl; // Используем табуляцию


   int n = 1;
   ZZ number00 = conv < ZZ>("0");// 4 bytes
   ZZ number01 = conv < ZZ>("1");// 4 bytes
   ZZ number02 = conv < ZZ>("2");// 4 bytes
   const int size01 = 1;// 4 bytes
   cout << "Enter the size of the matrix: ";
   n = Marray;
   cout << n << endl;
   ZZ** matr = new ZZ * [n];// 4 bytes*Marray*Marray

   ZZ Fibonacci[Marray][size01];// 4 bytes*Marray*Marray

   //=============================================================================
      // Формирование чисел Фибоначчи
   Fibonacci[0][0] = conv < ZZ>("1");
   Fibonacci[1][0] = conv < ZZ>("1");
   for (int j = 2; j < Marray; j++)
   {
      Fibonacci[j][0] = Fibonacci[j - 1][0] + Fibonacci[j - 2][0];
   }
   //=============================================================================

   cout << "Fibonacci numbers - done" << endl;
   //=============================================================================
      // Поиск значий циклической группы по заданному первообразному корню g
   for (int j = 0; j < Marray; j++)
   {
      argX[j] = PowerMod(g, j, Mod);//PowerMod(g, j, Mod)
```

```
}
ZZ tmp01[3]{};
tmp01[0] = 1;//при x^2
tmp01[1] = (-(argX[Marray - 1] + argX[Marray - 2]));//при x^1
tmp01[2] = ((argX[Marray - 1] * argX[Marray - 2]));//при x^0
cout << "Args X - done" << endl;
//=========================================================================


for (int i = 0; i < n; i++) {
    matr[i] = new ZZ[n];

}

cout << "The process of creating the matrix is underway. Please wait for..." << endl;
SetMtx(matr, n);

delete[] argX;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        inOut << matr[i][j] << " ";
    }
    inOut << '\n';
}
inOut.close();
cout << "The matrix was created " << endl;
// FreeMem(matr, n);
cout << "The inverse matrix is being calculated. Please wait for..." << endl;
// double time_spent1 = 0.0;
//clock_t begin1 = clock();


ZZ temp;
cout << "I create an inverse matrix..." << endl;
ZZ** E = new ZZ * [n];

for (int i = 0; i < n; i++) {
    E[i] = new ZZ[n];
}

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
    {
        E[i][j] = 0.0;

        if (i == j)
            E[i][j] = 1.0;
    }
cout << "An array of E..." << endl;
```

```
for (int k = 0; k < n; k++)
{
   temp = matr[k][k] % Mod;
   ZZ antiDet = conv<ZZ>("0");
   for (int ii = 1; ii < Mod; ii++)
   {
      ZZ antiDivider = temp * ii;
      ZZ antiDivider1 = antiDivider % Mod;
      if (antiDivider1 == 1)
      {
         antiDet = ii;
      }
   }
   for (int j = 0; j < n; j++)
   {
      matr[k][j] *= antiDet % Mod;
      E[k][j] *= antiDet % Mod;
   }

   for (int i = k + 1; i < n; i++)
   {
      temp = matr[i][k] % Mod;

      for (int j = 0; j < n; j++)
      {
         matr[i][j] -= (matr[k][j] * temp) % Mod;
         E[i][j] -= (E[k][j] * temp) % Mod;
      }
   }
}
cout << "The first cycle has passed..." << endl;
for (int k = n - 1; k > 0; k--)
{
   for (int i = k - 1; i >= 0; i--)
   {
      temp = matr[i][k] % Mod;

      for (int j = 0; j < n; j++)
      {
         matr[i][j] -= (matr[k][j] * temp) % Mod;
         E[i][j] -= (E[k][j] * temp) % Mod;
      }
   }
}
cout << "The second cycle has passed..." << endl;
for (int i = 0; i < n; i++)
   for (int j = 0; j < n; j++)
      matr[i][j] = E[i][j] % Mod;

for (int i = 0; i < n; i++)
   delete[] E[i];

delete[] E;
```

```
cout << "Array E is deleted ..." << endl;
cout << "Transpose matrices ..." << endl;

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        matr[j][i] = matr[i][j];
// ////=============================================================================================================
// ////                        Произведение матриц
// ////=============================================================================================================



cout << "Multiplying matrices..." << endl;
//Умножение матриц
ZZ** c = new ZZ * [Marray];
for (int i = 0; i < n; i++)
{
    c[i] = new ZZ[1];
    for (int j = 0; j < 1; j++)
    {
        c[i][j] = 0;
        for (int k = 0; k < Marray; k++)
            c[i][j] += (matr[i][k] * Fibonacci[k][j]) % Mod;
        c[i][j] = c[i][j] % Mod;
    }
}
//delete Fibonacci;
//delete tobr_matr;


FreeMem(matr, n);
auto stop1 = high_resolution_clock::now();//вторая точка отчета времени
auto start2 = high_resolution_clock::now();//третья точка отчета времени
auto duration1 = duration_cast<duration<double>>(stop1 - start1); // Преобразуем в секунды с двойной точностью


// clock_t end1 = clock();
 // рассчитать прошедшее время, найдя разницу (end1 - begin) и
    // деление разницы на CLOCKS_PER_SEC для перевода в секунды
// time_spent1 += (double)(end1 - begin) / CLOCKS_PER_SEC;
cout << endl;
cout << "Interpolation: The elapsed time mesured in seconds" << endl;
//auto start2 = high_resolution_clock::now();
//FreeMem(tobr_matr, n);
// Вывод матрицы произведения
cout << "Work matrix" << endl;
cout << "A one-dimensional array of coefficients: " << endl;
int counter01 = 0;
for (int i = Marray - 1; i >= 0; i--)
{
    Coef[counter01] = c[i][0];
    // cout << Coef[i] << " ";
```

```
    counter01++;
}



FreeMem(c, n);
cout << "Polynom f(x): " << endl;
for (int i = 0; i < Marray; i++)
{
    // Coef[i] = c[i][0];
    cout << Coef[i] << " ";
}


cout << endl;
cout << "Polynom f(g*x):" << endl;
int counter02 = Marray - 1;
for (int i1 = 0;i1 < Marray;i1++) {
    f_gx_[i1] = Coef[i1] * PowerMod(g, counter02, Mod);
    f_gx_[i1] = f_gx_[i1] % Mod;
    //cout << "f_gx_ [" << i1 << "] = " << f_gx_[i1] << endl;
   // cout << f_gx_[i1] << " ";
    counter02--;
}
//delete[] Coef;
cout << endl;

cout << "Polynom f(g^2*x):" << endl;
ZZ number03 = PowerMod(g, number02, Mod);
int counter03 = Marray - 1;
for (int i = 0;i < Marray;i++) {
    f_g2x_[i] = Coef[i] * PowerMod(number03, counter03, Mod);
    f_g2x_[i] = f_g2x_[i] % Mod;
    //cout << "f_g2x_ [" << i << "] = " << f_g2x_[i] << endl;
   // cout << f_g2x_[i] << " ";
    counter03--;
}
cout << endl;
cout << "Polynom f(g^2*x) - f(g*x) - f(x):" << endl;
for (int i = 0; i < Marray; i++) {
    subtractionArr[i] = f_g2x_[i] - f_gx_[i] - Coef[i];
    subtractionArr[i] = subtractionArr[i] % Mod;
    // cout << "subtractionArr [" << i << "] = " << subtractionArr[i] << endl;
  //  cout << subtractionArr[i] << " ";
}
cout << endl;
delete[]f_gx_;
delete[]f_g2x_;
delete[]Coef;

cout << "Polynom (x-a)*(x-b)*(f(g^2*x) - f(g*x) - f(x))" << endl;
const int NewSize = Marray + 2;
ZZ* mul = new ZZ[NewSize];
```

```
for (int i = Marray - 1; i >= 0; i--) {
    for (int j = 2; j >= 0; j--) {
        mul[i + j] += subtractionArr[i] * tmp01[j];//tmp02+
        mul[i + j] = mul[i + j] % Mod;

    }
}
delete[]subtractionArr;
int i10 = 0;
for (i10 = 0; i10 < NewSize; i10++) {
    // cout << "mul [" << i10 << "]  =" << mul[i10] << endl;
   // cout << mul[i10] << " ";
}
cout << endl;
cout << "Polynom (x-a)*(x-b)*(f(g^2*x) - f(g*x) - f(x))/x^n - 1" << endl;
//=============================================================================================
//=============================================================================================
//=============================================================================================
//
    //Polynom temp = p1;
const int degreeOfNumerator = NewSize;//degree = 7
const int degreeOfDivider = Marray;//degree = 6
//cout << "max2 = " << NewSize << endl;
//cout << "Marray = " << Marray << endl;
int rdeg = degreeOfNumerator - degreeOfDivider;
//int rdeg2 = rdeg;

ZZ* res = new ZZ[NewSize - Marray + 1]{};
ZZ* PolynomialDivider = new ZZ[degreeOfNumerator]{};

for (int i10 = 0; i10 < degreeOfNumerator; i10++) {
    PolynomialDivider[i10] = 0.0;
}

PolynomialDivider[1] = 1;
PolynomialDivider[degreeOfNumerator - 1] = 12;

for (i10 = 0; i10 < degreeOfNumerator; i10++) {
    //cout << "PolynomialDivider [" << i10 << "]  =" << PolynomialDivider[i10] << endl;
}



//**************************************************Рабочее**************************************************
//cout << "degreeOfNumerator = " << degreeOfNumerator << endl;
//cout << "rdeg = " << rdeg << endl;
int counter04 = 1;

for (int i11 = 0; i11 < rdeg; i11++) {
    res[rdeg - i11 - 1] = mul[0 + i11] / PolynomialDivider[1];
     //cout << "res [" << i11<< "]  =" << (res[rdeg - i11 - 1]) << endl;
```

```
      // cout << (res[rdeg - i11 - 1]) << " ";

       for (int j4 = 0; j4 <= degreeOfNumerator; j4++) {
          mul[0 + j4 + i11] -= PolynomialDivider[counter04] * res[rdeg - i11 - 1];

          counter04++;
          if (counter04 == NewSize) {
             break;
          }

       }

       counter04 = 1;

    }
    delete[]res;
    delete[]mul;
    delete[]PolynomialDivider;

    //*************************************************************************************************************
//*************<Time>*************************************************************
//*************************************************************************************************************

    auto stop2 = high_resolution_clock::now();//четвертая точка отчета времени
    auto duration2 = duration_cast<duration<double>>(stop2 - start2);
    auto duration3 = duration_cast<duration<double>>(stop2 - start1);
    auto duration4 = duration3 - duration2 - duration1;
    inOut1 << fixed << setprecision(9) << duration1.count() << "" << "";;
    inOut2 << fixed << setprecision(9) << duration2.count() << "" << "";;
    inOut3 << fixed << setprecision(9) << duration3.count() << "" << "";;
    inOut4 << fixed << setprecision(20) << duration4.count() << "" << "";;

    inOut1.close();
    inOut2.close();
    inOut3.close();
    inOut4.close();



    inOut1.close();
} // end of main

//*************************************************************************************************************
//                    Дополнительные функции
//*************************************************************************************************************
//Функция транспонирования матрицы
template <typename T> void TransponMtx(T** matr, T** tMatr, int n) {//
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            tMatr[j][i] = matr[i][j];
}
//Функция освобождения памяти
template <typename T> void FreeMem(T** matr, int n)
```

```
{
   for (int i = 0; i < n; i++)
      delete[] matr[i];
   delete[] matr;
}

//функция заполнения матрицы
template <typename T> void SetMtx(T** matr, int n)
{
   cout << "Enter a matrix:\n";
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         matr[i][j] = PowerMod(argX[i], j, Mod);
      }
   }

}

//функция печати матрицы
template <typename T> void PrintMtx(T** matr, int n)
{
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++)
         cout << matr[i][j] << " ";
      cout << endl;

   }
}

void inversion(ZZ** A, int N)
{
   ZZ temp;

   ZZ** E = new ZZ * [N];

   for (int i = 0; i < N; i++)
      E[i] = new ZZ[N];

   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
      {
         E[i][j] = 0.0;

         if (i == j)
            E[i][j] = 1.0;
      }

   for (int k = 0; k < N; k++)
   {
      temp = A[k][k];
      ZZ antiDet = conv<ZZ>("0");
      for (int ii = 1; ii < Mod; ii++) {
         ZZ antiDivider = temp * ii;
```

```
            ZZ antiDivider1 = antiDivider % Mod;
            if (antiDivider1 == 1)
            {
                antiDet = ii;
            }
        }
        for (int j = 0; j < N; j++)
        {
            A[k][j] *= antiDet;
            E[k][j] *= antiDet;
        }

        for (int i = k + 1; i < N; i++)
        {
            temp = A[i][k];

            for (int j = 0; j < N; j++)
            {
                A[i][j] -= A[k][j] * temp;
                E[i][j] -= E[k][j] * temp;
            }
        }
    }

    for (int k = N - 1; k > 0; k--)
    {
        for (int i = k - 1; i >= 0; i--)
        {
            temp = A[i][k];

            for (int j = 0; j < N; j++)
            {
                A[i][j] -= A[k][j] * temp;
                E[i][j] -= E[k][j] * temp;
            }
        }
    }

    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            A[i][j] = E[i][j] % Mod;

    for (int i = 0; i < N; i++)
        delete[] E[i];

    delete[] E;
}
```

ДОДАТОК В

Лістинг програми КПМАП-ZKSTARK-03

```
#include <NTL/ZZ_p.h>
#include <NTL/vector.h>
#include <iostream>
#include <fstream>
#include <ctime>
#include <cmath>
#include <vector>
#include <iostream>
#include <vector>
#include <chrono>
#include <iostream>
#include <sstream>
#include <string>
#include <stdexcept>
#include <iomanip>

NTL_CLIENT
using namespace NTL;
using namespace std;
using namespace std;
using namespace chrono;
/* unsigned long long = > не работает с отрицательными значениями!*/
//const ZZ P = ZZ(17);
const ZZ P = power(ZZ(2), 64) - power(ZZ(2), 32) + 1;
ZZ invNlenVIglob;
ZZ g =conv<ZZ>(7);

//ZZ g = conv<ZZ>(3);
ZZ number00 = conv < ZZ>("0");
ZZ number01 = conv < ZZ>("1");
ZZ number02 = conv < ZZ>("2");

//static_assert to check whether high_resolution_clock is actually aliased to a real clock
//https://stackoverflow.com/questions/38252022/does-standard-c11-guarantee-that-high-resolution-clock-measure-real-time-non
static_assert(
   std::is_same<high_resolution_clock, steady_clock>::value
   || std::is_same<high_resolution_clock, system_clock>::value,
   "high_resolution_clock IS NOT aliased to one of the other standard clocks!");
using maxres_sys_or_steady =
std::conditional<
   system_clock::period::den <= steady_clock::period::den,
   system_clock, steady_clock
>::type;
using maxres_nonsleeping_clock =
std::conditional<
   high_resolution_clock::is_steady,
   high_resolution_clock, maxres_sys_or_steady
>::type;
```

```
void initField() {
    ZZ_p::init(to_ZZ(P)); // Инициализация поля ZZ_p с модулем P
}

// Функция для вычисления примитивного корня из единицы n-ой степени
ZZ_p nthRootOfUnity(unsigned long long n) {

    ZZ_p g = ZZ_p(7); //- Златовласка

    //ZZ_p g = ZZ_p(3);// когда P = 17
    return power(ZZ_p(g), (P - 1) / n);
}

void FFT(Vec<ZZ_p>& x) {
    unsigned long long N = x.length();
    if (N <= 1) return;

    Vec<ZZ_p> even, odd;
    even.SetLength(N / 2);
    odd.SetLength(N / 2);

    for (unsigned long long i = 0; i < N / 2; ++i) {
        even[i] = x[2 * i];
        odd[i] = x[2 * i + 1];
    }

    FFT(even);
    FFT(odd);

    ZZ_p w = nthRootOfUnity(N);
    ZZ_p wn = to_ZZ_p(1);

    for (unsigned long long k = 0; k < N / 2; ++k) {
        ZZ_p t = wn * odd[k];
        x[k] = even[k] + t;
        x[k + N / 2] = even[k] - t;
        wn = wn * w;
    }
}


void IFFT(Vec<ZZ_p>& x, bool isTopLevel = true) {
    unsigned long long N = x.length();
    if (N <= 1) return;

    Vec<ZZ_p> even, odd;
    even.SetLength(N / 2);
    odd.SetLength(N / 2);
    for (unsigned long long i = 0; i < N / 2; ++i) {
        even[i] = x[2 * i];
```

```
        odd[i] = x[2 * i + 1];
    }

    IFFT(even, false);
    IFFT(odd, false);

    ZZ_p w = inv(nthRootOfUnity(N));
    ZZ_p wn = to_ZZ_p(1);

    for (unsigned long long k = 0; k < N / 2; ++k) {
        ZZ_p t = wn * odd[k];
        x[k] = even[k] + t;
        x[k + N / 2] = even[k] - t;
        wn *= w;
    }

    if (isTopLevel) { // Масштабируем результат на 1/N только на верхнем уровне рекурсии
        ZZ_p invN = inv(to_ZZ_p(N)); // Мультипликативный обратный к N
        for (unsigned long long i = 0; i < N; ++i) {
            x[i] *= invN;
        }
    }
}


ZZ invNlenVI(ZZ N) {
        ZZ invElofGroup = conv<ZZ>("1");
    ZZ temp = N;
    for (ZZ ii = conv<ZZ>(1); ii < P; ii++)
    {
        ZZ antiDivider = temp * ii;
        ZZ antiDivider1 = antiDivider % P;
        if (antiDivider1 == 1)
        {
            invElofGroup = ii;
        }
    }
    //unsigned long long invEl = conv< unsigned long long>(invElofGroup);
    return invElofGroup;
}

int main() {
    cout << "================ Verifier algorithm  & Interpolation by RFFT method ================" << endl;
    locale loc("ru_RU.UTF-8");
    auto start1 = high_resolution_clock::now();//Начальная (первая) точка отчета времени
    ofstream inOut1, inOut2, inOut3, inOut4;
    inOut1.open("Time_mesurement_STARK_RevMatr Interpolation time.csv", ios::app);
    inOut2.open("Time_mesurement_STARK_RevMatr Verification time.csv", ios::app);
    inOut3.open("Time_mesurement_STARK_RevMatr TOTAL time.csv", ios::app);
    inOut4.open("Mesurements checking.csv", ios::app);
    inOut1.imbue(loc);
    inOut2.imbue(loc);
    inOut3.imbue(loc);
```

```
    inOut4.imbue(loc);
    if (!inOut1.is_open() || !inOut2.is_open() || !inOut3.is_open() || !inOut4.is_open()) {
        std::cerr << " File was not opened successfully!" << std::endl;
        return 1;
    }
    inOut1 << " " << endl; // Используем табуляцию
    inOut2 << " " << endl; // Используем табуляцию
    inOut3 << " " << endl; // Используем табуляцию
    inOut4 << " " << endl;

    /*setlocale(0, "");
double time_spent = 0.0;
    clock_t begin = clock();

    double time_spent1 = 0.0;
    clock_t begin1 = clock();*/

    initField();

    Vec<ZZ_p> vi;
    Vec<ZZ> pi;
    //Vec<ZZ> results;
//=============================================================================================//
//============================ resverse FFT for Goldolocks field ==================================//
//=============================================================================================//
    //ZZ lenOfFibonacciSequence = power(ZZ(2), 2);// когда P = 17 и g = 3
       ZZ lenOfFibonacciSequence = power(ZZ(2), 3);
       // ZZ lenOfFibonacciSequence = power(ZZ(2), 6);
       // ZZ lenOfFibonacciSequence = power(ZZ(2), 7);
        //ZZ lenOfFibonacciSequence = power(ZZ(2), 8);
     //ZZ lenOfFibonacciSequence = conv<ZZ>("8");
//ZZ lenOfFibonacciSequence = power(ZZ(2), 6);
//ZZ lenOfFibonacciSequence = power(ZZ(2), 13);//0,065 sec
//ZZ lenOfFibonacciSequence = power(ZZ(2), 14);//0,129 sec
//ZZ lenOfFibonacciSequence = power(ZZ(2), 15);//0,266 sec
 //ZZ lenOfFibonacciSequence = power(ZZ(2), 16);//0,578 sec
    //ZZ lenOfFibonacciSequence = power(ZZ(2), 17);//1,137 sec
//ZZ lenOfFibonacciSequence = power(ZZ(2), 18);//2,276 sec
//ZZ lenOfFibonacciSequence = power(ZZ(2), 19); //4,666 sec
 //ZZ lenOfFibonacciSequence = power(ZZ(2), 20); // Вычисление числа - 9,45 sec
      //ZZ lenOfFibonacciSequence = power(ZZ(2), 21);// 19,256 sec
     //ZZ lenOfFibonacciSequence = power(ZZ(2), 22); // 39,255 sec
  //  ZZ lenOfFibonacciSequence = power(ZZ(2), 23);//80,358 sec
    //ZZ lenOfFibonacciSequence = power(ZZ(2), 24); // 164, 183 sec
    //ZZ lenOfFibonacciSequence = power(ZZ(2), 25); // error excessive length in vector::SetLength
    /*Необработанное исключение по адресу 0x00007FFA797A286E (ucrtbase.dll) в STARK-Arithmetization II-Verifier-01.exe:
     Запрашивается выход из программы в результате неустранимой ошибки.*/
    /*Метод SetLength() используется в библиотеке NTL
    для установки размера контейнеров, таких как векторы Vec.
    Этот метод является специфичным для NTL и не является стандартной частью языка C++.*/
     unsigned long long lenOfFibonacciSequenceLong;

    // Конвертация ZZ в строку через stringstream
```

```
    stringstream ss;
    ss << lenOfFibonacciSequence;
    string str = ss.str();

    try {
        lenOfFibonacciSequenceLong = stoull(str);  // Преобразование строки в unsigned long long

        //Vec<ZZ_p> vi;
        vi.SetLength(lenOfFibonacciSequenceLong);  // Установка длины вектора
        pi.SetLength(lenOfFibonacciSequenceLong);  // Установка длины вектора
        //results.SetLength(lenOfFibonacciSequenceLong);
        cout << "Длина вектора vi установлена успешно: " << vi.length() << endl;
        cout << "Длина вектора pi установлена успешно: " << pi.length() << endl;
    }
    catch (const invalid_argument& ia) {
        cerr << "Invalid argument: " << ia.what() << endl;
    }
    catch (const out_of_range& oor) {
        cerr << "Out of range: " << oor.what() << endl;
    }

    //pi.SetLength(lenOfFibonacciSequence);
    /*ZZ zzlenOfFibonacciSequence = conv<ZZ>((lenOfFibonacciSequence));
    ZZ temp2 = invNlenVI(zzlenOfFibonacciSequence);*/

    cout << "Length N of Fibonacci array is:  " << lenOfFibonacciSequence << endl;
   /* cout << "Inverse value of N is:  " << temp2 <<endl;*/
    /*invNlenVIglob = temp2;*/
    cout <<  endl;
    vi[0] = conv < ZZ_p>("1");
    vi[1] = conv < ZZ_p>("1");
    for (int j = 2; j < lenOfFibonacciSequence; j++)
    {
        vi[j] = vi[j - 1] + vi[j - 2];
       // pi[j] = pi[j - 1] + pi[j - 2];
        // cout << vi[j] << endl;
    }




//=============================================================================================================
=====================================//
 //==================================   REVERSE FFT
=============================================================================================//

//=============================================================================================================
=====================================//
    cout << endl;
    cout << "***** Interpolation part start *****" << endl;
    cout << "The result of reverse FFT  is: " << endl;

    IFFT(vi);
```

```
    /*for (unsigned long long i = 0; i < vi.length(); i++) {
        cout << "Result: [" << i << "]= " << vi[i] << std::endl;
    }*/

    /*clock_t end1 = clock();
    time_spent1 += (double)(end1 - begin1) / CLOCKS_PER_SEC;*/
    auto stop1 = high_resolution_clock::now();//вторая точка отчета времени
    auto start2 = high_resolution_clock::now();//третья точка отчета времени
    auto duration1 = duration_cast<duration<double>>(stop1 - start1); // Преобразуем в секунды с двойной точностью
    //cout << "Interpolation alg.: the elapsed time is %f" << time_spent1 << " seconds" << endl;
    cout << "***** Interpolation part finished *****" << endl;
    cout << endl;

//=================================================================================================================================
==================================//
 //=================================== VERIFIER PART
=============================================================================//

//=================================================================================================================================
==================================//
    for (int i = 0; i < vi.length(); i++) {
        pi[i] = rep(vi[i]);
    }

    //// Выведите элементы вектора vec_z
    //for (int i = 0; i < pi.length(); i++) {
    //    cout << pi[i] << " ";
    //}
    //cout << endl;
    ZZ number04 = (P - 1) / lenOfFibonacciSequenceLong;// упрощенная функция Эйлера для поля Златовласки
    ZZ gg = PowerMod(g, number04, P);//образующий элемент подгруппы порядка 2^i;

    ZZ* argX = new ZZ[lenOfFibonacciSequenceLong];
    for (int j = 0; j < lenOfFibonacciSequenceLong; j++)
    {
        argX[j] = PowerMod(gg, j, P);//подгруппа, которую образует образующий элемент подгруппы порядка 2^i

    }
    ZZ tmp01[3]{};
    tmp01[0] = 1;//при x^2
    tmp01[1] = (-(argX[lenOfFibonacciSequenceLong - 1] + argX[lenOfFibonacciSequenceLong - 2]));//при x^1
    tmp01[2] = ((argX[lenOfFibonacciSequenceLong - 1] * argX[lenOfFibonacciSequenceLong - 2]));//при x^0

    for (int i = 0; i < lenOfFibonacciSequenceLong / 2; i++) {
        ZZ temp = pi[lenOfFibonacciSequenceLong - i - 1];
        pi[lenOfFibonacciSequenceLong - i - 1] = pi[i];
        pi[i] = temp;
    }



    cout << "***** Verification part start *****" << endl;
    cout << "Polynom f(x): " << endl;
```

```
for (int i = 0; i < lenOfFibonacciSequenceLong; i++)
{

  // cout << pi[i] << " ";
}

ZZ* f_gx_ = new ZZ[lenOfFibonacciSequenceLong];
ZZ* f_g2x_ = new ZZ[lenOfFibonacciSequenceLong];
ZZ* subtractionArr = new ZZ[lenOfFibonacciSequenceLong];

//cout << endl;
cout << "Polynom f(g*x):" << endl;
int counter02 = lenOfFibonacciSequenceLong - 1;
for (int i1 = 0;i1 < lenOfFibonacciSequenceLong;i1++) {
    f_gx_[i1] = pi[i1] * PowerMod(gg, counter02, P);
    f_gx_[i1] = f_gx_[i1] % P;
    /*cout << "f_gx_ [" << i1 << "] = " << f_gx_[i1] << endl;*/
   // cout << f_gx_[i1] << " ";
    counter02--;
}
//  cout << endl;


cout << "Polynom f(g^2*x):" << endl;
ZZ number03 = PowerMod(gg, number02, P);
int counter03 = lenOfFibonacciSequenceLong - 1;
for (int i = 0;i < lenOfFibonacciSequenceLong;i++) {
    f_g2x_[i] = pi[i] * PowerMod(number03, counter03, P);
    f_g2x_[i] = f_g2x_[i] % P;
    /*cout << "f_g2x_ [" << i << "] = " << f_g2x_[i] << endl;*/
    //cout << f_g2x_[i] << " ";
    counter03--;
}
// cout << endl;
cout << "Polynom f(g^2*x) - f(g*x) - f(x):" << endl;
for (int i = 0; i < lenOfFibonacciSequenceLong; i++) {
    subtractionArr[i] = f_g2x_[i] - f_gx_[i] - pi[i];
    subtractionArr[i] = subtractionArr[i] % P;
    /*cout << "subtractionArr [" << i << "] = " << subtractionArr[i] << endl;*/
  //   cout << subtractionArr[i] << " ";
}
//  cout << endl;
delete[]f_gx_;
delete[]f_g2x_;
//delete[]x;

cout << "Polynom (x-a)*(x-b)*(f(g^2*x) - f(g*x) - f(x))" << endl;
const int NewSize = lenOfFibonacciSequenceLong + 2;
ZZ* mul = new ZZ[NewSize];


for (int i = lenOfFibonacciSequenceLong - 1; i >= 0; i--) {
    for (int j = 2; j >= 0; j--) {
```

```
        mul[i + j] += subtractionArr[i] * tmp01[j];//tmp02+
        mul[i + j] = mul[i + j] % P;

    }
}
delete[]subtractionArr;
int i10 = 0;
for (i10 = 0; i10 < NewSize; i10++) {
    /* cout << "mul [" << i10 << "]  =" << mul[i10] << endl;*/
   //  cout << mul[i10] << " ";
}
//   cout << endl;
cout << "Polynom (x-a)*(x-b)*(f(g^2*x) - f(g*x) - f(x))/(x^n - 1)" << endl;
//=============================================================================================
//=============================================================================================
//=============================================================================================
//
    //Polynom temp = p1;
const int degreeOfNumerator = NewSize;//degree = 7
const int degreeOfDivider = lenOfFibonacciSequenceLong;//degree = 6
int rdeg = degreeOfNumerator - degreeOfDivider;

ZZ* res = new ZZ[NewSize - lenOfFibonacciSequenceLong + 1]{};
ZZ* PolynomialDivider = new ZZ[degreeOfNumerator]{};

for (int i10 = 0; i10 < degreeOfNumerator; i10++) {
    PolynomialDivider[i10] = 0;
}

PolynomialDivider[1] = 1;
PolynomialDivider[degreeOfNumerator - 1] = 12;

//for (i10 = 0; i10 < degreeOfNumerator; i10++) {
//   /* cout << "PolynomialDivider [" << i10 << "]  =" << PolynomialDivider[i10] << endl;*/
//    cout << PolynomialDivider[i10] << " ";
//}

//************************************************Рабочее*************************************************
//cout << "degreeOfNumerator = " << degreeOfNumerator << endl;
//cout << "rdeg = " << rdeg << endl;
int counter04 = 1;

for (int i11 = 0; i11 < rdeg; i11++) {
    res[rdeg - i11 - 1] = mul[0 + i11] / PolynomialDivider[1];
    cout << "res [" << i11 << "]  =" << (res[rdeg - i11 - 1]) << endl;
    //cout << (res[rdeg - i11 - 1]) << " ";
    cout << endl;
    for (int j4 = 0; j4 <= degreeOfNumerator; j4++) {
        mul[0 + j4 + i11] -= PolynomialDivider[counter04] * res[rdeg - i11 - 1];
       // cout << "mul [" << 0 + j4 + i11 << "]  =" << mul[0 + j4 + i11] % P << endl;
        counter04++;
        if (counter04 == NewSize) {
            break;
```

```
        }

    }

    counter04 = 1;

}
delete[]res;
delete[]mul;
delete[]PolynomialDivider;
cout << "***** Verification part finished *****" << endl;
cout << " TIME: " << endl;
    //clock_t end = clock();
// рассчитать прошедшее время, найдя разницу (end - begin) и
    // деление разницы на CLOCKS_PER_SEC для перевода в секунды
//time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
////cout << endl;
//cout << "All program: the elapsed time is %f" << time_spent << " seconds" << endl;

//double time_spent2 = time_spent - time_spent1;
//cout << "Verification alg.: the elapsed time is %f" << time_spent2 << " seconds" << endl;
auto stop2 = high_resolution_clock::now();//четвертая точка отчета времени
auto duration2 = duration_cast<duration<double>>(stop2 - start2);
auto duration3 = duration_cast<duration<double>>(stop2 - start1);
auto duration4 = duration3 - duration2 - duration1;
inOut1 << fixed << setprecision(9) << duration1.count() << "" << "";;
inOut2 << fixed << setprecision(9) << duration2.count() << "" << "";;
inOut3 << fixed << setprecision(9) << duration3.count() << "" << "";;
inOut4 << fixed << setprecision(20) << duration4.count() << "" << "";;

inOut1.close();
inOut2.close();
inOut3.close();
inOut4.close();

return 0;
// end of main
}
```

ДОДАТОК Г

НАУКОВІ ПРАЦІ ЗА ТЕМОЮ МАГІСТЕРСЬКОЇ РОБОТИ

1. Аверков О.Ю., Кузнецов О.О., Лисицька І.В. Теоретична реалізація та тестування першого етапу роботи протоколу ZK-STARK «Арифметизація» / Вісник Харківського національного університету імені В. Н. Каразіна, сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління». - № - 2024. - . т. XX. С.X-XX. https://doi.org/10.26565/2304-6201-2022-53-01 - *стаття прийнята до публікації*

2. Аверков О.Ю., Кузнецов О.О. Теоретична реалізація етапу протоколу STARK «Арифметизація» / Збірник наукових праць IX науково-технічної міжнародної конференції «Комп'ютерне моделювання в наукоємних технологіях (КНМТ-2023)» - Харків: ХНУ ім. В.Н. Каразіна, 2023.- С.7-9.

3. Аверков О.Ю., Лисицька І.В. Результати тестування програмної реалізації першого етапу роботи протоколу ZK-STARK «Арифметизація» / збірник тез доповідей Всеукраїнської науково-практичної студентської конференції: IT-простір сьогодення: тенденції, інновації та перспективи розвитку (16 жовтня 2024 року, м. Харків, Україна) [Електронний ресурс]. – Харків : ХНУ імені В. Н. Каразіна, 2024. – С. 19-22. - Режим доступу: http://kbi.karazin.ua/wp-content/uploads/2024/11/proyekt_zbirnik-itmm.pdf (дата звернення: 19.11.2024).

4. Аверков О.Ю., Лисицька І.В. Програмна реалізація першого етапу роботи протоколу ZK-STARK «Арифметизація» та результати її тестування / Збірник наукових праць Десятої міжнародної науково-технічної конференції «Комп'ютерне моделювання у наукоємних технологіях» (КМНТ -2024), (м. Харків, 27-29 листопада 2024 року) – Харків: ХНУ імені В.Н. Каразіна, 2024 – С. – *тези прийняти до публікації*

5. Аверков О.Ю., Лисицька І.В. Результати тестування «Арифметизації» протоколу ZK-STARK на основі методу Гаусу та методу обернених матриць / Збірник наукових праць XIII міжнародної науково-технічної конференції «Інформаційні проблеми теорфї акустичних, радіоелектронних і телекомунікаційних систем» IPST-2024, (м. Харків, 11-13 листопада 2024 року) – Харків: НТУ «ХПІ», 2024. – С . 277-280.

ДОДАТОК Д

УЧАСТЬ ІЗ УСНИМИ ДОПОВІДЯМИ НА КОНФЕРЕНЦІЯХ ЗА ТЕМОЮ МАГІСТЕРСЬКОЇ РОБОТИ

Аверков О.Ю.

Тема доповіді: «Теоретична реалізація етапу протоколу STARK «Арифметизація»»

Харківський національний університет
імені В.Н. Каразіна

СЕРТИФІКАТ

учасника науково-технічної міжнародної конференції
«Комп'ютерне моделювання в наукоємних технологіях (КМНТ- 2023)»

брав(ла) участь у роботі конференції, яка проходила
25-27 жовтня 2023 року

Голова оргкомітету

Олена ТОЛСТОЛУЗЬКА

Аверков О.Ю.

Тема доповіді: «Результати тестування програмної реалізації першого етапу роботи протоколу ZK-STARK арифметизація»

Аверков О.Ю.

Тема доповіді:

Програмна реалізація першого етапу роботи протоколу ZK-STARK «Арифметизація» та результати її тестування