МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

НЕДЗЕЛЬСЬКИЙ Роман Володимирович

Генетичний метод пошуку параметрів еліптичних кривих / Genetic method for elliptic curves parameters search

спеціальність: 125 – Кібербезпека
освітньо-професійна програма –Кібербезпека
Кваліфікаційна робота

Виконав студент групи КБм -21
Р.В. Недзельський

Науковий керівник
к.т.н., доцент І.З. Якименко

Кваліфікаційну роботу допущено
до захисту:
«____» _____ ___2022 р.

Завідувач кафедри
_____ В.В.Яцків

ТЕРНОПІЛЬ - 2022

Факультет комп'ютерних інформаційних технологій

Кафедра кібербезпеки

Освітній ступінь «магістр»

спеціальність: 125 - Кібербезпека

освітньо-професійна програма –Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ В.В.Яцків

____" _____ 2021 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

ЯКИМЕНКО Ігор Зіновійович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Генетичний метод пошуку параметрів еліптичних кривих / Genetic method for elliptic curves parameters search
керівник роботи к.т.н., доцент І.З. Якименко
затверджені наказом по університету від 31 грудня 2021 року № 606
2. Строк подання студентом закінченої випускної кваліфікаційної роботи 16 листопада 2022 року.

3. Вихідні дані до кваліфікаційної роботи: завдання на випускну кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити:

– проаналізувати сучасний рівень захисту інформації в комп'ютерних системах (КС) та місце в ньому криптографії еліптичних кривих (ЕК);

– оцінити ефективність функціонування алгоритмів шифрування на ЕК та їх стійкість до атак;

– розробити ефективні алгоритми генерування параметрів ЕК стійких до різного виду атак;

– розробити швидкодіючі алгоритми шифрування за умови застосування ЕК;

– розробити спеціалізовані програмні засоби реалізації запропонованих алгоритмів та дослідити їх роботу

5. Перелік графічного матеріалу у роботі.

Геометрична інтерпретація додавання точок P і Q і подвоєння точки P.

Алгоритм генерування параметрів еліптичної кривої з використанням еволютивного підходу.

Ввід даних в програму.

Протокол шифрування на основі математичного апарату ЕК.

Дешифрування даних на основі запропонованого алгоритму.

Вивід даних при дешифруванні.

Шифрування інтерфейсом адміністратора вхідного тексту.

Дешифрування інтерфейсом адміністратора вихідного тексту.

6. Консультанти розділів кваліфікаційної роботи

	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 11 жовтня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз вимог стійкості криптографії еліптичних кривих	12.2021 р. – 03.2022 р.	
2	Теоретичні основи генерування параметрів та шифрування на основі еліптичних кривих	03.2022 р. – 05.2022 р.	
3	Програмна реалізація запропонованого алгоритму шифрування на еліптичній кривій	05.2022 р. – 11.2022 р.	

Студент _____ Недзельський Р.В.
(підпис)

Керівник роботи _____ к.т.н., доцент І.З.Якименко
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Генетичний метод пошуку параметрів еліптичних кривих» на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека» освітньо-професійної програми «Кібербезпека» написана обсягом 70 сторінки і містить 14 ілюстрації, 3 таблиця, 2 додатки та 21 джерело за переліком посилань.

Метою кваліфікаційної роботи є зменшення складності та підвищення швидкодії алгоритмів генерування параметрів ЕК і, як наслідок, підвищення ефективності захисту інформації в КС за умови застосування еліптичних кривих.

Методи досліджень. Для розв'язання поставлених задач у даній кваліфікаційній роботі використано: математичні основи теорії чисел та еліптичних кривих, алгебри Евкліда, теорії алгоритмів, теорії графів.

Результати дослідження: розроблено еволютивний алгоритм генерування параметрів еліптичних кривих, який дає змогу будувати стійкі криптоалгорими.

Розроблено ефективні алгоритми генерування параметрів ЕК на основі генетичного алгоритму стійких до різного виду атак, та швидкодіючі алгоритми шифрування за умови застосування ЕК.

Результати роботи можуть успішно застосовуватися при реалізації систем захисту інформаційних потоків у комп'ютерних системах.

Ключові слова: ЕЛІПТИЧНІ КРИВІ, ПАРАМЕТРИ ЕЛІПТИЧНИХ КРИВИХ, ЕВОЛЮТИВНИЙ АЛГОРИТМ, АЛГОРИТМИ ШИФРУВАННЯ.

ABSTRACT

Qualification work on " Genetic method for elliptic curves parameters search " for the degree of "Master" in the specialty 125 "Cybersecurity" educational and professional program "Cybersecurity" is written in 70 pages and contains 14 illustrations, 3 tables, 2 appendices and 21 source according to the list of links.

The purpose of the qualification work is to reduce the complexity and increase the speed of the EC parameter generation algorithms and, as a result, to increase the effectiveness of information protection in the CS, provided that elliptic curves are used.

Research methods. To solve the problems in this qualification work, the following mathematical foundations of number theory and elliptic curves, Euclid's algebra, theory of algorithms, theory of graphs were used.

Research results: an evolutionary algorithm for generating parameters of elliptic curves was developed, which makes it possible to build stable crypto-algorithms.

Effective algorithms for generating EK parameters based on a genetic algorithm resistant to various types of attacks, and fast-acting encryption algorithms under the condition of using EK have been developed.

The results of the work can be successfully applied in the implementation of information flow protection systems in computer systems.

Keywords: ELLIPTIC CURVES, ELLIPTIC CURVE PARAMETERS, EVOLUTIONARY ALGORITHM, ENCRYPTION ALGORITHMS

ЗМІСТ

ВСТУП

Актуальність. У зв'язку з повсюдним поширенням інформаційних систем у наші дні гостро постає питання захищеності даних, що зберігаються і передаються. З одного боку, спостерігається потреба суб'єктів інформаційних систем у надійному механізмі автентичності даних. З іншого — сучасні криптографічні протоколи, володіючи високим рівнем криптографічної стійкості, дозволяють суб'єктам систем передачі даних мати абсолютну впевненість у надійності комунікаційних систем [1].

Внаслідок цього останніми роками широке застосування отримали різні інформаційні системи, використовують засоби асиметричної криптографії, робота яких полягає у використанні відкритого ключа шифрування. Згідно з науковими дослідженнями вчених, серед криптосистем, що використовують відкритий ключ, найбільш стійкими до різноманітних атак є криптосистеми, засновані на еліптичних кривих (ЕК).

Результати цієї роботи ґрунтуються на дослідженнях таких вчених, як Ніл Кобліц, Рене Чуф, Джозеф Сільверман, Артур Аткін, Скотт Ванстоун, Альфред Менезес, Тацуакі Окамото Молодий дослідник Дону [2]. Деякі положення стосовно методик генерації криптографічно стійких еліптичних кривих отримали розвиток під час написання даної роботи.

Використання еліптичних кривих у криптосистемах запропонували американські вчені Ніл Кобліц та Віктор Міллер у 1985 р. [3–5].

Необхідністю розробки методів генерації криптографічно стійкої еліптичної кривої, що застосовується в реальних криптосистемах, є проблема вибору стійкої криптографічно еліптичної кривої для асиметричної криптосистеми, яка, в першу чергу, обумовлена трудомісткістю обчислень і складною реалізацією існуючих алгоритмів.

У ході дослідження виявлено два найбільш перспективні підходи до реалізації методів генерації криптографічно стійкої еліптичної кривої. Обговорюваними в роботі алгоритмами є методика «випадкового вибору» еліптичної кривої та підхід, що ґрунтується на застосуванні методу комплексного множення. Використання даних алгоритмів генерації сприяє підвищенню криптостійкості системи загалом.

У цій роботі наведено опис еліптичних кривих, дано їх загальну характеристику, а також розглянуто алгоритми генерації криптографічно стійких еліптичних кривих. Наведено їх докладний опис.

Метою роботи є дослідження алгоритмів генерації криптографічно стійких еліптичних кривих для визначення оптимального методу їх застосування в умовах відображення атаки зловмисника на секретні зашифровані дані, що передаються по каналу зв'язку. У процесі досягнення поставленої мети авторами було сформульовано та успішно вирішено такі завдання: дослідження еліптичних кривих з точки зору забезпечення безпеки та ефективності, виявлення методів генерації криптографічно стійких еліптичних кривих; аналіз існуючих підходів до генерації еліптичної кривої, що ґрунтується на випадковому переборі ЕК як математичних об'єктів, що використовують метод комплексного множення.

Основоположниками криптографії на основі еліптичних кривих стали американські вчені Ніл Кобліц та Віктор Міллер, які у 1985 році незалежно одна від одної запропонували системи криптографічного захисту на основі відкритого ключа, які використовують властивості адитивної групи точок на еліптичній кривій для реалізації шифрування. Згодом роботи цих дослідників стали основою криптографії на еліптичних кривих [3].

Вивченням питання генерації криптостійких еліптичних кривих займалися німецький вчений Геральд Байєр та швейцарський дослідник Йоганнес Баучман. У своїй спільній роботі «Методи генерації еліптичних кривих», опублікованій в 27 серпня 2002 як доповідь агентству з розвитку інформаційних технологій Японії. У

ції роботі вони описали підходи до створення еліптичних кривих над полем p (де p - просте число) та над полем 2^n, а також привели порівняння досліджуваних методів [6]. У Росії даною проблемою займалися В. В. Пилін у дисертації «Алгоритми та методи генерації еліптичної кривої для асиметричної криптосистеми» (2008 р.) та Н. В. Расторгуєва в дисертації «Підбір параметрів еліптичних кривих та аналіз їх криптостійкості для використання в асимметричній криптостійкості »(2014 р.) [7, 8].

Метою кваліфікаційної роботи є зменшення складності та підвищення швидкодії алгоритмів генерування параметрів ЕК і, як наслідок, підвищення ефективності захисту інформації в КС за умови застосування еліптичних кривих.

Об'єкт дослідження – процеси програмного захисту інформації в комп'ютерних системах за умови застосування ЕК.

Предмет дослідження – методи та алгоритми зменшення часової складності алгоритмів генерування параметрів ЕК та шифрування на основі математичного апарату ЕК.

Для досягнення поставленої мети в роботі необхідно розв'язати низку взаємопов'язаних задач:

– проаналізувати сучасний рівень захисту інформації в комп'ютерних системах (КС) та місце в ньому КЕК;

– оцінити ефективність функціонування алгоритмів шифрування на еліптичних кривих та їх стійкість до атак;

– розробити ефективні алгоритми генерування параметрів ЕК стійких до різного виду атак;

– розробити швидкодіючі алгоритми шифрування за умови застосування еліптичних кривих;

– розробити спеціалізовані програмні засоби реалізації запропонованих алгоритмів та дослідити їх роботу.

Методи дослідження. Для розв'язання поставлених задач у даній кваліфікаційній роботі використано: математичні основи теорії чисел та еліптичних кривих, алгебри Евкліда, теорії алгоритмів, теорії графів.

Наукова новизна роботи полягає у розробці еволютивного алгоритму генерування параметрів ЕК, який дає змогу будувати стійкі криптоалгорими.

Практичне значення отриманих результатів.

Розроблено ефективні алгоритми генерування параметрів ЕК на основі генетичного алгоритму стійких до різного виду атак, та швидкодіючі алгоритми шифрування за умови застосування ЕК.

Публікації та апробація ВКР.

1. Nykolaychuk Ya.M., Kasianchuk M.M., Yakymenko I.Z. Theoretical Foundations of the Modified Perfect form of Residue Number System. Cybernetics and Systems Analysis. 2016. Vol. 52, №2. P. 219-223.

2. Kasyanchuk M.M., Nykolaychuk Y.M., Yakymenko I.Z. Symmetric Cryptoalgorithms in the Residue Number System Cybernetics and Systems Analysis, 2021. Vol. 57(2). P. 329–336. ISSN: 1060-0396 (Print) 1573-8337 (Online)

Впровадження результатів ВКР. Результати роботи прийняті до впровадження в Інституті проблемно-орієнтованих комп'ютерних систем.

1 АНАЛІЗ ВИМОГ СТІЙКОСТІ КРИПТОГРАФІЇ ЕЛІПТИЧНИХ КРИВИХ

1.1 Порівняльний аналіз асиметричних криптосистем

Інформаційна безпека є однією з ключових проблем у передачі даних. Для безпечного обміну інформацією через мережу загального користування застосовуються різні криптографічні методи. Криптографічні методи широко класифікуються як симетричні та асиметричні. У симетричних методах ключі шифрування та дешифрування однакові або ключ дешифрування легко обчислюється з ключа шифрування. Проблема симетричного методу полягає в тому, що учасники повинні ділитися секретним ключем у безпечний спосіб, що важко [2].

Методи вирішення проблеми розподілу ключів за допомогою пари ключів. Обчислювально неможливо визначити ключ дешифрування, якщо знати лише криптографічний алгоритм і ключ шифрування [2].

Схема шифрування RSA, Elgamal і Paillier відноситься до асиметричних алгоритмів. Алгоритм RSA є одним із найстаріших і найпоширеніших методів шифрування [3]. У RSA пара ключів виводиться з добутку двох простих чисел, вибраних за спеціальними правилами [1]. Elgamal є фундаментальним, ефективним і простим асиметричним алгоритмом [4] і широко відомий як альтернатива RSA. Пайє є адитивним гомоморфним алгоритмом і широко відомий своєю семантичною безпекою.

У цьому документі представлено реалізацію та порівняння RSA, Elgamal і Paillier для змінних розмірів текстових файлів. Наша мета — обчислити час шифрування, час дешифрування, пропускну спроможність, розмір зашифрованого файлу та розмір розшифрованого файлу для кожного алгоритму, щоб визначити, які алгоритми перевершують інші за параметрами оцінки.

Оцінка продуктивності різних симетричних і асиметричних алгоритмів шифрування була широко досліджена в літературі. Сет та ін. [1] провів

порівняльний аналіз останніх досягнень в інформаційній науці ISBN: 978-1-61804-140-1 121 оцінки продуктивності симетричних і асиметричних алгоритмів шифрування, тобто AES, DES і RSA, з точки зору часу обчислення, використання пам’яті та виведення. байт на різні розміри файлів. Результат їхніх експериментів показав, що алгоритм DES працює краще серед інших з врахуванням часу шифрування, AES має найменше використання пам’яті, а алгоритм RSA створює найменший вихідний файл. Чалла та ін. al. порівняли продуктивність асиметричних алгоритмів RSA та NTRU на змінних розмірах текстових файлів з розміром ключа 51 біт і 20 біт для процесу шифрування і дешифрування відповідно [5].

Вони прийшли до висновку, що NTRU працює краще з точки зору шифрування, дешифрування та автентифікації, ніж RSA. Віджаялакшмі та ін. al. порівняли продуктивність асиметричних алгоритмів RSA та криптосистеми еліптичної кривої (ECC) за часом виконання та розміром пам’яті для процесу шифрування та дешифрування зі змінною довжиною слова та ключі різних розмірів. Їхні результати показали перевагу ECC над RSA щодо часу виконання та вимог до пам’яті [9].

Ельмінаам та ін. провели різні експерименти для оцінки продуктивності симетричних алгоритмів, наприклад шифрування AES (Rijndael), DES, 3DES, RC2, Blowfish і RC6, з точки зору енергоспоживання, енергоспоживання, різних розмірів ключів, типів даних і розміру пакета в [7]. Їх результати показали, що Blowfish працює краще, ніж інші алгоритми шифрування на змінному розмірі пакета. Ельмінаам та ін. порівнювали продуктивність цих алгоритмів на різних розмірах блоків даних, різних типах даних, споживанні енергії батареї, передачі даних через бездротову мережу [8] та експериментальні результати показали, що Blowfish знову показав кращі результати на змінному розмірі пакета. Міттал провів дослідження, щоб порівняти продуктивність з точки зору часу обробки та пропускної здатності симетричних алгоритмів, наприклад DES,

Алгоритми 3DES і AES (Rijndael) відрізняються в [6]. Алгоритм AES (Rijndael) продемонстрував менший час виконання порівняно з іншими алгоритмами, а процесор 2,00 ГГц (подвійний) показав найкращу пропускну здатність порівняно з іншими апаратними процесорами.

1.1.1 Параметри оцінювання

Автори вибрали наступні параметри для оцінки алгоритмів асиметричного шифрування RSA, ElGamal & Pallier для схем шифрування та дешифрування.

• Час шифрування (Час обчислення/Час відповіді) Час шифрування вважається часом, потрібним алгоритму шифрування для створення зашифрованого тексту зі звичайного тексту.[1]

• Час дешифрування (Час обчислення/Час відповіді) Час дешифрування вважається часом, потрібним алгоритму шифрування для відтворення звичайного тексту із зашифрованого тексту.

• Пропускна здатність Пропускна здатність дорівнює загальній кількості відкритого тексту в зашифрованих байтах, поділеній на час шифрування [1]. Чим вища пропускна здатність, тим вищою буде продуктивність.

• Розмір зашифрованого файлу Розмір зашифрованого файлу називається розміром зашифрованого файлу.

• Розмір розшифрованого файлу Розмір розшифрованого файлу називається розміром розшифрованого файлу.

1.1.2 Експериментальна установка та дані

Ми проводили експерименти на процесорі Intel(R) Core(TM) 2 Duo CPU 2,09 ГГц з 4 ГМ оперативної пам’яті в операційній системі Windows XP. Для експериментів використовується компілятор Python(x,y) 2.7.2.3. Ми провели експерименти з текстовими файлами розміром 68 КБ, 105 КБ, 124 КБ і 235 КБ. У цьому документі розміри приватного ключа вибрано відповідно до рекомендацій

NIST. Розмір приватного ключа 1024 біт для RSA, 160 біт для ElGamal і Paillier використовувався для експериментальних цілей, оскільки RSA забезпечує такий самий рівень безпеки на 1024-бітному розмірі ключа, що й надається від Elgamal і Paillier на 160 біт.

1.1.3 Результати та їх обговорення

На рисунку 1.1 показано порівняння часу шифрування в секундах між RSA, ElGamal і Paillier. Вторинна вісь Y представляє час шифрування Paillier через величезну різницю в часі Paillier порівняно з RSA та ElGamal. RSA показав кращу продуктивність порівняно з ElGamal і Paillier щодо часу шифрування, а ElGamal продемонстрував кращу продуктивність порівняно з RSA та Paillier щодо часу дешифрування, як показано на рисунку 1.2.

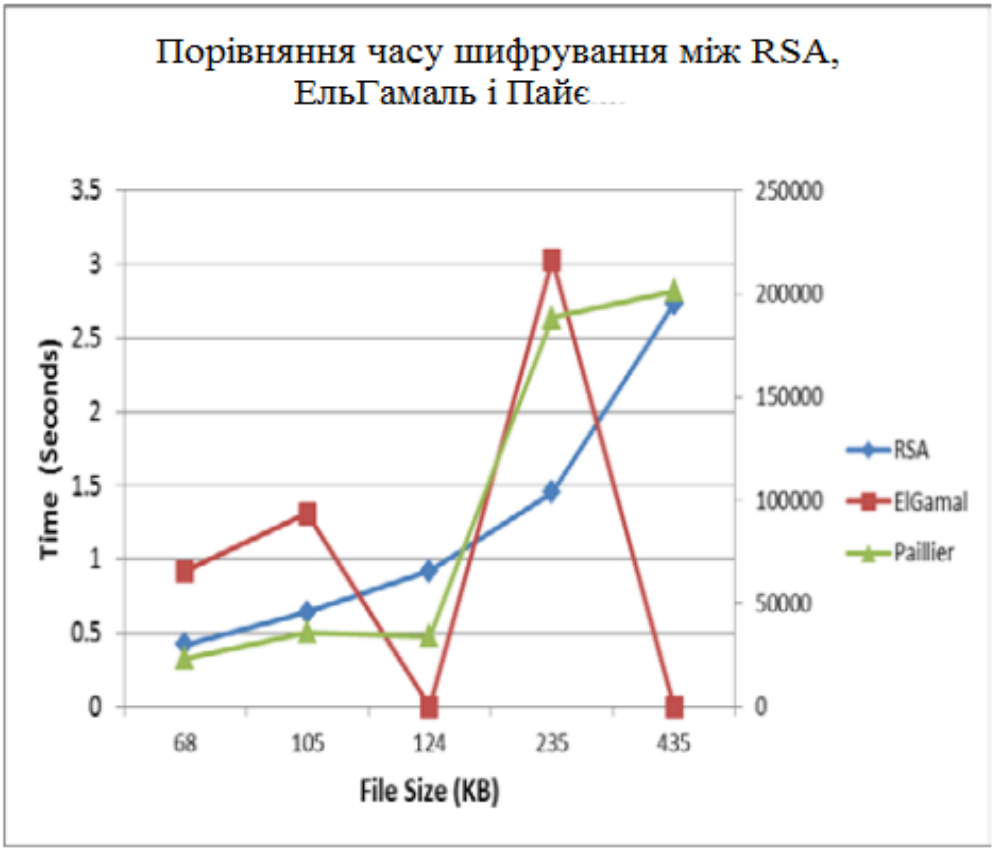


Рисунок 1.1 – Порівняння часу шифрування між RSA, ЕльГамаль і Пайє

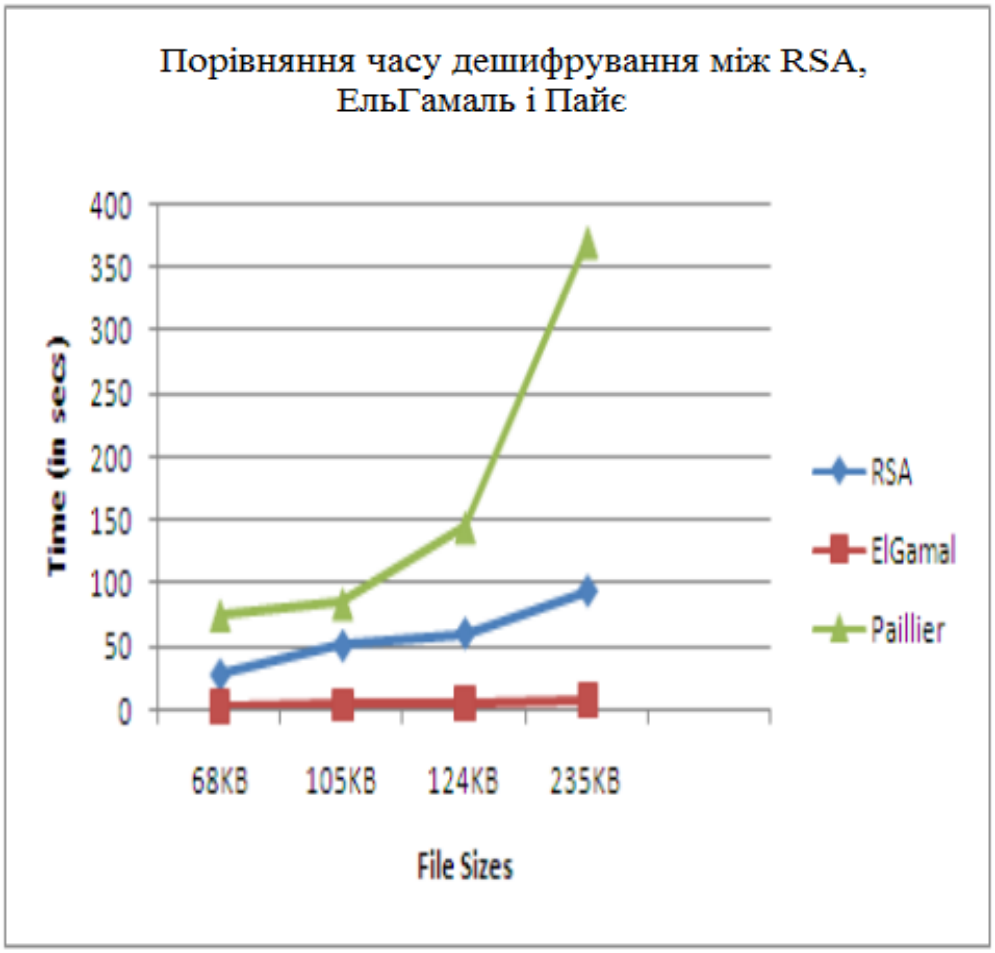


Рисунок 1.2 – Порівняння часу дешифрування між RSA, ЕльГамаль і Пайє

На рисунку 1.3 показано пропускну здатність RSA, ElGamal і Paillier для процесу шифрування, а на малюнку 4 показано пропускну здатність RSA, ElGamal і Paillier для процесу дешифрування. З рисунків 1.3 і 1.4 зроблено висновок, що RSA показав кращу пропускну здатність порівняно з ElGamal і Paillier у процесі шифрування, а Elgamal продемонстрував кращу пропускну здатність порівняно з RSA та Paillier у процесі дешифрування.

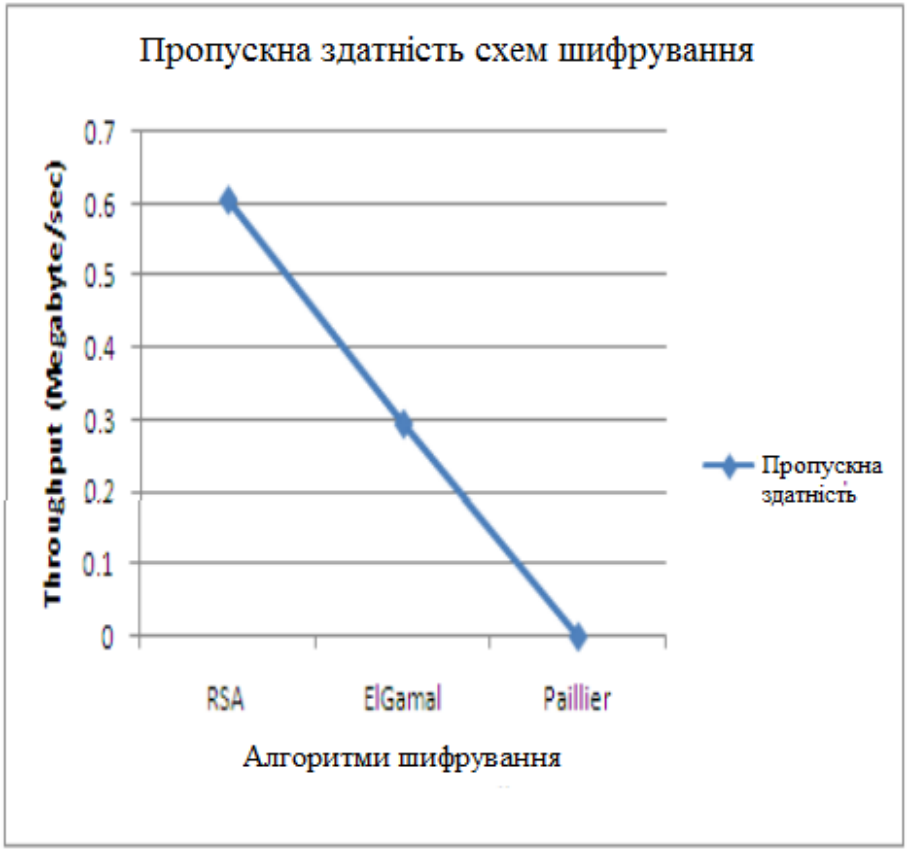


Рисунок 1.3 – Пропускна здатність алгоритмів шифрування

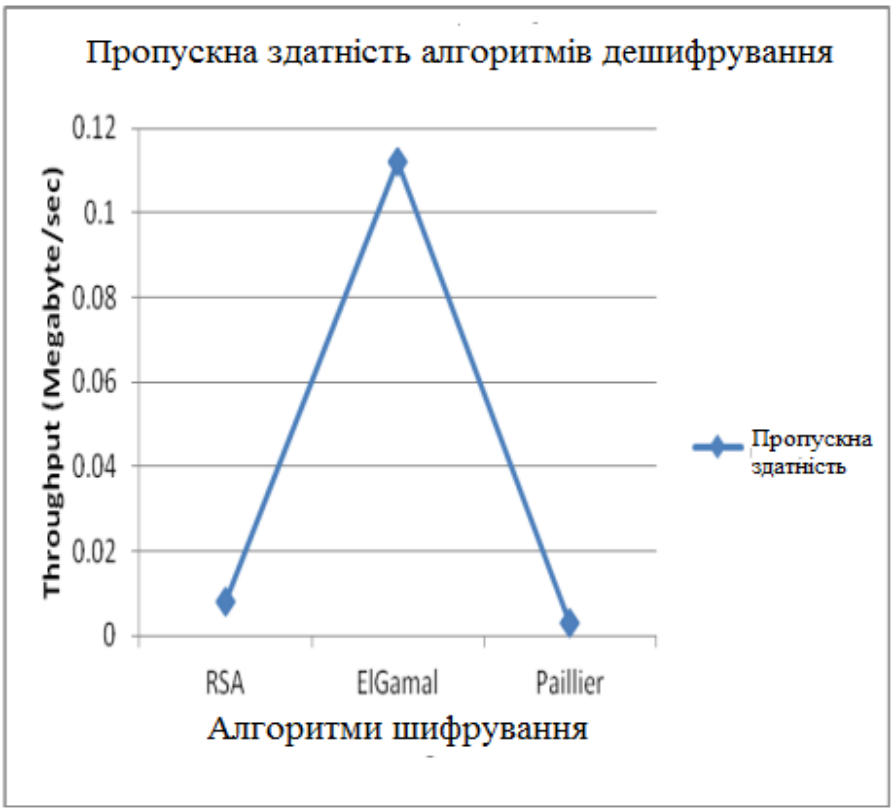


Рисунок 1.4 – Пропускна здатність алгоритмів дешифрування

На рисунках 1.5 і 1.6 показано порівняння розмірів файлів шифрування та дешифрування між RSA, ElGamal і Paillier. Paillier показав найгірший результат щодо розміру зашифрованого файлу, і розмір його зашифрованого файлу зростав експоненціально зі збільшенням розміру вхідного файлу. Найкращий результат показав RSA. ElGamal, RSA та Paillier продемонстрували таке ж експоненціальне збільшення розміру розшифрованого файлу зі збільшенням розміру вхідного файлу.

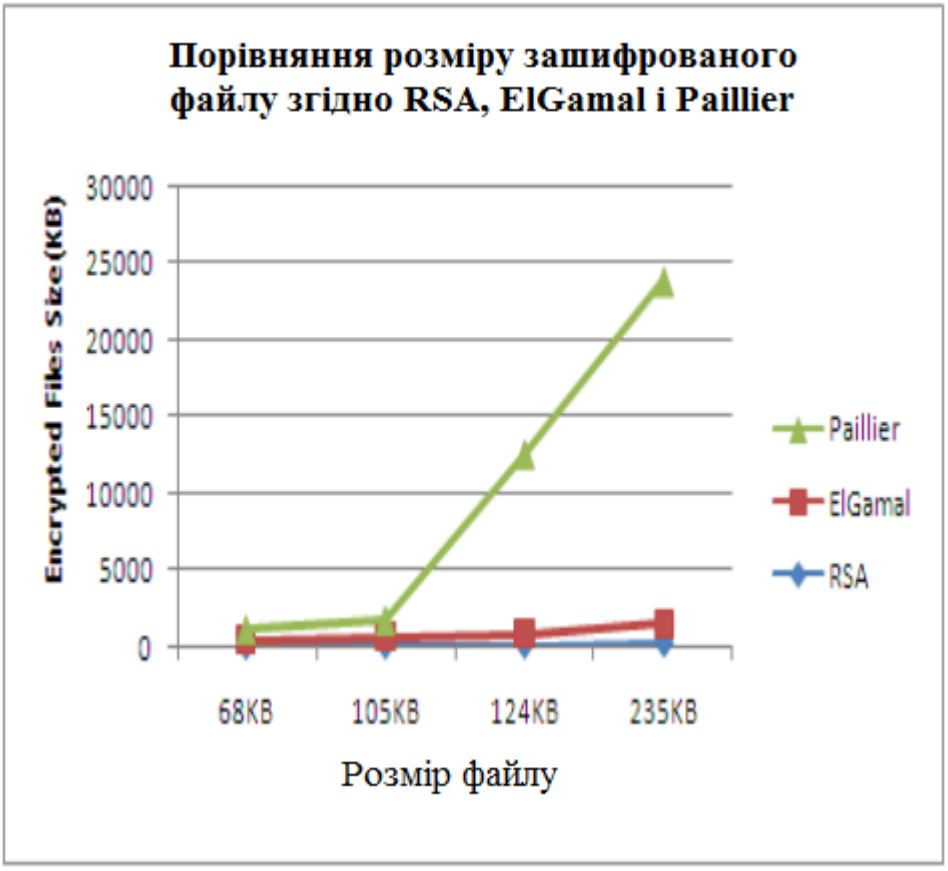


Рисунок 1.5 – Порівняння розміру зашифрованого файлу за RSA, ElGamal і Paillier

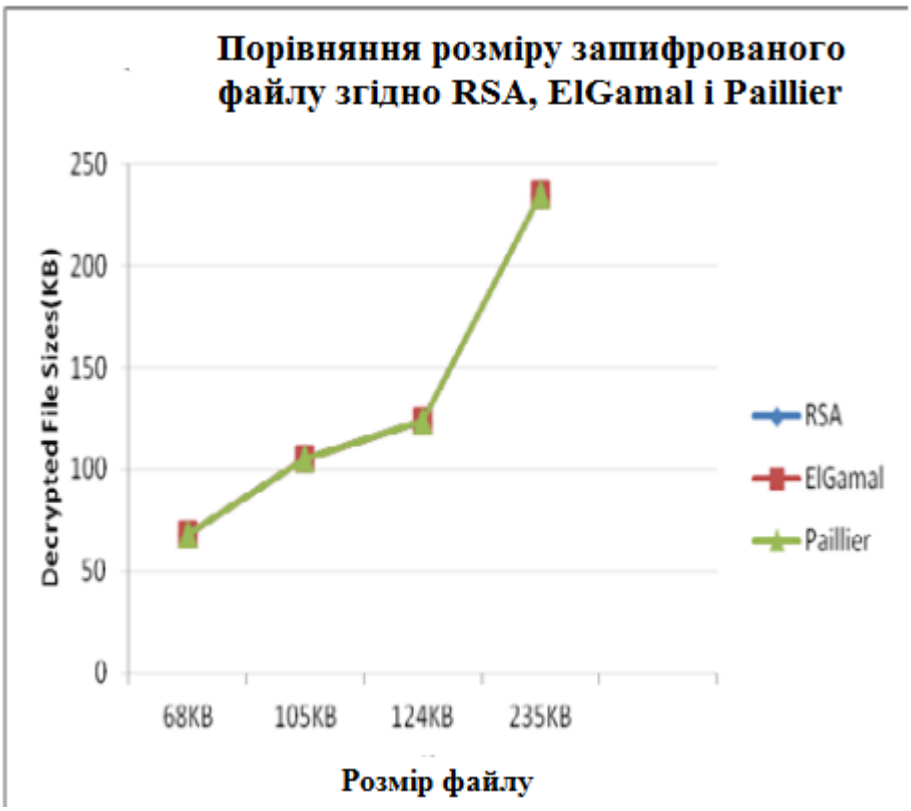


Рисунок 1.6 – Порівняння розміру зашифрованого файлу за RSA, ElGamal і Paillier

У цьому розділі представлено порівняння RSA, ElGamal & Pallier щодо часу шифрування, часу дешифрування, пропускної здатності, розміру зашифрованого та розшифрованого файлу. Було проведено різні експерименти для порівняння цих алгоритмів і зроблено висновок, що RSA показав кращі результати щодо часу шифрування, а ElGamal – щодо часу дешифрування. Пропускна здатність є найважливішим параметром, який демонструє продуктивність будь-якого алгоритму. Помічено, що пропускна здатність RSA краща в процесі шифрування, ніж усі інші, а ElGamal кращий у процесі дешифрування. Для RSA потрібно найменше місця для зберігання зашифрованих файлів. Розміри розшифрованих файлів усіх трьох алгоритмів, обраних для цієї статті, еквівалентні оригінальним розмірам файлів. Загальна продуктивність RSA краща, ніж у ElGamal і Paillier з точки зору параметрів, які використовуються в цій роботі.

1.2 Застосування еліптичних кривих у криптографії

Безпека криптосистем на еліптичних кривих визначається кількістю точок $E(F_p)$. Таким чином, щоб вирішити, чи підходить група раціональних точок для використання у криптографії, необхідно знати порядок її групи. Перший підхід, званий випадковим підходом, вибирає випадкову криву E. Порядок групи $E(F_p)$ визначається за допомогою алгоритмів підрахунку точок. На підставі підрахунку кількості точок можемо визначити, чи є ця група придатною для використання у криптографії. Якщо виявиться, що знайдена еліптична крива не задовольняє безпеку криптосистеми, вибирається інша еліптична крива [2].

Другий метод використовує теорію комплексного множення (CM-метод – complex multiplication method). Цей метод має досить велику відмінність від попереднього. У цьому методі насамперед виконується пошук відповідних точок групи. Це може бути здійснено без знання відповідних еліптичних кривих за вхідними даними. Після того, як безліч точок знайдено, еліптична крива визначається за допомогою комплексних формул множення [2].

Нехай $q = p$ – є простим числом, де $p \geq 5$. Еліптичною кривою над полем Fp є пара $E = (a,b) \in F_p^2$, де $4a^3 + 27b^2 \neq 0$. Точка на кривій E є рішенням $(x,y) \in F_p^2 2$ таким, що $y^2 = x^3 + ax + b$ або точка на нескінченності O, яка діє як одиничний елемент. Безліч точок E над полем F_p позначається $E(F_p)$. Наведена структура називається групою раціональних точок E над полем F_p [1]. Еліптична крива є криптографічно стійкою, якщо вона відповідає умовам безпеки та ефективності.

Спочатку розглянемо стійкість кривої з погляду безпеки. Безпека криптосистеми на еліптичних кривих полягає в складності вирішення проблеми дискретного логарифму в $E(F_p)$. На даний момент відомо кілька алгоритмів розв'язання дискретних логарифмів. Щоб зробити їх рішення неможливим, потрібно, щоб еліптична крива E задовольняла наступним умовам:

1) $\left|E\left(F_p^m\right)\right| = k \times r, r \geq 2^{160}$ - просте, $k > 0$ - ціле;

2) прості числа r та p різні;

3) порядок p у мультиплікативній групі F_p^* з F_r не менше ніж B , де $B \geq 20$.

Перша умова унеможливлює застосування загальних алгоритмів дискретного логарифму. Друга умова унеможливлює аномальну атаку. І, нарешті, остання умова виключає атаки на закриті ключі, такі як відомі атаки Менезеса, Окамото, Ванстоуна, і навіть атаки Фрея і Рюка [6]. Далі розглянемо криптографічну стійкість криптосистем на еліптичних кривих з погляду ефективності. Припустимо, що еліптична крива E, задана над кінцевим полем Fp, відповідає умовам безпеки. Якщо ця крива використовується в криптографічній системі, ефективність цієї системи залежить від ефективності арифметичних операцій в кінцевому полі Fp . Тому p має бути малим, наскільки це можливо. Це випливає з теореми Хассе:

$$\left(\sqrt{\left|E\left(F_p\right)-1\right|}\right)^2 \leq p \leq \left(\sqrt{\left|E\left(F_p\right)+1\right|}\right)^2. \qquad (1.1)$$

Відповідно, $\left|E\left(F_p\right)\right|$ також потрібно бути невеликим.

Розглянемо першу умову безпеки:

$$\left|E\left(F_p^m\right)\right| = k \times r, \qquad (1.2)$$

де $r \geq 2^{160}$ - просте, $k > 0$ – ціле (кофактор).

Безпека криптосистеми, у якій використовується $E\left(F_p\right)$, заснована на складності рішення проблеми дискретного логарифмування у підгрупі порядку r

в групі точок еліптичної кривої $E\left(F_p\right)$. Таким чином k повинно бути малим. В подальшому ми покращуємо першу умову:

$\left|E\left(F_p^m\right)\right| = k \times r, r \geq 2^{160}$ - просте, $k > 0$ – ціле.

Під третьою умовою розуміємо, що ендоморфізм кільця $End\left(E\left(F_p\right)\right)$ ЕК над алгебраїчним замиканням F_p є уявним квадратичним порядком.

У результаті цих трактувань, можна сказати, що еліптична крива $E\left(F_p\right)$ є криптографічно стійкою, якщо вона задовольняє умовам:

1) $\left|E\left(F_p^m\right)\right| = k \times r, r \geq 2^{160}$ - просте, $k \leq 4$ - ціле;

2) $p \neq r$;

3) $p^s \equiv 1 \bmod r, 1 \leq s \leq 20.$

У криптографії еліптичні криві розглядаються над двома типами кінцевих полів: простими полями непарної характеристики (Zp, p – просте число) та полями характеристики 2 $GF\left(2^m\right)$.

Для використання еліптичної криптографії учасники повинні узгодити всі параметри, що визначають ЕК, тобто набір параметрів криптографічного протоколу.

Еліптична крива визначається константами a, b, p. Абелева підгрупа точок є циклічною і задається однією породжувальною точкою G. Отже, для кінцевого поля Z_p, p >3 необхідно задати набір параметрів (p, a, b, G, n). Де n - порядок точки утворюючого елемента G.

Існує декілька рекомендованих наборів параметрів:

– NIST;

– SECG.

Для створення власного набору необхідно для початку вибрати набір параметрів і знайти еліптичну криву, що задовольняє цим параметрам.

Для пошуку кривої для заданого набору параметрів використовують два методи:

– вибрати випадкову криву, потім використовувати алгоритм підрахунку точок;

– вибрати точки, після чого побудувати криву за цими точками, використовуючи техніку множення.

NIST рекомендує 15 еліптичних кривих, багато з яких було отримано Jerry Solinas, деякі з них:

– поля F_p, де просте число p має довжину 192,224,256,384 або 521 біт.

– поля $GF\left(2^m\right)$, де m =163,233,283,409,571.

Розмір поля повинен принаймні в 2 рази перевищувати розмір ключа. Наприклад, для 128 бітного ключа рекомендується використовувати еліптичну криву над полем F_p , де p має довжину 256 біт.

1.3 Область застосування еліптичної криптографії

У сучасних вітчизняних стандартах формування та перевірки ЕЦП (електронного цифрового підпису) ГОСТ Р 34.10-2001 та ГОСТ Р 34.10.2012 також застосовуються алгоритми на еліптичних кривих, стійкість яких ґрунтується на складності обчислення дискретного логарифму в групі точок ЕК, а також на стійкості хеш – функції.

ЕК застосовують у сучасних системах:

1) Інформаційні системи організацій великого бізнесу. Підприємства великого бізнесу зацікавлені переважно у захисті своєї комерційної таємниці. У зв'язку з цим питання ціни у такому разі йдуть на другий план. Тут доцільним є

застосування сертифікованих засобів захисту інформації, таких як програмний комплекс CSP VPN.

2) Інформаційні системи організації середнього та малого бізнесу, наприклад, ідентифікатори RuToken ЕЦП, eToken ГОСТ.

3) Мобільна торгівля. У цій сфері поширене застосування різних протоколів передачі даних, наприклад протокол бездротової передачі даних WAP в стільникових телефонах, кишенькових комп'ютерах і т.д.

4) Інформаційні системи державних установ. Застосовують різні сертифікаційні комплекси ЗАСТАВ, CPN VPN Server.

5) Операції у банківських установах.

6) Інтернет - додатки. У разі поширене застосування криптографічних протоколів з алгоритмами на еліптичних кривих, наприклад, Secure Sockets Layer (SSL) – криптографічний протокол захищеності сокетів.

1.3.1 Безпека еліптичної криптографії

Безпека, що забезпечується підходом на основі еліптичних кривих, залежить від того, наскільки важким є завдання визначення k за даними kP і P. Це завдання називають проблемою логарифмування на еліптичній кривій. Логарифмування на ЕК за допомогою методу Поларда представлено у таблиці 1.1. Складність атаки на ключ у цьому випадку експоненціально пов'язана з довжиною ключа, тобто наростає дуже швидко й при деякій довжині ключа стає практично нереалізованою. Стійкість же криптосистем RSA й Ель-Гамаля субекспоненціальна. Практично це означає, що криптографія еліптичних кривих при однаковій стійкості має розмір модуля на порядок менше, ніж у старих криптосистем. Відомо [6, 7], що криптосистеми на ЕК із розміром поля в 160 біт забезпечує ту ж стійкість, що й традиційні криптосистеми з розміром модуля в 1024 біт. В таблиці 1.1 подані порівняльні характеристики алгоритмів шифрування.

Це кардинально змінює характеристики пам'яті й швидкодії. Більше того, виграш КЕК швидко прогресує зі збільшенням розміру модуля. Тому сьогодні, по суті, немає альтернативи цим криптосистемам. У майбутньому на зміну їм, швидше за все, прийдуть криптосистеми на гіпереліптичних кривих, які активно вивчаються останні роки.

Таблиця 1.1 – Порівняльні характеристики асиметричних криптосистем

Час на злом, MIPS років	Розмір ключа RSA/DSA	Розмір ключа КЕК	Відношення довжин ключів RSA/DSA
10^4	512	106	5:1
10^8	768	132	6:1
10^11	1. 024	160	7:1
10^20	2. 048	210	10:1
10^78	21.000	600	35:1

У результаті успішних досліджень як в області безпеки, так й у продуктивності виконання алгоритмів (або ефективності крипто-перетворення), наприкінці XX століття почався активний процес затвердження міжнародних і національних стандартів КЕК. Зокрема, в 2000 році затверджується національний стандарт цифрового підпису США FIPS 186-2-2000, в 2002 році - ДСТУ 4145-2002. Сьогодні налічується більше десятка корпоративних і національних стандартів КЕК.

ЕК використовують для побудови цифрового електронного підпису. Алгоритм ECDSA (Elliptic Curve Digital Signature Algorithm) прийнятий як стандарти ANSI, X9F1 і IEEE P1363. Перерахуємо переваги еліптичної криптографії:

– порівняно менша довжина ключа;

– швидкість роботи еліптичних алгоритмів набагато вища, ніж у класичних. Це як розмірами поля, і застосуванням ближчої комп'ютерів структури бінарного кінцевого поля.

– через невелику довжину ключа та високу швидкість роботи алгоритми асиметричної криптографії на еліптичних кривих можуть використовуватися в смарт – картах та інших пристроях з обмеженими обчислювальними можливостями.

Секретність і стійкість (стійкість щодо атак зловмисників) шифрування наведеного алгоритму згідно з структурною схемою полягає декількома важливими характеристиками, введеними в алгоритм:

1) Використання групи точок ЕК на базі вивчених кривих алгебраїчної геометрією збільшують швидкість шифрування, дешифрування. Дані переваги виявляються при порівнянні шифру еліптичної кривої з ключем однакової довжини в порівнянні з шифрами тільки на базі операцій на кінцевих полях. Таких як у криптосистемах з відкритим ключем, криптосистемі без передачі ключів, криптосистемі з електронним підписом. Ці системи використовують кінцеві поля з характеристикою p (з операціями над цілими числами, зведення в цілий ступінь по mod p), секретність яких полягає в використанні функції Ейлера і теореми Ейлера – Ферма. Групова операція добутку двох чисел замінюється на групову операцію додавання чисел, що значно зменшує обчислювальну складність алгоритмів і ймовірність помилки обчислень при роботі з великими цілими числами, оскільки величини всіх чисел не перевищують характеристики поля p.

2) Секретність шифру полягає у використанні хеш – функції випадкового числа, яке каналом зв'язку може передаватися. Несиметричність каналу зв'язку та протоколу обміну даними задовольняє всім сучасним вимогам стійкості шифрування еліптичної криптографії.

3) Секретність шифру полягає в унікальності алфавітного рядка, перебір яких вимагатиме близько 10^{30} років з розрахунку перебору 109 рядків алфавіту в 1

с. Це за умови, що зловмисник знає всі параметри еліптичної кривої і навіть утворює елемент групи G .

4) Простота виконання складання двох точок еліптичної кривої, робота з порівняно невеликими числами (порівняно з множенням двох чисел) значно збільшує швидкість шифрування та дешифрування. У той час як стійкість шифру на основі еліптичної криптографії значно вища, ніж для методів із застосуванням теореми Ейлера – Ферма, тому що вирішення зворотного завдання у разі еліптичного шифрування значно складніше при однаковому розмірі ключа, що пов'язано з груповими структурами в геометрії алгебри.

Тому виникають задачі, які необхідно вирішити для ефективного використання математичного апарату еліптичних кривих, а саме:

а) проаналізувати сучасний рівень захисту інформації;

б) оцінити ефективність засобів захисту інформаційних ресурсів із застосуванням ЕК;

в) розробити та дослідити швидкодіючі алгоритми генерування параметрів ЕК в задачах захисту інформації;

г) розробити програмний продукт реалізації генерування параметрів та дослідити ефективність розглянутих алгоритмів шифрування інформації на еліптичних кривих;

д) розробити програмний продукт шифрування інформації на еліптичних кривих.

2 ТЕОРЕТИЧНІ ОСНОВИ ГЕНЕРУВАННЯ ПАРАМЕТРІВ ТА ШИФРУВАННЯ НА ОСНОВІ ЕЛІПТИЧНИХ КРИВИХ

2.1 Постановка задачі. Еліптичні криві

У криптографічних методах використовують еліптичні криві над полем цілих чисел з характеристикою поля $r = 2$ або більше $r > 3$. Надалі ми розглядатимемо поле цілих чисел з характеристикою $r > 3$.

Криптографічні криві з характеристикою поля $r > 3$ мають канонічний вигляд:

$$y^2 = x^3 + ax + b, \tag{2.1}$$

де a,b – цілочисленні коефіцієнти кривої, p - просте досить велике число.

Як видно з формули (1.2), якщо точка з координатами (x, y) задовольняє рівняння (1.2), то рівняння (1.2) задовольняє також і точка з (x, $–y$). Під ЕК розуміють геометричне місце точок (1.2) доповнене нескінченно - віддаленою точкою.

Наступне число, яке називається дискримінантом кривої $\Delta = -16\left(4a^3 + 27b^2\right)$ не повинно бути рівним нулю (у цьому випадку відсутні точки самоперетину та точки повернення). Якщо дискримінант позитивний $\Delta > 0$, графік кривої має 2 частини, якщо $\Delta < 0$, то одну частину.

На множині точок ЕК визначають групу зі складання точок (розділ математики називається алгебраїчною геометрією). Сумою двох точок P,Q називається третя точка R, що лежить на прямій PQ і ЕК одночасно, і позначається як $R = P + Q$, тобто. $- R + P + Q = 0$.

Операцією групового додавання називають 3 точки ЕК, що задовольняють рівняння:

$$R'+P+Q = 0 \qquad (2.2)$$

Звідки видно, що $R'=-R$ (R', R – елементи взаємно зворотні за груповою операцією). З іншого боку, пряма паралельна координатній осі, перетинає рівно 2 точки ЕК (дзеркально симетричні відносно осі OX) і нескінченно віддалену точку, і взаємно обернені точки ЕК R', $-R$ – мають координати(x, y) і (x,$-y$) відповідно. Одиницею по груповому додаванню визначають геометрично нескінченно віддалену точку і позначають 0. Отже, для групової операції додавання, необхідно провести січну через точки P,Q і відобразити дзеркально точку R, $R' = -R$.

Можливі окремі випадки:

1) $P + Q$ - січна пряма вироджується в дотичну $R'+2P = 0$

2) $P+Q+0=0$ еквівалентно $P=-Q$ точки P,Q мають однакові абсциси. Наступною точкою по додаванню вибирають $Q+0=Q$.

3) $P+P+0=0$ еквівалентно $P=0$ – січна пряма одночасно є вертикальною прямою та дотичною.

Криптографія використовує кінцеві циклічні абелеві групи з елементом G, що породжує. При цьому будь-яку точку ЕК циклічної групи $0 < k \le n0$ одержують за формулою $P_k = \underset{k}{(GGG\ldots G)}$. Порядком групи точок ЕК називається число $n0$, таке що $P_{n0}= O$ – нульовий елемент групи.

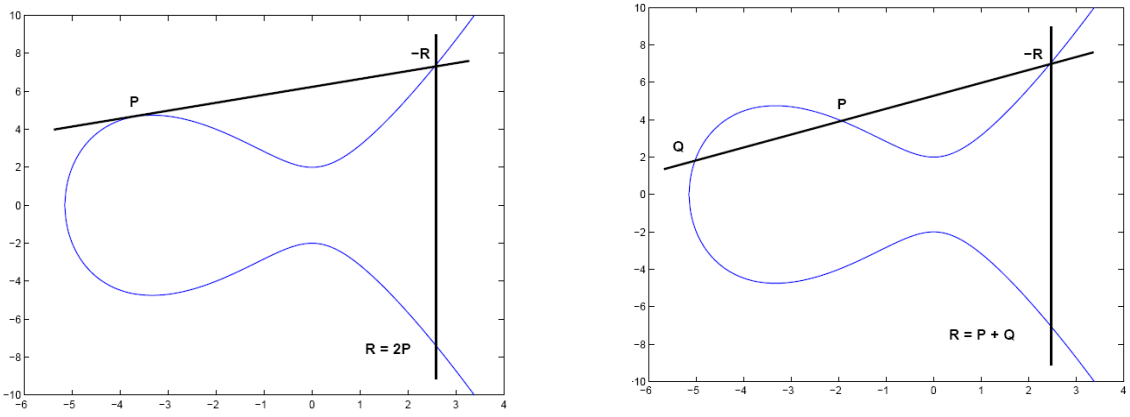
Знаючи породжувальний елемент групи G, можна скласти таблицю всіх точок еліптичної кривої, відносно операції додавання з порядком $k > n0$ всі точки періодично повторюються $P_k = P_{k-n0\cdot s}$, де $0 \le k - n0 \cdot s \le n0 - 1, s \in N$. Залежно від загальної ситуації та окремих випадків 1), 2), 3) координати точок ЕК обчислюють за формулами (індекси 1 та 2 відповідають точкам P,Q відповідно).

$$\begin{cases} x = \left(\dfrac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 = k^2 - x_1 - x_2 \\ y = -y_1 + \left(\dfrac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x) = -y_1 + k\left(2x_1 + x_2 - k^2\right) \end{cases}, \tag{2.3}$$

$$\begin{cases} x = \left(\dfrac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 = k^2 - 2x_1 \\ y = -y_1 + \left(\dfrac{3x_1^2 + a}{2y_1}\right)(x_1 - x) = -y_1 + k\left(3x_1 - k^2\right) \end{cases}, \tag{2.4}$$

де кутовий коефіцієнт прямої яка проходить через дві точки ЕК рівний $k = \dfrac{y_2 - y_1}{x_2 - x_1} = \dfrac{y - y_1}{x - x_1}$, точка (x, y) є ковзною по прямій (змінній).

Геометрична інтерпретація додавання та пошуку кратних точок на ЕК представлено на рисунку 2.1.



а б

Рисунок 2.1 - Геометрична інтерпретація додавання точок *P* і *Q* (а) і подвоєння точки *P* (б)

Для точок $(x_1,\ y_1),\ 2(x_2,\ y_2),(x,\ y)$ отримаємо: $y^2 = x^3 + ax + b$, $y_1{}^2 = x_1{}^3 + ax_1 + b$, $y_2{}^2 = x_2{}^3 + ax_2 + b$

Вирахуємо $(y_2 - y_1)(y_2 + y_1) = (x_2 - x_1)\left(x_2^2 + x_1x_2 + x_1^2\right) + a(x_2 - x_1)$, звідки $k = \frac{y_2 - y_1}{x_2 - x_1}$, $k(y_2 + y_1) = \left(x_2^2 + x_1x_2 + x_1^2\right) + a$, аналогічно $k = \frac{y - y_1}{x - x_1}$, $k(y + y_1) = \left(x^2 + x_1x + x_1^2\right) + a$ і остання формула $k = \frac{y - y_2}{x - x_2}$, $k(y + y_2) = \left(x^2 + xx_2 + x_2^2\right) + a$.

Вирахуємо із третьої формули другу і отримаємо $k(y_2 - y_1) = x(x_2 - x_1) + (x_2 - x_1)(x_2 + x_1)$, звідки $k^2 = x + x_2 + x_1 \Leftrightarrow x = k^2 - x_2 - x_1$. Для кординати $y = y_1 + k(x - x_1) = y_1 + k\left(k^2 - 2x_1 - x_2\right)$. Залишається згадати, що для групової операції потрібно вибрати дзеркальну точку:

$$(x,-y) = \left(k^2 - x_2 - x_1, -y_1 + k\left(-k^2 + 2x_1 + x_2\right)\right) \tag{2.5}$$

Формула (1.4) доведена.

У випадку переходу січної до дотичної отримаємо $x_2 = x_1, x = k^2 - 2x_1$. В подальшому диференціюємо рівняння (1.1) по x, і отримаємо:

$2yy' = 3x^2 + a \Leftrightarrow k = y' = \frac{3x^2 + a}{2y} = \frac{3x_1{}^2 + a}{2y_1}$, тоді з формули (1.6) отримаємо:

$$(x,-y) = \left(k^2 - 2x_1, -y_1 + k\left(-k^2 + 3x_1\right)\right), k = \frac{3x_1{}^2 + a}{2y_1}. \tag{2.6}$$

Таким чином, доведено формулу (2.4).

Циклічну групу утворюють з безлічі точок еліптичної кривої (рівняння 1.1)), пов'язаних геометричною груповою структурою (формули (2.3),(2.4)),

доповнюють польовою цілочисленною структурою за модулем простого числа p, тобто. замість (1.1) вирішують порівняння:

$$y^2 = \left(x^3 + ax + b\right) \bmod p\,. \tag{2.7}$$

Зрештою ми користуємося формулами (2.3), (2.4) і (2.7)., отримуючи послідовно всі точки ЕК циклічної абелевої групи.

Нагадаємо правила порівнянь, позначимо цілі числа $\bar{x}, \bar{y}, \bar{x}_1, \bar{y}_1$ (всього можливо $p - 1$ різних залишків по mod p) $\bar{x}_1 = \bar{x} \bmod p, \bar{y}_1 = \bar{y} \bmod p$:

1) $\left(\bar{x} \pm \bar{y}\right) \bmod p = \left(\bar{x}_1 \pm \bar{y}_1\right) \bmod p\,.$
2) $\left(\bar{x} \cdot \bar{y}\right) \bmod p = \left(\bar{x}_1 \cdot \bar{y}_1\right) \bmod p\,.$ (2.8)
3) $\left(\frac{\bar{x}}{=}{\bar{y}}\right) \bmod p = \left(\bar{x}_1 \cdot \bar{y}_2\right) \bmod p : \bar{y}_2 \cdot \bar{y}_1 \equiv 1 \bmod p$. В цьому випадку $\bar{y}_2 = \bar{y}_1^{-1}$, $\bar{y}_1$ називаються взаємно зворотними згыдно аксіоматикою порівнянь чисел за модулем р.

Як видно з формул (2.3) та (2.4) координати точок ЕК є раціональними числами, якщо перші 2 точки кривої також є раціональними, тобто геометрична групова операція залишає координати точок раціональними і надалі. Правила (2.8) звужують безліч раціональних точок, як правило, до кінцевої множини точок ЕК з цілими координатами x, y. Аналіз формул (2.3) та (2.4) показує, що якщо кутовий коефіцієнт прямої приймає цілочисельні значення у сенсі формули (2.8), то координати x, y будуть і далі цілими. Таким чином, необхідно вирішити порівняння:

$$\begin{cases} \left(\frac{y_2 - y_1}{x_2 - x_1}\right) \equiv (y_2 - y_1) \bmod p \cdot (x_2 - x_1)^{-1} \bmod p, (x_2 - x_1)(x_2 - x_1)^{-1} \equiv 1 \bmod p \\ \left(\frac{3x_1^2 + a}{2y_1}\right) \equiv (3x_1^2 + a) \bmod p \cdot (2y_1)^{-1} \bmod p, (2y_1) \cdot (2y_1)^{-1} \equiv 1 \bmod p \end{cases}. \quad (2.9)$$

Наведемо приклад пошуку оберненого елемента при вирішенні порівнянь, який є найбільш складним на практиці.

$x^{-1} = \frac{1}{x} \bmod 11$: *якщо* $x \equiv 1 \bmod 11$, $x^{-1} = 1 \Leftrightarrow 1 \equiv 1 \bmod 11$

якщо $x \equiv 2 \bmod 11$, $x^{-1} = 6 \Leftrightarrow 2 * 6 = 12 \equiv 1 \bmod 11$

якщо $x \equiv 3 \bmod 11$, $x^{-1} = 4 \Leftrightarrow 3 * 4 = 12 \equiv 1 \bmod 11$

якщо $x \equiv 4 \bmod 11$, $x^{-1} = 3 \Leftrightarrow 4 * 3 = 12 \equiv 1 \bmod 11$

якщо $x \equiv 5 \bmod 11$, $x^{-1} = 9 \Leftrightarrow 5 * 9 = 45 \equiv 1 \bmod 11$

якщо $x \equiv 6 \bmod 11$, $x^{-1} = 2 \Leftrightarrow 6 * 2 = 12 \equiv 1 \bmod 11$

якщо $x \equiv 7 \bmod 11$, $x^{-1} = 8 \Leftrightarrow 7 * 8 = 56 \equiv 1 \bmod 11$

якщо $x \equiv 8 \bmod 11$, $x^{-1} = 7 \Leftrightarrow 8 * 7 = 56 \equiv 1 \bmod 11$

якщо $x \equiv 9 \bmod 11$, $x^{-1} = 5 \Leftrightarrow 9 * 5 = 45 \equiv 1 \bmod 11$

якщо $x \equiv 10 \bmod 11$, $x^{-1} = 10 \Leftrightarrow 10 * 10 = 100 \equiv 1 \bmod 11$

Короткий опис алгоритму побудови послідовності точок ЕК.

1. На початковому етапі знаходимо обернений елемент за модулем у (2.9) до $2y_1$, або $x_2 - x_1$.
2. Знаходимо числа $k_1 = (y_2 - y_1) \bmod p \cdot (x_2 - x_1)^{-1} \bmod p$, або $k_1 = (3x_1^2 + a) \bmod p \cdot (2y_1)^{-1} \bmod p$.
3. Знаходимо:

$$\begin{cases} x = (k^2 - x_1 - x_2) \bmod p \\ y = (-y_1 + k(2x_1 + x_2 - k^2)) \bmod p \end{cases}. \quad (2.10)$$

або згідно формул:

$$\begin{cases} x = \left(k^2 - 2x_1\right) \bmod p \\ y = \left(-y_1 + k\left(3x_1 - k^2\right)\right) \bmod p \end{cases}. \quad (2.11)$$

Координати точки з використанням формул (2.9), (2.10), (2.11) дають нову точку ЕК з урахуванням усіх зазначених вимог.

Перший основний етап алгоритму полягає у пошуку порядку циклічної обелевої групи, тобто. одиниці за додаванням - *O* та записи таблиці точок ЕК, починаючи з утворюючого елемента G . Як було сказано, елементи шифру з номером $k > n0$ знаходяться по формулі $P_k = P_{k-n0 \cdot s} \cdot 1 \le k - n0 \cdot s \le n0 - 1, s \in N$.

2.2 Еволютивний алгоритм генерування параметрів ЕК

Окрім суттєвих переваг при використанні математичного апарату ЕК, існують задачі, які є досить складними. До них можна віднести –генерування параметрів еліптичної кривої; обчислення порядку еліптичної кривої; задача дискретного логарифма.

Найбільщш важливою задачею в криптографії ЕК є генерування параметрів, а саме для кривої виду:

$$y^2 = x^3 + ax + b (\bmod p) \text{ де } a,b \in F_p^2,\ (x,y) \in F_p^2 \quad (2.12)$$

над простим полем GF(p):

а) модуля перетворення р групи точок ЕК, яке повинно бути простим і задовольняти нерівність $p > 2^{255}$. Крім того верхня межа його встановлюється в залежності від конкретної реалізацієї криптосистеми. При створення електронно-цифрового підпису розміром 512 біт, як у стандарті ГОСТ 34.310-95, тоді

необхідно, що p задовольняло нерівність $p < 2^{256}$. Тому, з врахуванням даних обмежень, будемо вважати, що p належить діапазону $2^{255} < p < 2^{256}$;

б) важливим параметром при реалізації криптосистеми ЕК є порядок групи точок ЕК просте число q [2].

Генерування простих чисел можна здійснювати з використанням таких процедур:

а) використання вже згенерованих q зі стандарту ГОСТ 34.310-95;

б) генерування простого числа випадковим чином, яке буде задовольняти нерівність $2^{255} < p < 2^{256}$;

в) генерування сильного простого числа;

г) найбільш важливими параметрами при шифруванні є коефіцієнти $a,b \in F_p^2$, які задають ЕК.

Для генерування простого числа можна скористатися розробленим алгоритмом, який дозволяє ефективно шукати прості числа [30].

В роботі [70] прозглядається метод пошуку простих чисел за наступним алгоритмом.

Алгоритм пошуку найбільшого простого числа.

1. Нехай відоме останнє просте число $p_i - \max$, представляємо його у двійковій системі числення..

2. З використанням розмежованої СЗК та властивість періодичності, рекурентно обчислюємо залишок числа p_i по заданому модулю (формула 2.13):

$$\begin{cases} c_{i+1} = c_i \cdot 2(\text{mod } p), c_1 = 1, a_1 = 1 \\ b_{i+1} = (c_{i+1} \cdot a_i + b_i) \text{ mod } p, b_1 = 1, i = 1, a_i = 0,1 \end{cases}. \quad (2.13)$$

Таким чином знаходиться зображення цього числа в СЗК до числа простих чисел, які не перевищують $\sqrt{p_i}$ - половина розряду p_i

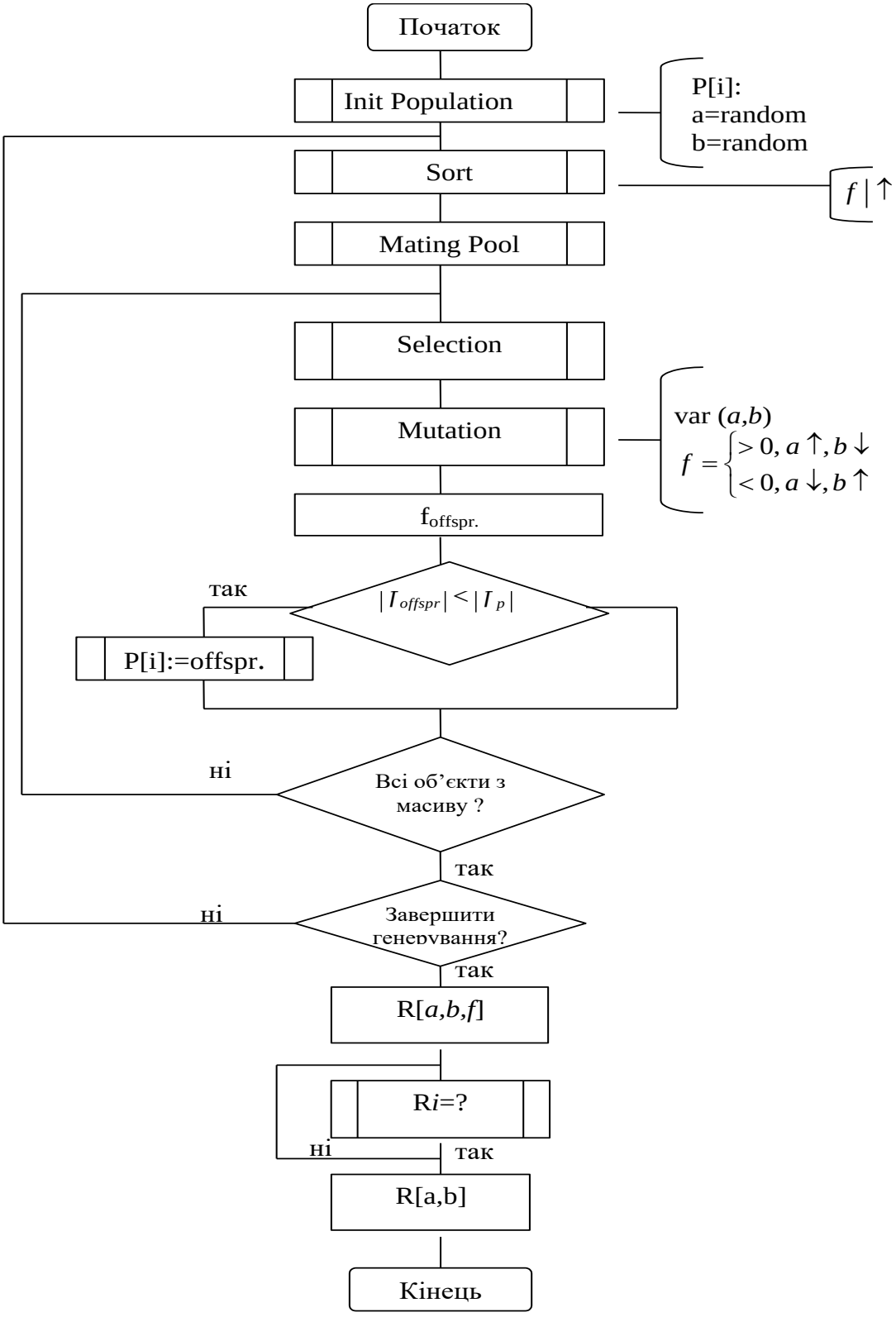


Рисунок 2.2 – Алгоритм генерування параметрів еліптичної кривої з використанням еволютивного підходу

3. Додаємо до залишків число 2 по всіх модулях, до тих пір, поки по одному з модулів отримаємо 0. Таким чином ми можемо пропускати числа кратні модулю, тобто коли залишок рівний 0. Операція повторюється поки не знайдеться найбільше просте число.

До основних переваг цього методу слід віднести використання СЗК, що надає змогу проводити операції над числами, які не перевищують модулі, що суттєво підвищує ефективність та швидкодію алгоритму. Отже, використання алгоритму 2.1 генерування модуля перетвореня групи точок р ЕК, спрощує процедуру та зменшує часові витрати для генерування прости х чисел.

А для генерування параметрів, які задають ЕК доцільно використати еволютивний підход на основі алгоритму А.12.4 стандарту IEEE P1363 [14]. Проведено розробку алгоритму генерування параметрів ЕК вище зазначеним підходом (див. рис. 2.2), та проведено аналіз часових характеристик.

Важливим моментом в процесі розробки еволютивних алгоритмів є задання критерію, тобто цільової функції, яка дає змогу здійснювити відбір нових об’єктів. На основі критерію оцінки гладкості кривої (див. алгоритм А.12.4 стандарту IEEE P1363), видбір параметрів здійснюється на основі виразу:

$$f = c \cdot b^2 (\mathrm{mod}\, p) - a^3 (\mathrm{mod}\, p), \qquad (2.14)$$

де $a, b \in GF(p)$.

Сутність запропонованого алгоритму відповідає типовим еволютивним алгоритмам і полягає в наступному (див. рис. 2.2). На початковому етапі у Init Population здійснюється генерування випадковим чином об’єктів з параметрами ЕК *a* та *b*. В подальшому сортуються параметри по зростанню значення *f*. Після вибираються 1024 найкращі, за принципом мінімального абсолютного значення критерію *f*, які заносяться у масив Mating Pool. В подальшому відбувається ітеративна процедура модифікації об’єктів P_i з масиву Mating Pool на основі

використання операції мутації. Слід відмітити, що оператор мутації дозволяє враховувати напрям зміни коефіцієнтів *a, b*, що дало змогу суттєво підвищити ефективність та швидкодію алгоритм. На наступному етапі обчислюється значення критерію *f* для отриманого шляхом мутації нового об'єкта. Перевіряється чи знайдене абсолютне значення менше за попереднє, якщо так, то новий об'єкт записується на місце попереднього (предка). На завершальному етапі перевіряється на невиродженість отриманої ЕК. На основі запропонованого алгоритму розроблено програмний засіб для отримання пари коефіцієнтів *a, b*.

Незважаючи на вказані переваги, слід зазначити, що запропонований алгоритм має функціональні обмеження, а саме: простий перебір параметрів (a, b), хоча і з врахуванням операторів мутації та напрямків зміни коефіцієнтів a, b. Отже, для ефективного генерування параметрів ЕК потрібно використовувати підходи, які б дозволили пришвидшити роботу алгоритму.

2.3 Методи генерування криптографічно стійких еліптичних кривих

Продовжуючи дослідження, розглянемо два методи пошуку криптографічно стійких еліптичних кривих. Для цього повиння бути вирішена наступна задача: нехай r_0 і k_0 додатні цілі числа, де $r \geq 2^{160}$ - просте, $k \leq 4$ - ціле. Потрібно занайти еліптичні криві, коефіцієнти яких $k \times r$ такі, що $r > r_0$, $k > k_0$. Тому, цілі числа r_0 і k_0 слугують границями для r і k для визначення ефективності і безпеки [2].

Перед розглядом алгоритмів ми опишемо алгоритм перевірки простого p, який називається $isStrongP(r_0, k_0, p, N)$. На вхід алгоритму подаються додатні цілі числа r_0 і k_0, де $r \geq 2^{160}$ - просте, $k \leq 4$ - ціле, просте p і цілочисельне N.

Даний алгоритм повертає просте r, якщо $N = k \times r$ є порядком криптографічно стійкої еліптичної кривої E над полем F_p, де $r > r_0$, $k > k_0$, у іншому випадку на виході буде 0. Розглянемо більш детально даний алгоритм.

Алгоритм перевірки простого p.

1) Перевіряється, чи належить N інтервалу Хассе.
2) if $|N-(p+1)|>2\sqrt{p}$ then
3) return (0);
4) $r \leftarrow 0$, $k \leftarrow 0$: // ініціалізуємо r і k, рівними 0;
5) // перевірка умови 1:
6) for $i \leftarrow 1; i \le k_0; i \leftarrow i+1 do$:
7) if $i \mid N$ and $is\,\mathrm{Pr}ime(N/i,50) = true$ and $N/i \ge r_0$ then
8) $r \leftarrow N/i; k \leftarrow i;$ break;
9) if $r = 0$ then
10) return (0);
11) //перевірка умови 2:
12) if $p = r$ then
13) return (0);
14) //перевірка умови 3:
15) $pr \leftarrow 1 \bmod r;$
16) for $i \leftarrow 1; i \le 19; i \leftarrow i+1 do$:
17) $pr \leftarrow p \times pr \bmod r;$
18) if $pr = 1$ then
19) return (0);
20) return (r).

2.3.1 Алгоритм, який ґрунтується на випадковому виборі ЕК.

Перше завдання полягає в тому, щоб знайти просте число p. На сьогоднішній день не відомі атаки на криптосистеми еліптичної кривої, які використовують властивості деякого поля F_p. Таким чином, вибір простого числа p не є критичним. Однак ми повинні розглянути граничні умови $r > r_0$, $k > k_0$.

Змінна b є довжиною біта $k_0 \times r_0$. Пропонується вибрати таке p, що $k_0 \times r_0 \le p \le 2b.$ Метод $get\,\mathrm{Pr}ime(k_0, r_0)$ повертає таке просте число. Користувач може вибрати свою власну реалізацію $get\,\mathrm{Pr}ime$, наприклад, використовувати прості числа в інтервалі $[k_0 \times r_0, 2b]$ [9].

Після того як p відоме, далі виконується наступне: виберемо параметри a і b для яких $4a^3 + 27b^2 \neq 0$, визначимо порядок групи раціональних точок кривої (a,b) над F_p і, нарешті, перевіримо, чи криптографічно сильна ця група.

Насамперед з'ясуємо, як вибрати a та b . Зазвичай вибір параметрів відбувається випадковим чином. Основна ідея полягає в тому, щоб використовувати односторонню властивість криптографічної хеш-функції. Через h позначимо таку хеш-функцію, а L - довжину в бітах на виході h. Припускаємо, що $L \geq 160$. Щоб генерувати криву випадковим чином, спочатку вибирається бітовий рядок довжиною не менше L . Записується *SEED* для цього рядка (який будує з бітового рядка нову послідовність). Як тільки *SEED* відомо, значення h(*SEED*) використовується для обчислення a та b загальновідомим детермінованим алгоритмом. Таким чином, якщо ми присвоїмо *SEED*, хеш-функцію h та детермінований алгоритм обчислення (a,b) із h(*SEED*), будь-який суб'єкт може перевірити, що a і b фактично обчислюються з використанням *SEED*. Одностороння властивість h гарантує, що параметри фактично вибрано випадково. У цій роботі ми будемо писати *getParamaters*(p,*SEED*) для будь-якого алгоритму, який повертає еліптичні криві E, визначені над F_p у випадковому порядку.

Якщо крива E = (a,b) обрана, після цього визначається порядок групи кривої $E(F_p)$. В даний час найбільш відомим алгоритмом для вирішення цього завдання є алгоритм SEA. SEA-алгоритм був розроблений американськими вченими Чуфом, Елкісом та Аткіном. Позначимо *SEA* (p, E). Позначимо результат *SEA* (p, E) через N. Якщо $isStrongP(r_0, k_0, p, N) \neq 0$, то наше завдання вирішено. В іншому випадку

нам потрібно викликати *getParamaters*(*p*,*SEED*) , *SEA* (*p*,*E*) із $isStrongP(r_0, k_0, p, N)$, поки не буде досягнуто успіху.

Алгоритм випадкового вибору ЕК

1) $p \leftarrow get\,\mathrm{Pr}ime(k_0, r_0)$;
2) while true do
3) $E \leftarrow getParamaters(p,SEED)$;
4) $N \leftarrow SEA\ (p,E)$;
5) $r \leftarrow isStrongP(r_0, k_0, p, N)$;
6) if $r \neq 0$ then
7) return (*p*,*E*,*r*,*N*/*r*).

2.3.2 Метод комплексного множення

Центральним терміном в рамках методу комплексного множення є уявний квадратичний дискримінант [1]. Позначимо такий дискримінант через Δ. Це ціле від'ємне число: $\Delta = 0,1 \bmod 4.$

Через O_Δ позначимо уявний квадратичний порядок дискримінанта Δ.

$$O_\Delta = Z\left(\frac{\Delta + \sqrt{\Delta}}{2}\right). \tag{2.15}$$

Крім того, запишемо $h(A)$ для O_Δ. Якщо p – просте число, то воно називається нормою в O_Δ, при існуванні цілих чисел t, y, то:

$$t^2 + \Delta y^2 = 4p. \tag{2.16}$$

Якщо p є нормою в O_Δ, то еліптичні криві $E_{1,p}$ і $E_{2,p}$ над полем F_p з кільцем ендоморфізмів O_Δ будуються по наступній схемі, використовується комплексне множення.

$$\left|E_{1,p}\left(F_p\right)\right| = p+1-t, \left|E_{2,p}\left(F_p\right)\right| = p+1+t. \tag{2.17}$$

Нехай $H \in Z[X]$ – мінімальний многочлен $j\left|\frac{\Delta+\sqrt{\Delta}}{2}\right|$, де j – еліптична модулярна функція. Степінь H рівна $h(\Delta)$. По модулю p многочлен H розбивається на лінійні множники. Нехай $H\left(j_p\right) = 0 \bmod p$, і припустимо, що $\Delta < 4.$ Тоді маємо: $j_p = \{0;1728\}$.

Нехай S_p – квадратичний нестаціонарний $\bmod p$. Разом з рівнянням:

$$\left(a_p, b_p\right) = \left(3k_p; 2k_p\right), \tag{2.18}$$

де $k_p = \frac{j_p}{1728-j}$.

В результаті чого отримаємо:

$$\left\{E_{1,p}\left(F_p\right), E_{2,p}\left(F_p\right)\right\} = \left\{\left(a_p, b_p\right), \left(a_p s_p^2, b_p s_p^3\right)\right\}. \tag{2.19}$$

Еліптичні криві $E_{1,p}$ і $E_{2,p}$ називаються скрученими еліптичними кривими над полем F_p [10]. Для даної конструкції не відомо, яка з еліптичних кривих $E_{1,p}$ та $E_{2,p}$ є криптографічно стійкою. Однак, вибираючи точки на кожній кривій і

перевіряючи, чи є їх порядок дільником $p+1-t$ або $p+1+t$ можна ідентифікувати криві $E_{1,p}$ та $E_{2,p}$.

Нам потрібно знати уявлення простого числа *p*, як у формулі (2.16). Тоді ми знаємо групові порядки 1, $E_{1,p}(F_p)$ і $E_{2,p}(F_p)$ із формули (2.17). Використовуючи ці порядки та алгоритм є *isStrongP* , ми можемо перевірити умови безпеки. Зазвичай більшість часу витрачається на обчислення поліному *H*. Причина в тому, що коефіцієнти *H* стають досить великими навіть для дискримінанта з невеликим значенням.

Однак, залежно від значення Δmod24, можна використовувати альтернативні багаточлени, коефіцієнти яких дуже малі в порівнянні з *H*. Робота з цими поліномами значно прискорює метод «комплексного множення» на практиці. Бітова складність методу комплексного множення інваріантна.

2.3.3 Порівняльний аналіз методів

У цьому розділі порівнюються випадковий підхід і метод комплексного множення, щоб знайти криптографічно сильну групу еліптичних кривих над *Fp* . Порівнюватимемо вплив на безпеку та ефективність.

Спочатку розберемося з безпекою. Основною перевагою випадкового підходу є те, що кожна криптографічно сильна група еліптичних кривих над *Fp* обчислюється приблизно однаковою ймовірністю. Метод комплексного множення застосовується тільки в тому випадку, якщо використовуються невеликі дискримінанти, наприклад, дискримінанти зі значеннями не більше 1000. Тоді згенеровані криві є особливими у випадку, якщо їхнє кільце ендоморфізмів має значення не більше ніж 1000. Таким чином, не кожна криптографічно сильна група еліптичних кривих може виводитися шляхом комплексного множення.

Тепер обговоримо ефективність. Розберемо нагоду при k_0=1, тобто ми шукаємо групу еліптичних кривих простого порядку. Запишемо *b* для довжини біт r_0. Раніше було заявлено, що бітова складність випадкового підходу залежить

лише від b. Однак бітова складність комплексного множення залежить від значення уявного квадратичного дискримінанта. Далі буде розглянуто, для якого значення обидва підходи мають один і той же час виконання на практиці деякого заданого фіксованого b. Це число названо значенням перетину і позначається воно $h_c(b)$.

Для визначення $h_c(b)$ спочатку виміряли час виконання випадкового підходу $P(r_0, k_0)$. Зазначено, що тут реалізовано стратегію раннього переривання та використання скрученої кривої, яка описана вище.

Всі тести були виконані на різних видах комп'ютерів та засобів обчислювальної техніки із використанням програмного забезпечення вільного доступу. Результати досліджень наведено у таблиці 2.1.

Таблиця 2.1 – Середній час виконання випадкового підходу для отримання криптографічно сильної еліптичної групи $E(F_p)$ простого порядку

B	160	170	180	190	200	210
Час виконання (хв.)	3,63	4.87	7.97	10.3	13.1	16.7
$h_c(b)$	50	820	960	1040	1090	1200

У таблиці 2.1 b означає довжину біта p. Було проведено 100 обчислювальних експериментів, виходячи з яких було визначено середньостатистичні значення.

На підставі проведених експериментів було виявлено, що метод комплексного множення буде більш швидким алгоритмом на практиці, якщо $h(\Delta) < h_c(b)$ при дискримінанті Δ, що використовується в методі комплексного множення. За даними таблиці 2.1 можна дійти невтішного висновку, що значення перетину досить велике. Таким чином, навіть якщо врахувати додаткову вимогу GISA (німецьке агентство інформаційної безпеки) про те, що значення

фундаментального дискримінанта, що відповідає Δ, становить не менше 200 – метод комплексного множення краще методу випадкового вибору ЕК по довжині біт ключів, що використовуються в криптосистемах.

Отже, застосування еліптичних кривих є однією з основних та найбільш надійних технологій побудови відкритих ключів в асиметричній криптографії. Основним критерієм стійкості таких криптографічних систем є проблема складності розв'язання дискретного логарифму.

Для протистояння наявним алгоритмам розв'язання цього завдання характеристики еліптичної кривої повинні задовольняти певним певним умовам. Відповідно, еліптичні криві, що задовольняють такі умови, є криптографічно сильними.

Важливим завданням еліптичної криптографії є створення криптографічно сильних еліптичних кривих над кінцевим полем простого порядку. Існує кілька способів генерації, серед яких найбільш поширеними та надійними є випадковий підхід до генерації еліптичної кривої та підхід, що використовує метод комплексного множення. Після генерації методами розрахунку числа точок знаходиться порядок кривої та перевіряється виконання відомих умов. На жаль, алгоритми розрахунку числа точок еліптичної кривої над великими полями надто повільні.

Якщо кільце ендоморфізмів еліптичної кривої має малу кількість класів, то метод комплексного множення краще методу випадкової генерації. Але за великих значеннях дискримінанта ($D > 200$) існуючі методи комплексного множення стають непрактичними через низьку швидкість. Тому найбільш актуальною є розробка швидких алгоритмів генерації кривих еліптичних методом комплексного множення.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНОГО АЛГОРИТМУ ШИФРУВАННЯ НА ЕЛІПТИЧНІЙ КРИВІЙ

3.1 Опис алгоритму шифрування і дешифрування на ЕК

Для початку опишемо алгоритм шифрування і дешифрування на основі використання математичного апарату еліптичних кривих. Даний алгоритм складається з 17 основних кроків. Крім того описано структуру протоколу обміну даними.

1) Усі ключі, кожен символ повідомлення, що передається і породжує елемент групи є точками еліптичної кривої з цілими координатами.

2) Нехай абонент А передає абоненту повідомлення *m*, що складається з послідовності символів. Кожен символ переводиться в число, що дорівнює *x–y*, де *x* і *y* координати деякої точки групи точок ЕК. При цьому довжина рядка символу вибирається 80 символів. Це робиться з таких міркувань. По – перше, різниця *x–y* пробігає поспіль не весь ряд цілих значень, починаючи з нуля до числа рівному розмірності алфавіту. По - друге, маючи довгий кодовий рядок ключ можна збільшити і загалом простір ключів настільки, що їх перебір навіть при відомих параметрах кривої *a, b, p* утворював елемент групи G, що стає неможливим для зловмисника. Навіть для алфавіту з 36 символів - 26 літер англійського алфавіту та 10 цифр, мінімальний кодовий рядок має число переборів $36! = 3.7*10^{41}$. Нехай суперкомп'ютер може аналізувати шифри зі швидкістю мільярд шифрів на секунду. При цьому знадобиться часу порядку $3.7 * 10^{32}$с $=10^{25}$ років. Тобто вирішення цієї задачі стає неможливим методом перебору навіть при відомих параметрах кривої еліптичної і утворює елементі групи *G*.

3) Абонент А вибирає випадкове число $k< p$. Абоненту А відомий відкритий ключ тобто точку ЕК з координатами $P_b(x_b, y_b) = n_b G$. Під шифром ЕК розуміють обидві координати 2 точок еліптичної кривої (всього 4 цілих числа для кожного символу повідомлення), побудованих за наступним правилом:

Символ алфавіту S_k переводиться до числа $m_k = x_k - y_k$, для якого знаходиться деяка точка ЕК $P_m(x_k, y_k)$, різниця координат якої $x_k - y_k$ дає це ціле число m_k зі списку символьного рядка алфавіту (точок ЕК може бути кілька із групи точок, що дозволить додатково ускладнити зашифроване повідомлення).

4) $shifr = (kG, kP_b + P_m)$.

Зазначимо, що обидві точки шифру задовольняють вимоги до конфіденційності. По-перше, крипто аналітик по першій точці не зможе визначити утворює елемент групи G, оскільки він має іншу точку kG. З іншого боку, зловмисник не зможе отримати повідомлення P_m, оскільки воно не дорівнює точці $kP_b + P_m$.

5) Тоді абонент B, знаючи свій секретний шифр n_b дешифрує даний зашифрований символ за формулою:

$$P_m = kP_b + P_m - n_b \cdot kG = kn_bG + P_m - n_b \cdot kG = P_m$$

З огляду на перестановки елементів груповий операції у циклічній абелевой групі.

Структурна схема протоколу обміну даними представлено на рисунку (3.1). Нехай два абоненти домовилися обмінюватися даними за допомогою точок еліптичної кривої з параметрами a,b та характеристикою поля p .

Абонент А, знаючи відкритий ключ В P_b, вибирає випадкове число k_A і шифрує повідомлення m_A методом еліптичної криптографії $E(m_A.k_A, a, b, p, P_b, G)$.

Повідомлення m_A і випадкове число k_A і утворює елемент групи G у відкритому вигляді не передаються, оскільки координати шифру $(kG, k_AP_A + m_A)$ їх не містять.

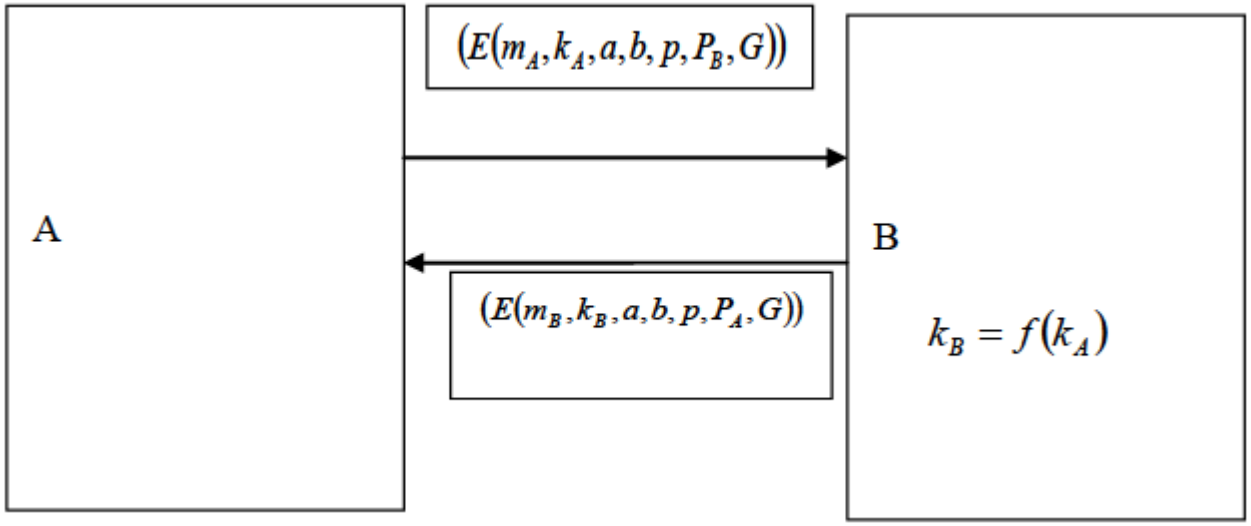


Рисунок 3.1 - Структурна схема протоколу обміну даними

Абонент В дешифрує повідомлення і від випадкового числа k_A бере хеш-функцію по формулі $k_b = f(k_A) \equiv k_{\max} \underset{k_{\max}=\max\{k_A,\overline{k_A}\},k_{\min}=\min\{k_A,\overline{k_A}\}}{\bmod} k_{\min}$, де $\overline{k_A}$ - число k_A, яке прочитано справа наліво. Дійсно, функція $f(k_A)$ є хеш-функцією, так як для, наприклад, симетричних чисел типу 11, 121,… вона рівна нулю, тобто по k_b неможливо відновити k_A і розшифрувати по шифру $E(m_A.k_A, a, b, p, P_b, G)$ повідомлення m_A, не знаючи k_A. Абонент В поступає аналогічним чином. Він хешує випадкове число $k_b = f(k_A)$ і шифрує повідомлення m_b обернено для А $E(m_B.k_B, a, b, p, P_A, G)$.

Опис роботи основної програми:

1) Для завантаження основної програми необхідно ввести параметри еліптичної кривої – *a*, *b*, *p*, відкритий ключ, що є цілою точкою еліптичної кривої (*kx*, *ky*), випадкове число *k* , шифр – текст, що складається з nn символів. Шифр – текст записується як символьна змінна масив *str*[80]. Далі, масив символів тексту порівнюється із символами алфавітного рядка, запам'ятовується масив int *kk*[int*nn*] номер позиції відповідного символу в алфавіті. 2

2) Потім, використовуючи утворюючий елемент групи $G=(res[0][0],res[1][0])$, заповнюється масив res[2][N] координатами цілих чисел еліптичної кривої, що утворюють кінцевопороджену групу за формулами (1.1) - (1.12). Число N згідно з теорією має оцінку $N \le p+2\sqrt{p}+1$, де p – характеристика поля кінцевопородженої групи точок ЕК. Усі координати точок ЕК додатні, якщо ні, то додаємо до них характеристику поля p .

Серед списку точок ЕК особливе місце займає нейтральний елемент групи з номером у списку *nol*. Усі точки списку ЕК масиву res[2][N] періодично повторюються, тобто:

$$(res[0][j],res[1][j]=res[0][nol+j],res[1][nj1+j],j=\overline{1,N-nol}).$$

3) Дальше заповнюється масив $xy[nol]=x-y$, як різниця координат абсциси та ординати точок ЕК. Усі значення масиву також мають бути невід'ємними.

4) На наступному етапі кожному елементу масиву *kk*[*nn*] зіставляється деяка точка зі списку ЕК. Якщо $kk[j]=xy[i], j=0,nn, i=0,nol-1$. У цьому заповнюється масив *pm*[2][*nn*], тобто масив точок ЕК відповідного рядку вихідного тексту.

5) Виділяємо елемент *kG* з масиву точок ЕК $(res[2][N],(res[0][k-1],res[1][k-1]))$, які служать першими двома координатами шифр - тексту для масиву $shifr[j][0],shifr[j][1], j=\overline{0,nn-1}$.

6) В масив res1[2][N] отримують координати точки *k* * *Pb* послідовним додаванням елемента *Pb* (точка *k* * *Pb* має координати $(res1[0][k0-1],k0,res[1][k-1])$.

7) Додаванням точок масиву *pm*[2][*nn*] з елементом *k* * *Pb* з масиву res1[2][N] заповнюють масив $shifr[j][2],shifr[j][3], j=\overline{0,nn-1}k*Pb+pm$. Таким чином, ми отримуємо 4 цілих числа для кожного вихідного символу – координати

2 точок еліптичної кривої $k*G, k+Pb+pm = (shifr[j][0], shifr[j][1]), (shifr[j][1], shifr[j][2])\, j = \overline{0, nn-1}$.

8) Для дешифрування повідомлення необхідно отримати точку $n_b * kG$, для чого створюємо масив $de[0][n_b], de[1][n_b]$, у якому точка $n_b * kG$ займає позицію $de[0][n_b - 1], de[1][n_b - 1]$.

9) Для дешифрування повідомлення утворюється обернений елемент до точки $n_b * kG$ з координатами $-n_b * kG = de[0][n_b - 1], -de[1][n_b - 1]$.

10) Дешифрування є додавання 2 точок ЕК $k \cdot Pb + pm$ - останні дві координати масиву $shifr[j][2], shifr[j][3], j = \overline{0, nn-1}$ з точкою $-n_b * kG$ з координатами $-n_b * kG = de[0][n_b - 1], -de[1][n_b - 1]$, отриманих по 2 перших координатах масиву $(shifr[j][0], shifr[j][1])$.

$k \cdot Pb + pm - n_b \cdot G = k \cdot n_b \cdot G + pm - n_b \cdot G = pm$, або використовуючи підпрограми основної програми:

$$(x_{pm}, y_{pm}) = \begin{pmatrix} xp(de[0][n_b - 1], -de[1][n_b - 1], shifr[j][2], shifr[j][3], a, p) \\ yp(de[0][n_b - 1], -de[1][n_b - 1], shifr[j][2], shifr[j][3], a, p) \end{pmatrix}, j = \overline{0, nn-1}.$$

11) Координати дешифрування записуються у масив $(otv[0][j], otv[1][j]) = (x_{pm}, y_{pm}), j = \overline{0, nn-1}$.

12) Утворюємо масив даних $des[j] = x_{pm} - y_{pm}, j = \overline{0, nn-1}$.

13) Порівнюємо значення масиву з номерами відповідних символів алфавітної стрічки і виводимо символи розшифрованого тексту

```
for(j = 0; j <= nn - 1; j++)
{
jj = des[j];
printf("%c \ n", str[jj]);
 }
```

14) Основна програма утворює два текстові файли balka1.txt і balka2.txt . Bbalka2.txt записується випадкове число *k*. Запис випадкового числа потрібний

тільки для повноти протоколу адміністратора. При дешифруванні знання окремо взятого числа *k* не потрібно. У текстовому файлі balka1.txt міститься сам шифр. Якщо змінити кілька цифр у рядках файлу, то при дешифруванні будуть неправильно розпізнані ті символи, яким відповідають рядки в записі balka1.txt.

15) У процесі шифрування та дешифрування тексту методом ЕК наприкінці операції складається протокол шифрування та дешифрування та правильність проведення операції по кожному окремому символу вихідного тексту за допомогою допоміжної функції int prov (int x1, int y1, int a, int b, int p), тобто, перевіряється належність точки з координатами (x_1, y_1) еліптичної кривої з параметрами *a,b* в кінцевому полі цілих чисел з характеристикою p .

16) У процесі шифрування і дешифрування даних необхідно використовувати одні й самі параметри *a*, *b*, *p*, *k*, (*kx*,*ky*), інакше, процес дешифрування не здійснюється правильно. Цю програму можна покласти в основу створення еліптичного генератора кривих. Код основної програми міститься у додатку.

17) Крім того, у програмному пакеті міститься спрощена основна програма, яка виконує лише функцію дешифрування за вихідними текстовими файлами. Вона зчитує текст з $(shifr[j][0], shifr[j][1]), (shifr[j][1], shifr[j][2])\, j = \overline{0, nn-1}$ з файлу balka1.txt і повертає по цим данним вихідний символьний текст *m* довжиною *nn* символів.

3.1.1 Опис роботи допоміжних програм

1) *int nu*(*int t*) - визначає число розрядів цілого числа в десятковій системі (відображає ціле число в ціле число $int \rightarrow nu(int\ t)$).

2) *int xyz*(*char cx*)- викликає символьний рядок з текстового файлу і перетворює посимвольно їх у цифру десяткової системи числення, інакше, якщо перетворений символ не цифра, то кінець роботи з цим символьним рядком ($char\ cx \rightarrow int(char\ cx)$).

3)*char zyx*(*int m, int nol*)- *m, nol* – вхідні цілочисельні змінні програми, друге – номер нейтрального елемента звичайнопородженої групи ЕК. Відкривається файл *balka1,txt*. Дана підпрограма заповнює файл *balka1, txt* шифр - текстом, послідовно приймаючи шифр *m* з основної програми (кожному символу вихідного тексту відповідає 4 числа - координати та 4 рядки у файлі *balka1, txt*).

4) *int xp*(*int x1, int y1, int x2, int y2, int a, int p*) - за відомими 2 цілими точками, що лежать на ЕК (x_1, y_1), (x_2, y_2), параметрам ЕК *a, b, p* отримує x_3 - абсцису третьої точки, що належить ЕК, у довільному випадку. Якщо точки (x_1, y_1), (x_2, y_2) різні, то використовуються формули додавання. Якщо точки(x_1, y_1), (x_2, y_2) збігаються і $y_1 = y_2$ відмінні від нуля, то використовуються формули подвоєння. Якщо $x_1 = x_2, y_1 = -y_2 \Leftrightarrow y_1 + y_2 \equiv 0 \bmod p \Leftrightarrow y_1 + y_2 \equiv p \bmod p$, через вихідні точки проходить вертикальна пряма, і наступна точка (x_3, y_3) є нескінченно віддаленою, або нейтральним елементом скінченнопородженої групи.

5) *int yp* (*int x1, int y1, int x2, int y2, int a, int p*) - аналогічно функції *int xp*(*int x1, int y1, int x2, int y2, int a, int p*) утворює ординату третьої точки ЕК по 2 відомим точкам ЕК у довільному випадку.

6) *int prov* (*int x1, int y1, int a, int b, int p*) - дозволяє перевірити чи належить точка з координатами (x_1, y_1) ЕК з параметрами *a, b, p*. Якщо точка належить кривою, то функція *prov*(*x1, y1, a, b, p*), запущена з основної програми, повертає число 0. В іншому випадку *prov*(*x1, y1, a, b, p*) повертає інше число.

Число *nol* визначає по всьому масиві цілочисельні точки ЕК $nol : \underbrace{G + G + \ldots + G}_{nol} = O$. Код допоміжних програм представлений у додатках.

3.2 Шифрування, дешифрування і тестування програмного засобу

Для тестування шифрування даних введемо фразу з латинським шрифтом Полоцький державний університет 2016 (polotsk state university 2016) з довжиною

рядка *nn* = 29. Результат введення тексту, параметри еліптичної кривої *a*=-1, *b*=188, *p* =751, відкритий текст (*k x*, *k y*) =(201,5) наведені на рисунку 3.1.

На рисунку 3.2 бачимо результат шифрування чорним кольором. Кожному вихідному символу тексту відповідає чотири координати 2 точок еліптичної кривої, що розташовані в одному рядку. Для зручності введення та рядкового зчитування той же шифр в один стовпець записується в текстовий файл balka1.txt. Ми бачимо повний збіг шифру двох етапах.

Крім того, видно також, що випадкове число k =17 (рисунок 3.1), що вводиться в програму, і записане програмою в текстовий файл balka2.txt також збігаються.

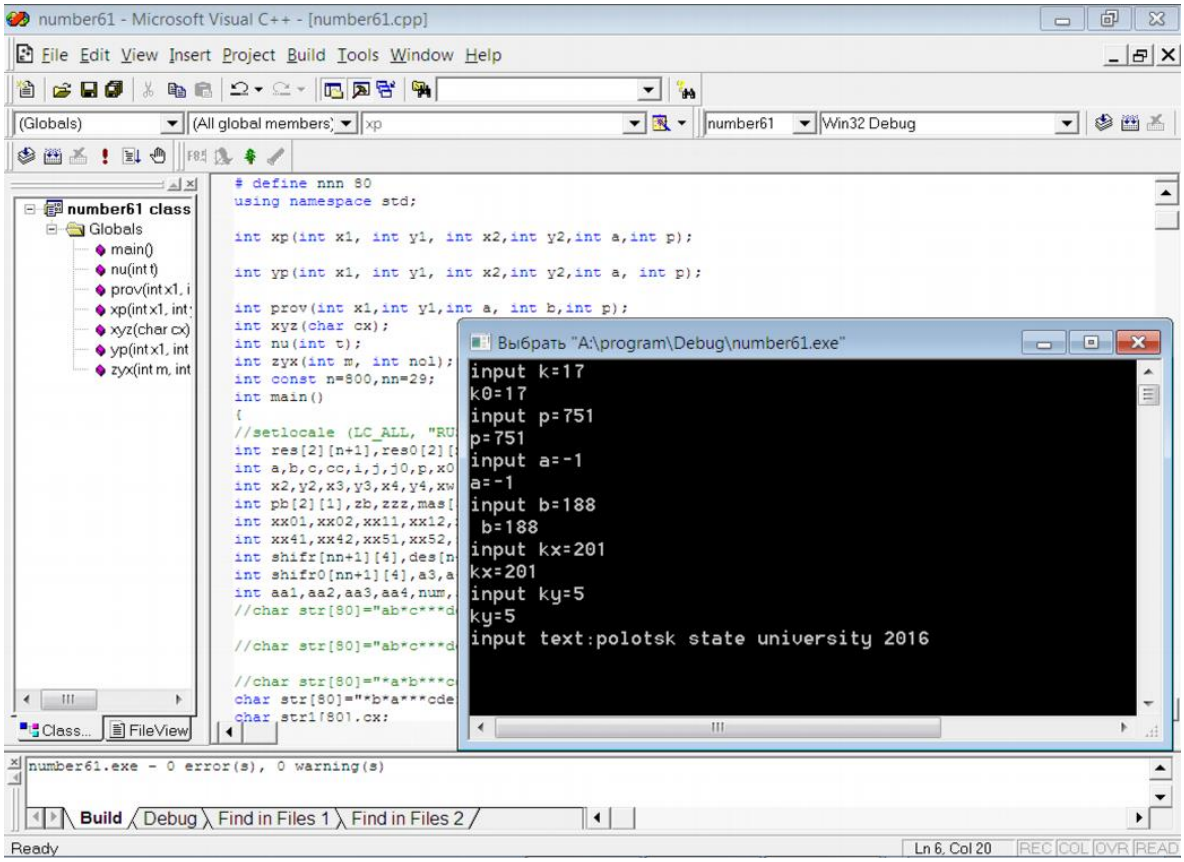


Рисунок 3.1 – Ввід даних в програму

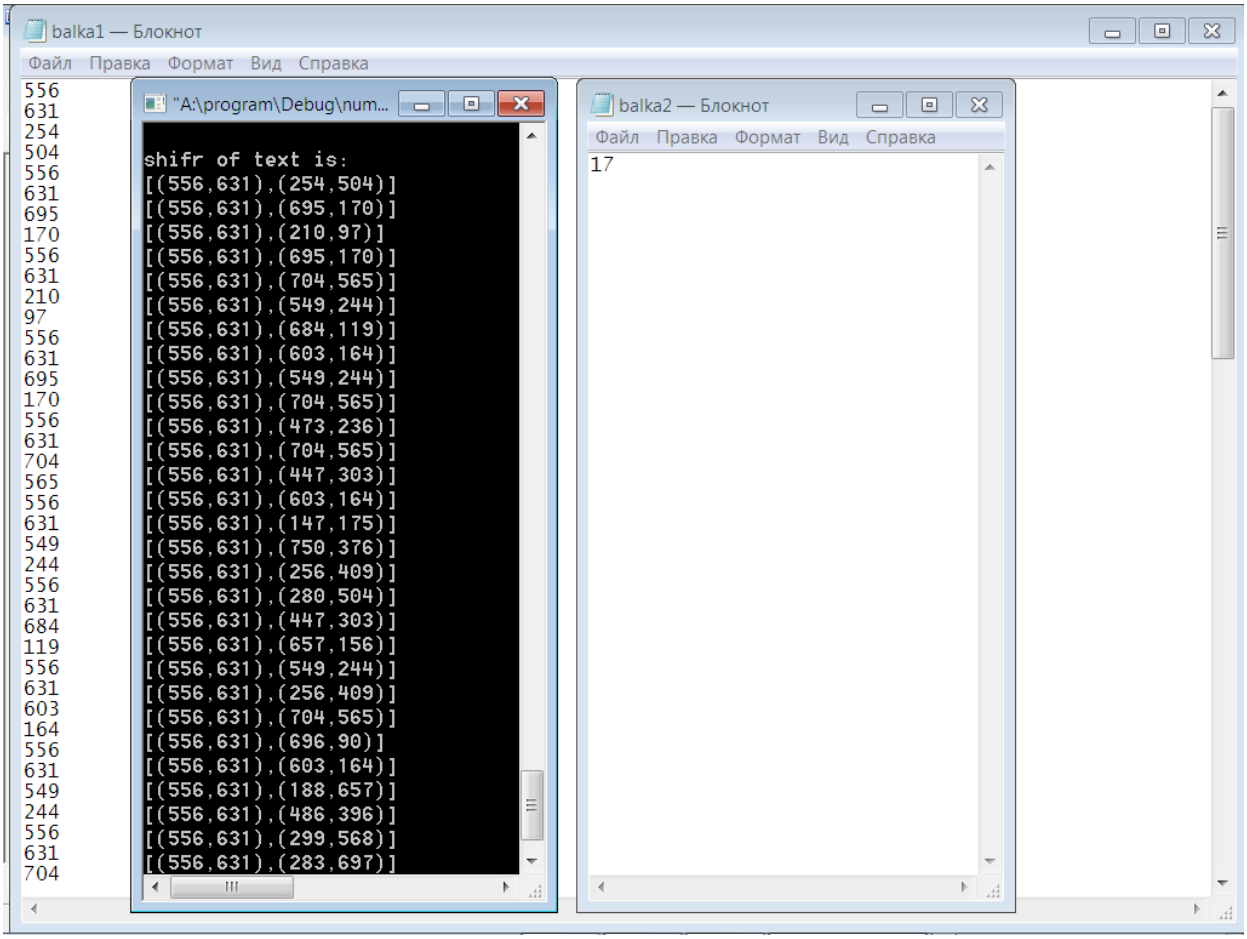


Рисунок 3.2 – Порівняння програмного шифру і шифру текстового файлу.

У ході роботи програми складається протокол шифрування по кожному вхідному символу, що шифрується. Функція перевірки *prov*(*x1*, *y1*, *a*, *b*, *p*) здійснює приналежність кожної точки, що перевіряється з координатами (x_1, y_1) еліптичної кривої з параметрами *a*, *b*, *p* . Якщо точка належить ЕК, то програма перевірки повертає число 0, інакше повертається інше ціле число. У протоколі шифрування вказуються координати точки ЕК, що відповідають кожному вхідному символу. Різниця координат точки *x* - *y* дорівнює позиції вихідного символу в алфавітному стоку.

Наприклад, по точці еліптичної кривої з координатами (*xm*, *ym*) ми можемо визначити (*xm*, *ym*) = (680,657), *des*=(*xm* – *xy*) =(680– 657)= 23 – вихідний символ,

використовуючи алфавітний рядок (нумерація символів в алфавітному рядку починається з нуля, тому *des*=23 відповідає 24 символу, тобто латинській букві p).

char str[80]="**b*a ***cdefghi * jkl **mnopqrs**tuvwxyz*01*2**3456789** ";

Дійсно, фраза (polotsk state university 2016) починається з букви p. На рисунку 3.3 представлено протокол шифрування на основі ЕК.

Другий символ (xm, ym)= (266,244), des = xm – xy = 266- 244 = 22 відповідає 23 за рахунком символу в алфавітному рядку, тобто латинській літері o, що відповідає другій літері у слові poltsk. Ми з рисунка 3.3 бачимо, що ці точки тексту перетворюють функцію перевірки у нуль, тобто, всі точки є точками еліптичної кривої.

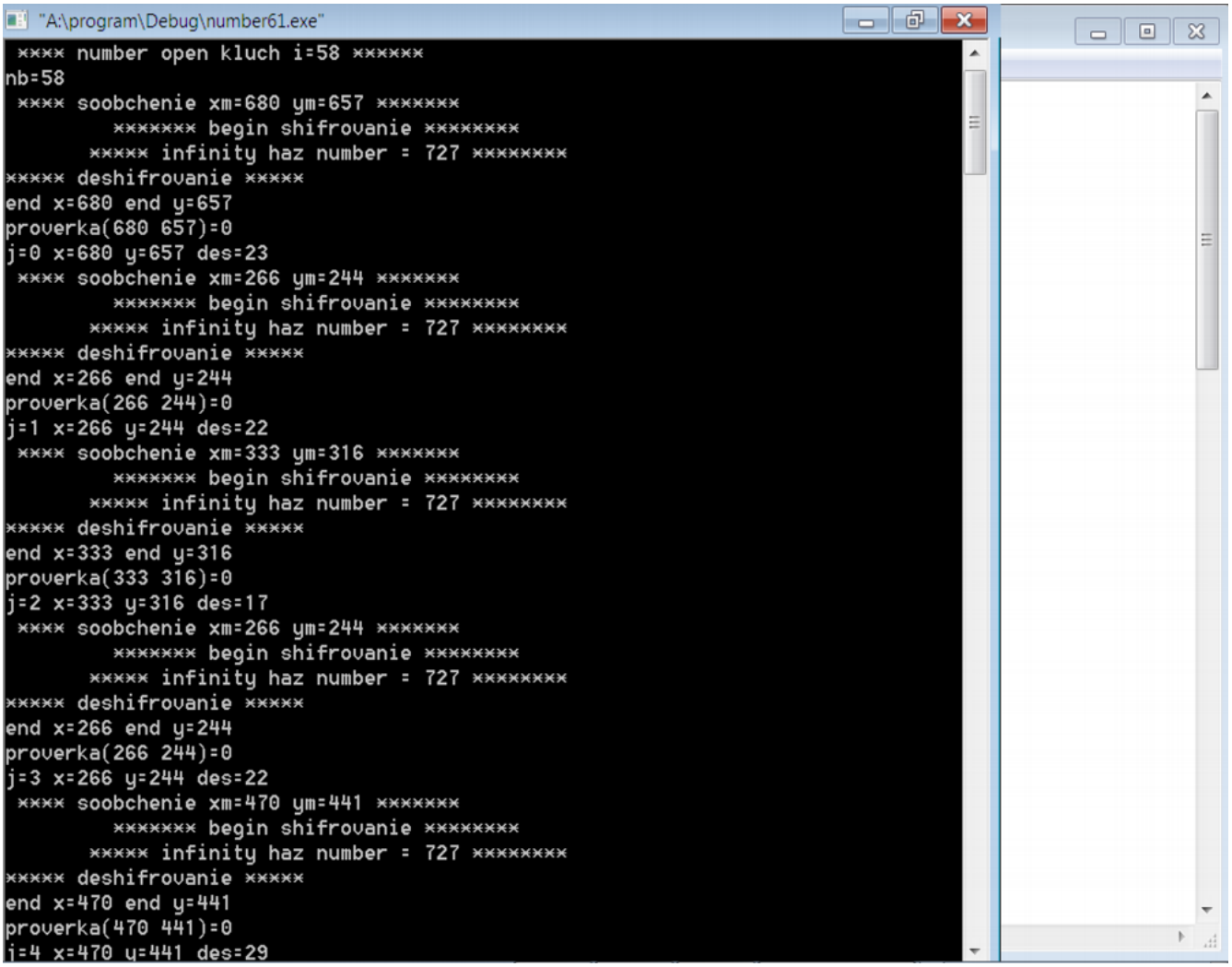


Рисунок 3.3 – Протокол шифрування на основі математичного апарату ЕК

3.2.1 Приклади дешифрування даних

Дешифрування даних можливе основною програмою, яка зчитує шифр із текстового файлу balka1, txt. Дешифрування здійснюється за формулами пунктів 10)-11 та алгоритму12)-13). Ми бачимо, що вихідна фраза рисунку 3.1 та кінцева рисунку 3.4 (polotsk state university 2016) повністю збігаються з огляду на прогалини між словами.

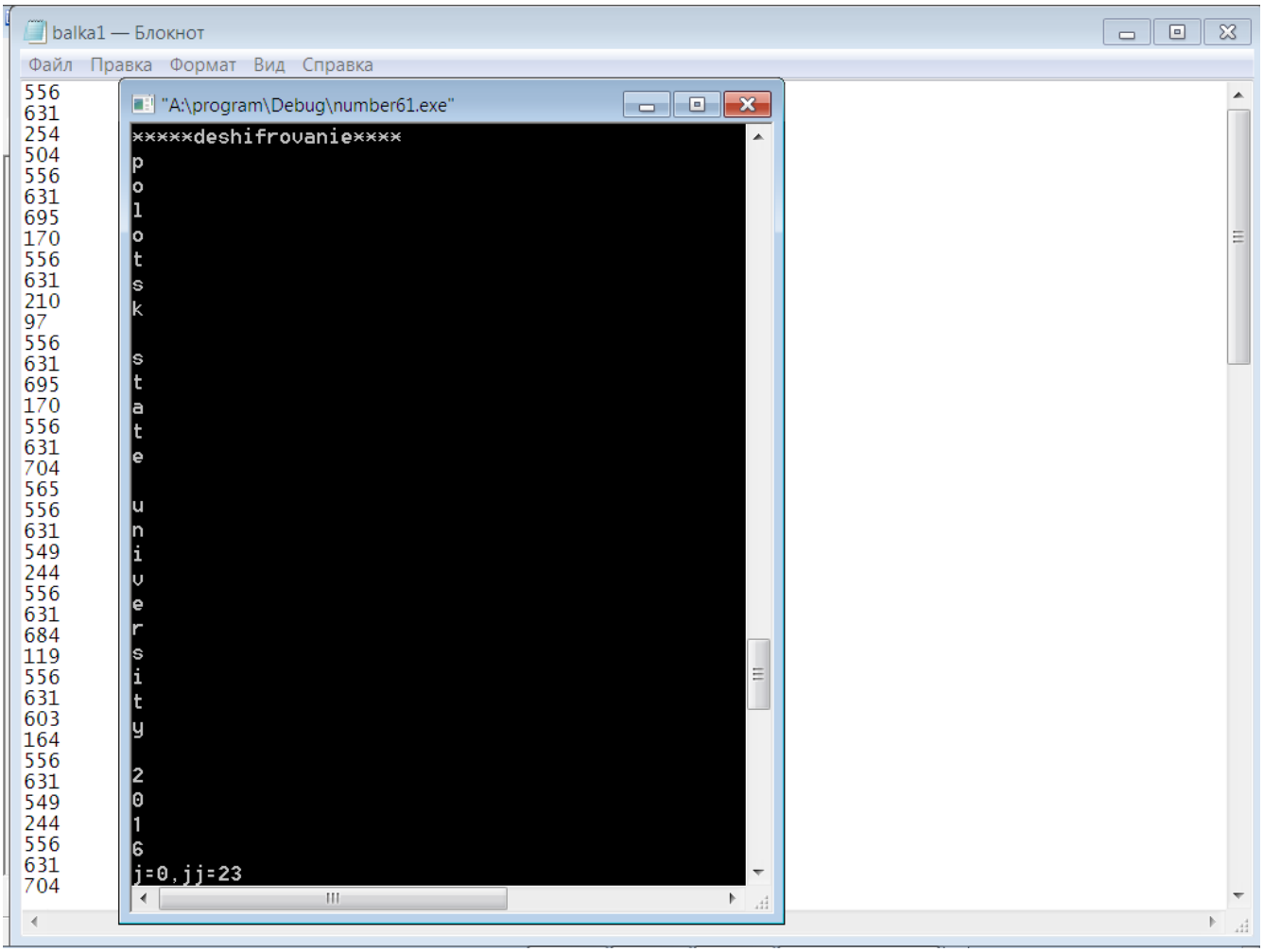


Рисунок 3.4 – Дешифрування даних на основі запропонованого алгоритму

Дешифрування вихідних даних можливе іншою незалежною програмою, призначеною лише для роботи з текстовим файлом balka1, txt. Функція дешифрування використовує тільки параметри кривої a = -1, b =188, p = 751,

відкритий ключ (*kx*,*ky*) = (201,5) та шифр – текст із файлу balka1,txt . На рисунку 3.5 представлено вікно програмного засобу для виведення даних при дешифруванні.

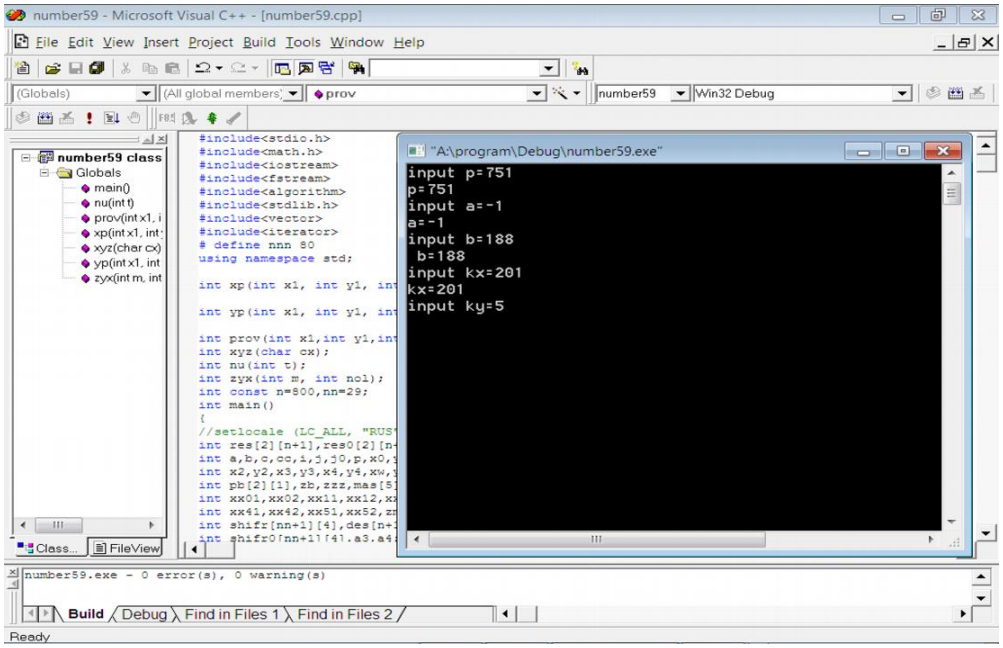


Рисунок 3.5 – Вивід даних при дешифруванні

А на рисунку 3.6 представлено вікно з дешифрованими даними на основі використання математичного апарату ЕК при заданих вхідних параметрах.

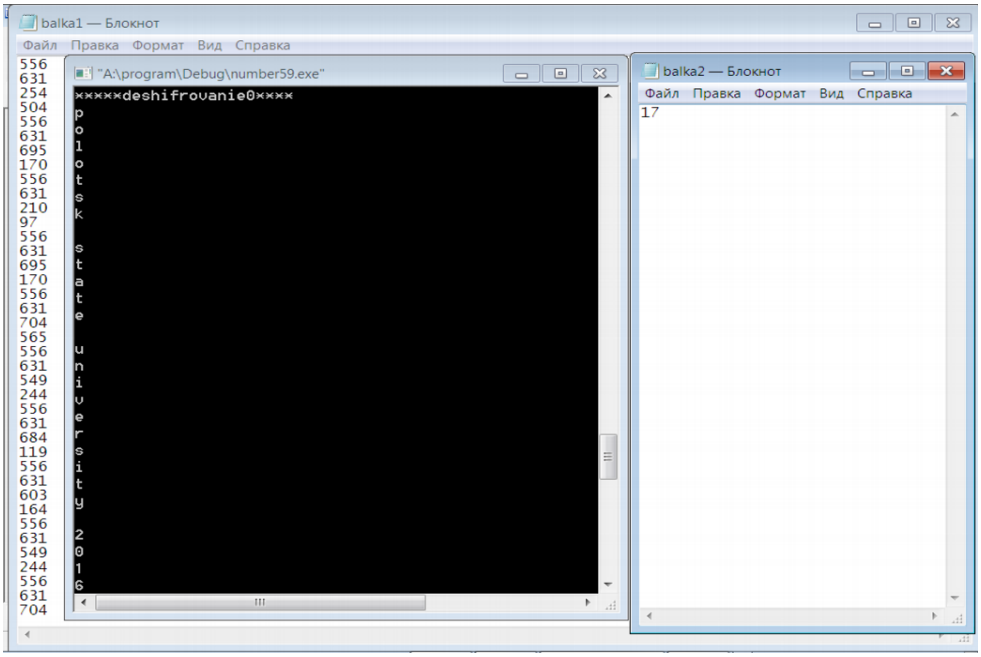


Рисунок 3.6 – Дешифрований текст

Ми бачимо, що при дешифруванні обома програмами із записаного в текстовому файлі balka1,txt шифру еліптичної кривої, виходить вихідний текст мал.1 - polotsk state university 2016 .

3.2.2 Інтерфейс користувача

Як інтерфейс користувача програми введення даних можна розглянути, наприклад, наступний інтерфейс із введенням довжини повідомлення в символах *nn* , параметри еліптичної кривої *a, b, p* , відкритий ключ *kx, ky*, випадкове число *k*.

Інтерфейс включає наступні вікна – вікна введення чисел *a, b, p*, вікна введення відкритого ключа OpenKey X, OpenKey Y, вікно введення випадкового числа Random numberk.

Функціональні елементи запуску, очищення та зупинки інтерфейсу виходу з програми Run, Clear, Stop, Out.

Вікно введення первинної інформації Soursetext, куди вводиться вихідний текст за допомогою клавіатури та встановлення курсору у вікно введення.

Вікно шифрованого тексту безпосередньо перед записом шифру текстового файлу 1.txt Writing text.

Довжина запису вихідного тексту у вікні не є активною, тобто. у це вікно нічого не заноситься. Довжина рядка тексту nn, що вводиться, автоматично змінюється в даному вікні при видаленні або додаванні кожного нового символу.

На рисунках 3.7 і 3.8 представлено інтерфейси користувача шифрування та дешифрування даних відповідно.

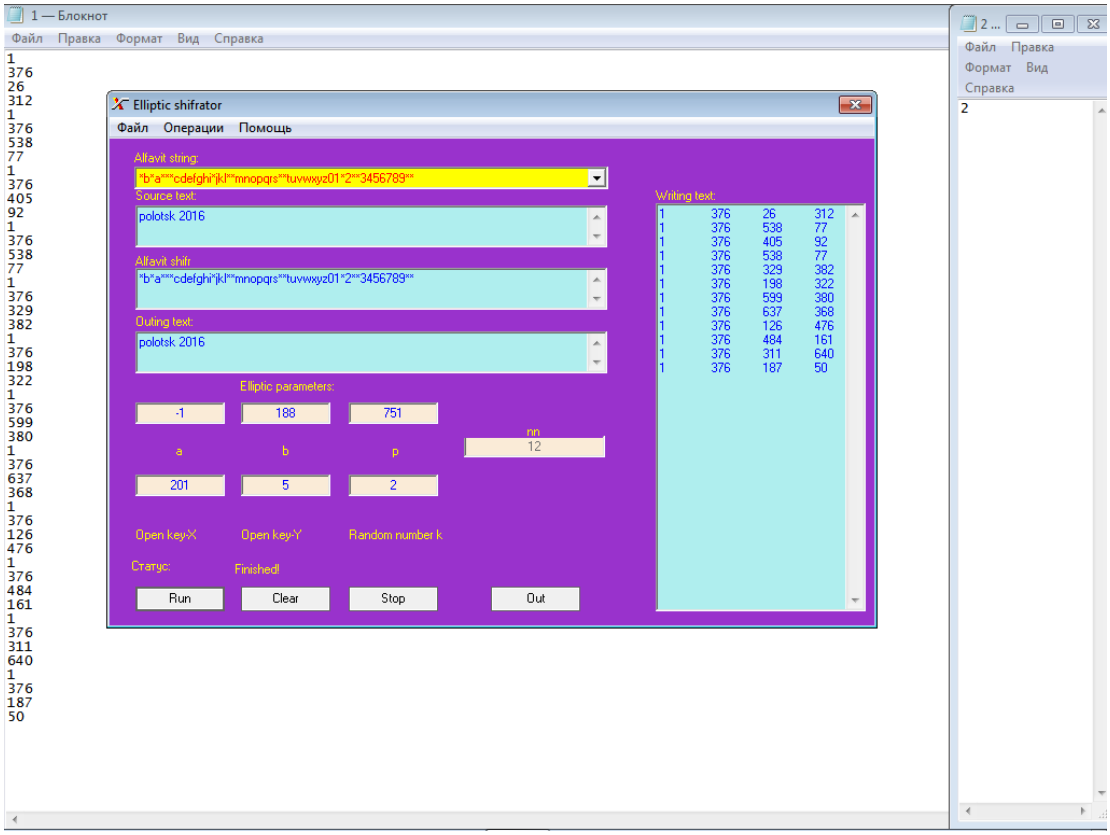


Рисунок 3.7 – Шифрування даних інтерфейсом користувача

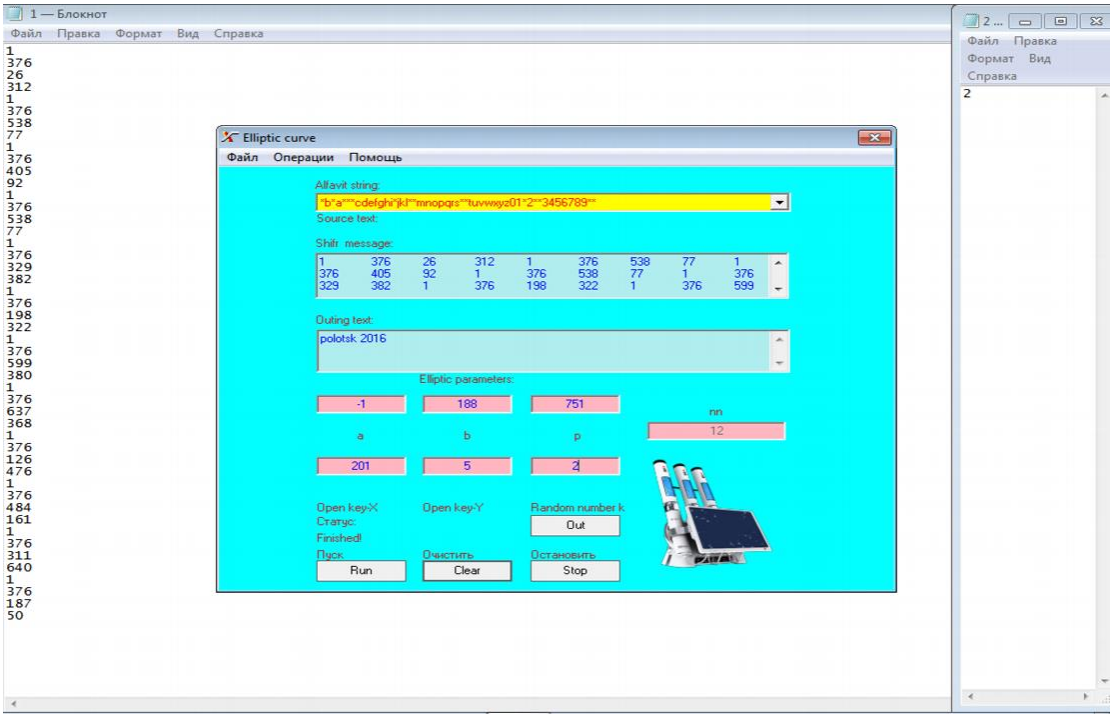


Рисунок 3.8 – Дешифрування інтерфейсом користувача вхідного тексту.

Вікно дешифрованого тексту Outing text виводить текст, що символічно збігається з вихідним текстом.

Цей інтерфейс не тільки шифрує вихідний текст, але і є програмою – контролем на всіх етапах шифрування та дешифрування та створення текстового файлу для шифру.

Інтерфейс містить також функціональні вікна з падаючим списком опцій, за допомогою яких також можна шифрувати вихідний текст (наприклад, операції – кодувати, закрити; файл – переглянути; допомога – про програму).

Порівнюючи рисунок 3.7 і рисунок 3.8 бачимо, що з запуску інтерфейсу паралельно відбувається як шифрування вихідного тексту, і створення текстового файлу для запису шифру 1.txt. Довжина вхідної фрази символи дорівнює 12 символів. На етапах шифрування (вертикальне вікно інтерфейсу) записи шифру в текстовий файл (лівий стовпець на рис.8 переданого з файлу 1.txt) повністю збігаються. Збігаються також символи тексти вихідний (у вікні *Soursetext*) і дешифрований текст (у вікні *Outing text*).

Крім того, інтерфейс програми містить пасивний логічний оператор Статус, який у разі успішного запуску повертає стан *Finished* . У разі помилки даний логічний оператор вказує стан (рід помилки, що відбулася).

Праворуч на рис.8 вказано випадкове число k=2, записане програмою текстовий файл 2.txt. Воно збігається з числом, введеним в інтерфейс вікна *Random numberk*. У додатку D містить модуль інтерфейсу програми.

Зазначимо, що кодові алфавітні рядки на рисунку 3.7, рисунку 3.8 при шифруванні та дешифруванні тексту повинні збігатися.

*char str[80]="*b*a***cdefghi*jkl**mnopqrs**tuvwxyz01*2**3456789** ";*

Програми шифратора та дешифратора можна оформити у вигляді окремих бібліотек dll. Такі бібліотеки легко підключити до інтерфейсів, написаних на будь-яких об'єктно-орієнтованих мовах, наприклад, Compaq Visual Fortan, Visual C++ та інших.

3.3 Інтерфейс адміністратора

Інтерфейс адміністратора на відміну інтерфейсу користувача має координати утворюючого елемента звичайно породженої групи Gx, Gy.

Це зроблено для того, щоб адміністратору було простіше тестувати алфавітні рядки – ключі під еліптичну криву. За замовчуванням вибрано параметри кривої:

a=0,b=-4,p=211,kx=115,ky=48,Gx=Dy=2,k=17,Alfavit string:
*str[80]="ab*c***d**ef*gh*i*j*kl*m**n**o**pqrs*t**u**v*wxyz012****3**
4*56*789*** ";*

Шифруємо фразу:*polotsk 2016 june 30*.

На рисунках 3.9 та 3.10 представлено шифрування та дешифрування інтерфейсом адміністратора вхідного тексту відповідно.

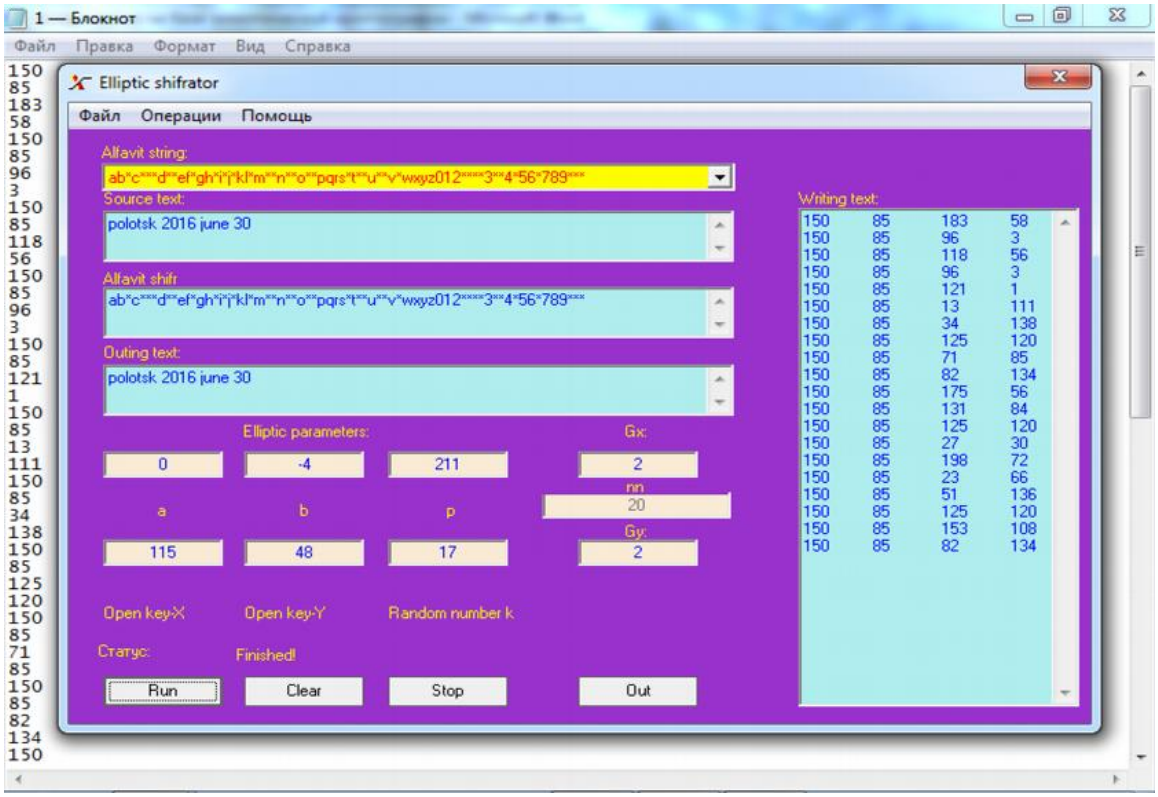


Рисунок 3.9 – Шифрування інтерфейсом адміністратора вхідного тексту

Як видно з рисунку 3.9 контрольний рядок auting text збігається з рядком вихідного тексту sourse text, а шифр тексту в рядку writing text збігається з шифром, записаним у текстовий файл 1.txt.

Програма дешифратора за замовчуванням налаштована на еліптичну криву з параметрами:

a=0,b=-4,p=211,kx=115,ky=48,Gx=Dy=2,k=17,Alfavit string:
*str[80]="ab*c***d**ef*gh*i*j*kl*m**n**o**pqrs*t**u**v*wxyz012****3**4*56*789*** ";*

Для дешифрування передаємо файл 1.txt, програму дешифратора (шлях 2.26 - primer - debug) і запускаємо програму дешифратора.

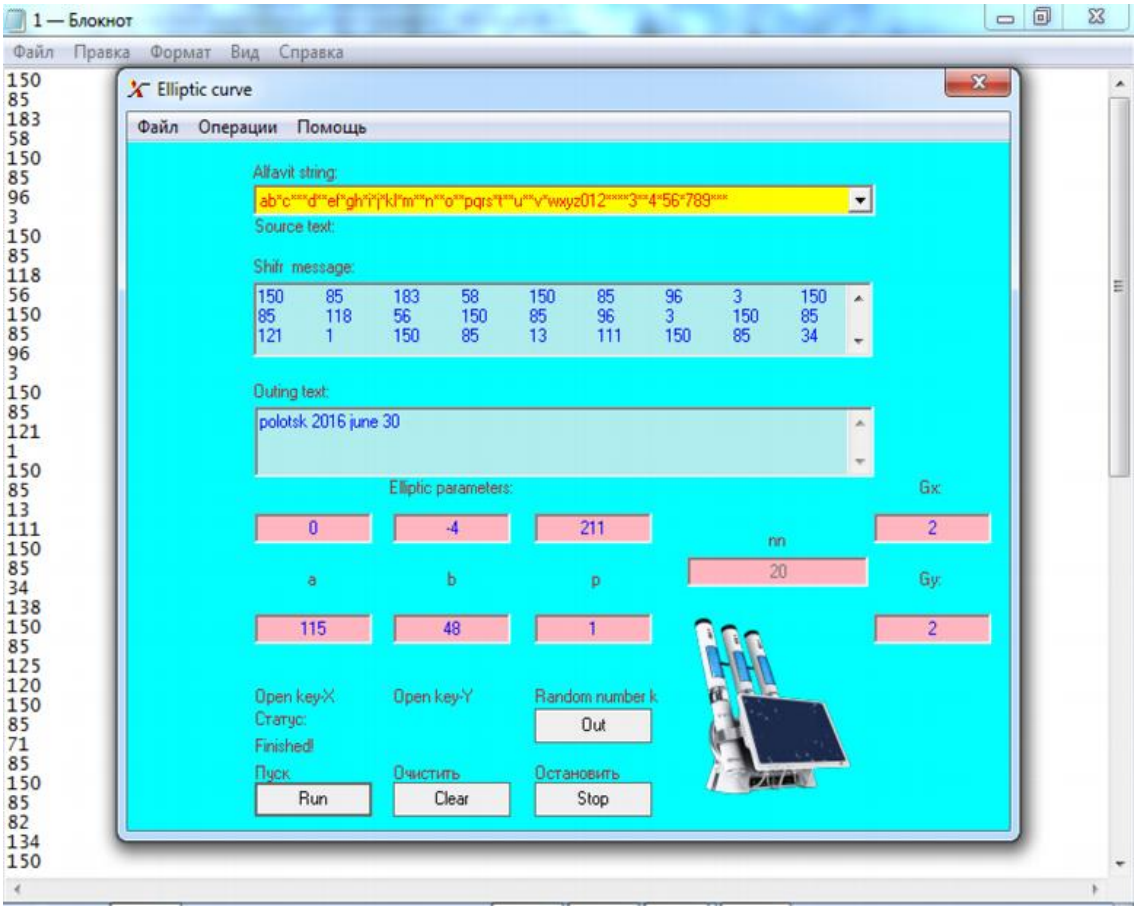


Рисунок 3.10 – Дешифрування інтерфейсом адміністратора вихідного тексту.

З рисунка 3.10 видно, що шифр – текст, прочитаний із файла 1.txt збігається з шифром прочитаною програмою дешифратора і введеним у вікно *shifr message*.

Дешифроване повідомлення з тими ж параметрами еліптичної кривої, що й у шифратора тотожно дорівнює вихідному повідомленню polotsk 2016 30.

Розглянемо тепер шифрування фрази polotsk 2016 june 30 на базі еліптичної кривої з параметрами:

*a=-1,b=188,p=751,kx=201,ky=5,Gx=0,Gy=376,k=11,Alfavit string: char str[80]="*b*a***cdefghi*jkl**mnopqrs**tuvwxyz01*2**3456789** ";*

Як видно з рисунку 3.11, контрольний рядок *auting text* збігається з рядком вихідного тексту *sourse text*. Шифр тексту в рядку *writing text* збігається із шифром, записаним у текстовий файл 1.txt. У той же час шифр із файлу 1.txt рисунку 3.11 відрізняється від шифру з файлу 1.txt рисунку 3.9.

Дешифруємо текст, користуючись програмою дешифратора, згенерованими файлами 1.txt і параметрами еліптичної кривої:

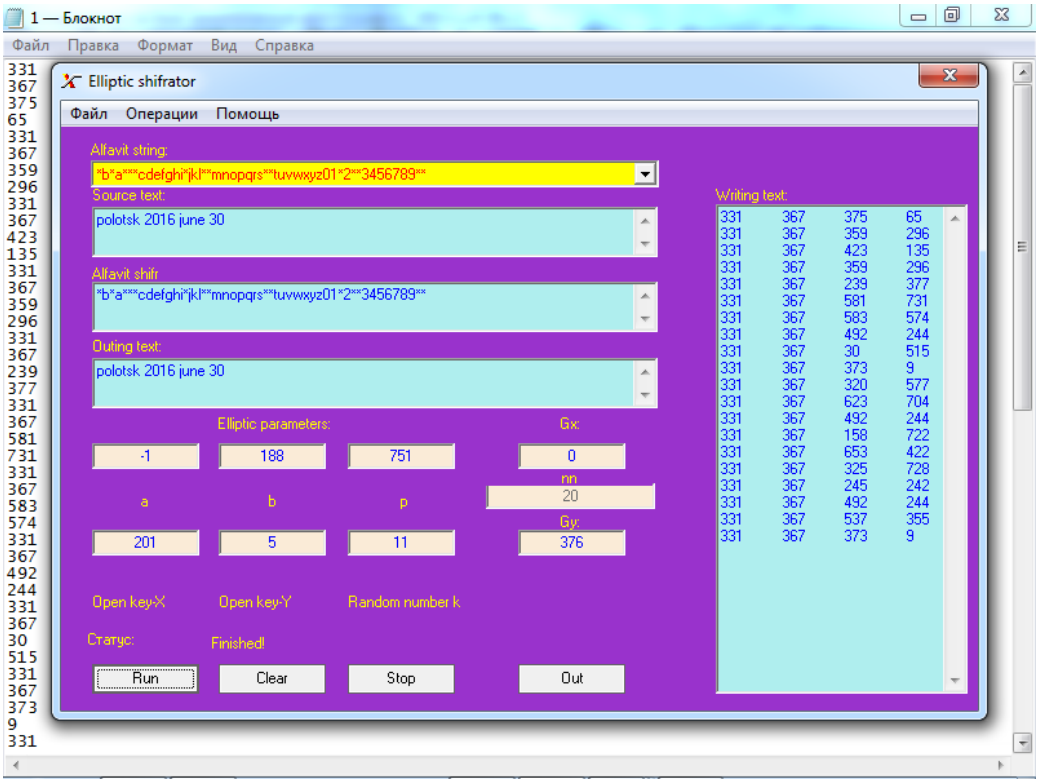


Рисунок 3.11 – Шифрування інтерфейсом адміністратора вхідного тексту

Дешифруємо текст, користуючись програмою дешифратора, згенерованими файлами 1.txt і параметрами кривої еліптичної:

*a=-1,b=188,p=751,kx=201,ky=5,Gx=0,Dy=376,k=11,Alfavit string: char str[80]="*b*a***cdefghi*jkl**mnopqrs**tuvwxyz01*2**3456789** ";*

На рисунку 3.12 видно, що шифр - текст, прочитаний з файлу 1.txt, збігається з шифром, прочитаним програмою дешифратора і введеним у вікно *shifr message*. Дешифроване повідомлення з тими ж параметрами еліптичної кривої, що й у шифратора тотожно дорівнює вихідному повідомленню polotsk 2016 30 (рисунок 3.11).

Таким чином, інтерфейси шифратора та дешифратора для адміністратора дозволяють синтезувати алфавітні рядки – ключі як при фіксованих параметрах еліптичної кривої з фіксованою довжиною алфавітного рядка, так і отримання нових алфавітних рядків – ключів різної довжини методом використання різних еліптичних кривих.

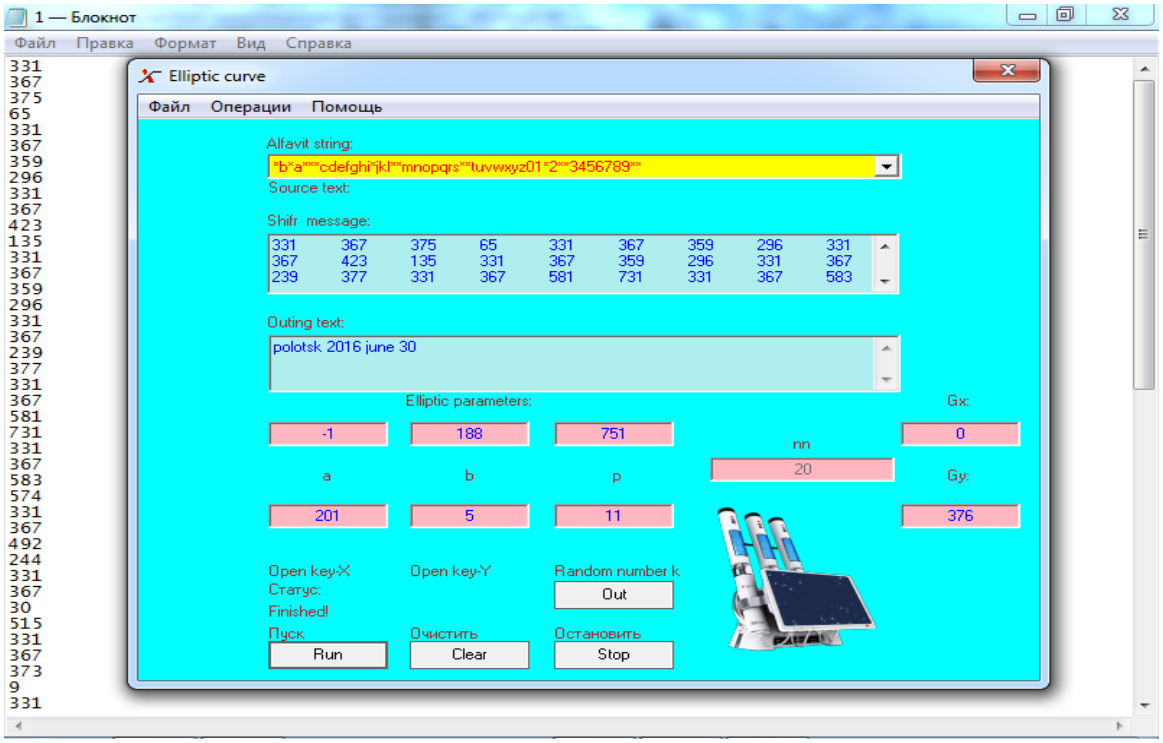


Рисунок 3.12 – Дешифрування інтерфейсом адміністратора вхідного тексту

Інтерфейси адміністратора включають меню з використанням гарячих клавіш (декодувати (alt + ctrl + D), вихід (alt + C)), операції (очистити (alt + o), зупинити (alt + s)), допомогу (про програму (alt +a)).

В результаті чого запускати програму та деякі її функції можна як із меню, так і за допомогою гарячих клавіш.

ВИСНОВКИ

Кваліфікаційна робота присвячена актуальній науковій задачі забезпечення захисту інформаційних потоків за умови застосування математичного апарату еліптичних кривих. Для використання ЕК необхідною умовою стійкості криптосистеми є вибір параметрів, які задовольняють певним умовам. Тому у роботі проведені дослідження та вирішено наступні задачі.

1. Проведено аналіз сучасного рівня захисту інформації в комп’ютерних системах (КС) та місце в ньому криптографії ЕК.

2. Проведено дослідження оцінки ефективності функціонування алгоритмів шифрування на еліптичних кривих та їх стійкість до атак.

3. Розроблено ефективні алгоритми генерування параметрів ЕК на основі генетичного алгоритму стійких до різного виду атак.

4. Розроблено швидкодіючі алгоритми шифрування за умови застосування еліптичних кривих.

5. Розроблено спеціалізовані програмні засоби реалізації запропонованих алгоритмів та досліджено їх роботу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Shashi Mehrotra Seth, Rajan Mishra, Comparative Analysis Of Encryption Algorithms For Data Communication, IJCST Vol. 2, Issue 2, June 2011
2. Evaluation Of Performance Characteristics Of Cryptosystem Using Text Files
3. R.L.Rivest, A.Shamir, L.Adleman “A method for obtaining digital signatures and Public-Key Cryptosystems”, Communications of the ACM 21 (1978), 120-126.
4. Myungsun Kim, Jihye Kim, And Jung Hee Cheon, Compress Multiple Ciphertexts Using Elgamal Encryption Schemes
5. Narasimham Challa and Jayaram Pradhan, Performance Analysis of Public key Cryptographic Systems RSA and NTRU, IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.8, August 2007
6. Mohit Mittal, Performance Evaluation of Cryptographic Algorithms, International Journal of Computer Applications (0975 – 8887) Volume 41– No.7, March 2012
7. Diaa Salama Abd Elminaam, Hatem Mohamed Abdual Kader, and Mohiy Mohamed Hadhoud, Evaluating The Performance of Symmetric Encryption Algorithms, International Journal of Network Security, Vol.10, No.3, PP.216–222, May 2010
8. Diaa Salama Abdul. Elminaam, Hatem M. Abdul Kader and Mohie M. Hadhoud, Performance Evaluation of Symmetric Encryption Algorithms on Power Consumption for Wireless Devices, International Journal of Computer Theory and Engineering, Vol. 1, No. 4, October, 2009 1793-8201
9. P.R.Vijayalakshmi, K. Bommanna Raja, Performance Analysis of RSA and ECC in Identity-Based Authenticated New Multiparty Key Agreement Protocol, International Conference on Computing, Communication and Applications (ICCCA), 22-24 Feb. 2012, pp 1-5

10. Березин Б.В., Дорошкевич П.В. Цифровая подпись на основе традиционной криптографии: вып. 2.- М.:МП Ирбис – II,1992.-202 с.

11. Бутакова Н.Г., Семененко В.А., Федоров Н.В.Криптографическая защита информации: учебное пособие для вузов.- М.:Изд-во МГИУ, 2011. –т 316 с.

12. Жданов О.Н., Чалкин В.А. Эллиптические кривые: Основы теории и криптографические приложения.- М.: Книжный дом ЛИБРИКОМ, 2013.- 200с.

13. Neal Koblitz. Random Curves: Journeys of a Mathematician, - Spinger, 2009.- C. 312 -313. – 402 с. – ISBN 9783540740780.

14. Koblitz N.A. Course in namber theory and cryptography. – USA: Spinger – Verlag.–1994.–235с.

15. Ишмухаметов Ш.Т. Методы факторизации натуральных чисел.- Казань:КФУ,2011.-1990 с. 13. ГОСТ Р 34.10- 2012.

16.

ДОДАТОК А
Допоміжні програми

```
int nu(int t)
{
int i,t1,sss; t1=t; sss=0; for(i=0;i<=100;i++)
{
if(t1>=10)
{
t1=(t1-t1%10)/10; sss=sss+1;
}
else if(t1<10)
{
t1=t1; sss=sss+1; i=100;
}
}
return sss;
}
//////////////
int zyx(int m, int nol)
{
ofstream out("balka1.txt" , ios::app); out<< m ; out<<endl; printf("\n"); return 1.0;
}
////////////////
int xyz(char cx)
{
int k,i,j,nx; char str0[10]={'0','1','2','3','4','5','6','7','8','9'}; for(k=0;k<=9;k++)
{
if(cx==str0[k] )
{
nx=k;
}
}
return nx;
}
int xp(int x1,int y1,int x2,int y2,int a, int p)
{
int s,s1,s2,s3,s4,ss,w,i; if(!(x2==x1))
{
s=(y2-y1)%p; ss=(x2-x1)%p; if(s<0)
{
s=s+p;
```

```
}
if(ss<0)
{
ss=ss+p;
}
if(!ss==0)
{
for(i=1;i<=p-1;i++)
{
s1=(i*ss)%p; if(s1==1 )
{
w=i; }
}
}
else if(ss==0)
{
w=p; }
s1=(w*s)%p; s2=(s1*s1-x2-x1)%p; if(s2<0)
{
s2=s2+p;
}
return s2;
}
else if(x2==x1 && y2==y1 && !y1==0)
{
s=(3*x1*x1+a)%p; if(s<0)
{
s=s+p;
}
s1=(2*y1)%p; if(s1<0)
{
s1=s1+p;
}
if(!s1==0)
{
for(i=1;i<=p-1;i++)
{
ss=(s1*i)%p; if(ss==1)
{
w=i;
}
}
```

```
}
else if(s1==0)
{
w=p; }
s2=(w*s)%p; s3=(s2*s2-2*x1)%p; if(s3<0)
{
s3=s3+p;
}
return s3;
}
}
int yp(int x1,int y1,int x2,int y2, int a, int p)
{
int s,s1,s2,s3,s4,s5,ss,sss,w,i; if(x2==x1 && y2==y1 && !y1==0)
{
s=(3*x1*x1+a)%p; s1=(2*y1)%p; if(s<0)
{
s=s+p;
}
if(s1<0)
{
s1=s1+p;
}
if(!(s1==0))
{
for(i=1;i<=p-1;i++)
{
ss=(i*s1)%p; if(ss==1)
{
w=i; }
}
}
else if(s1==0)
{
w=p; }
s2=(w*s)%p; s3=(3*x1-s2*s2)%p; s4=(s3*s2-y1)%p; if(s4<0)
{
s4=s4+p;
}
return s4;
}
else if(!(x2==x1))
```

```
{
s=(y2-y1)%p; ss=(x2-x1)%p; if(s<0)
{
s=s+p;
}
if(ss<0)
{
ss=ss+p;
}
if(!(ss==0))
{
for(i=1;i<=p-1;i++)
{
s1=(i*ss)%p; if(s1==1 )
{
w=i; }
}
}
else if(ss==0)
{
w=p; }
sss=(w*s)%p; s2=(-sss*sss+x2+2*x1)%p;
40
s3=(s2*sss-y1)%p; if(s3<0)
{
s3=s3+p;
}
return s3;
}
}
int prov(int x1,int y1,int a,int b,int p)
{
int s,s1,s2,s3,s4; s1=(y1*y1)%p; s2=(x1*x1)%p; s3=(s2*x1)%p; s4=s3+(a*x1+b)%p;
s=(s1-s4)%p; if(s<0)
{
s=s+p;
}
return s;
}
```

ДОДАТОК Б
Код основної програми і бібліотеки

```
#include<stdio.h> #include<math.h> #include<iostream> #include<fstream>
#include<algorithm> #include<stdlib.h> #include<vector> #include<iterator> # define
nnn 80 using namespace std; int xp(int x1, int y1, int x2,int y2,int a,int p); int yp(int x1,
int y1, int x2,int y2,int a, int p); int prov(int x1,int y1,int a, int b,int p); int xyz(char cx);
int nu(int t); int zyx(int m, int nol); int const n=800,nn=32; int main() {
//setlocale (lc_all, "rus"); int res[2][n+1],res0[2][n+1],res1[2][n+1],nol,l,xy[n+1]; int
a,b,c,cc,i,j,j0,p,x0,y0,x,y,x1,y1,k,kkk,zz; int x2,y2,x3,y3,x4,y4,xw,yw,k0,t,a1,a2; int
pb[2][1],zb,zzz,mas[5][2],jj,xx2,yy2; int xx01,xx02,xx11,xx12,xx21,xx22,xx31,xx32;
int xx41,xx42,xx51,xx52,zm,zz0; int  pm[2][nn+1],jjj[nn+1],kk[nn+1],de[2][n+1],n1;
int shifr[nn+1][4],des[n+1],nb,otv[2][nn+1]; int shifr0[nn+1][4],a3,a4; int
aa1,aa2,aa3,aa4,num,ss,p1,t1,k00;
//char str[80]="ab*c***defghij*klm**nopqrst**uvwxyz012*3**456789*";
//char str[80]="ab*c***defghij*klm**nopqrst**uvwxyz012*3**456789*** ";
//char str[80]="*a*b***cdefghi*jkl**mnopqrs**tuvwxyz01*2**3456789*"; char
str[80]="*b*a***cdefghi*jkl**mnopqrs**tuvwxyz01*2**3456789**  "; char
str1[80],cx; char str00=('0','1','2','3','4','5','6','7','8','9'); char str2[12][nn+1]; char
str33[12][nn*4+1]; int int2[12][nn+1]; printf("input k="); scanf(" %d", &k0);
printf("k0=%d \n",k0); printf("input p="); scanf("%d", &p); printf("p=%d\n",p);
printf("input a="); scanf("%d", &a); printf("a=%d \n",a); printf("input b="); scanf("%d",
&b); printf(" b=%d\n",b); printf("input kx="); scanf(" %d", &pb[0][0]);
printf("kx=%d\n",pb[0][0]); printf("input ky="); scanf(" %d", &pb[1][0]);
printf("ky=%d\n",pb[1][0]); printf("input text:"); for(i=0;i<=nn;i++) {
scanf("%c", &str1[i-1]); }
printf("proverka\n"); for (i=0;i<=nn-1;i++) {
printf("i=%d %c\n",i,str1[i]);
}
for(i=0;i<=nn-1;i++) for(j=0;j<=80;j++) {
if(str[j]==str1[i]) {
kk[i]=j; printf("i=%d kk=%d\n",i,kk[i]); }
}
p1=p; ss=1; for(k00=0;k00<=100;k00++) {
p1=(p1-p1%10)/10; if(p1>10) {
ss=ss+1; }
else {
num=ss+1; k00=100; }
}
printf("num=%d\n",num); res[0][0]=0; res[1][0]=376; x0=res[0][0]; y0=res[1][0];
x1=x0; y1=y0;
```

```
//c=y1-p; xx2=xp(x0,y0,x0,y0,a,p); res[0][1]=xx2; yy2=yp(x0,y0,x0,y0,a,p);
res[1][1]=yy2; x2=xx2; y2=yy2; //verhnya ocenka m<=p+2*p^0.5+1
for(k=2;k<=p+sqrt(p)+1;k++) {
xw=x2-x1; if((xw==0) && (y1+y2==p)) {
res[0][k]=-1000; res[1][k]=-1000; nol=k; printf("nol=%d\n",nol); res[0][k+1]=x0;
res[1][k+1]=y0; res[0][k+2]=res[0][1]; res[1][k+2]=res[1][1]; x2=x0; y2=y0;
x3=res[0][1]; y3=res[1][1]; k=k+2; }
else {
x3=xp(x1,y1,x2,y2,a,p); y3=yp(x1,y1,x2,y2,a,p); res[0][k]=x3; res[1][k]=y3; }
printf("k=%d (%d %d)(%d %d)\n",k,x2,y2,x3,y3); printf("prov(%d
%d)=%d\n",x3,y3,prov(x3,y3,a,b,p)); x1=x0;
x2=x3; y1=y0; y2=y3; }
printf("nol=%d\n",nol); if(k0>nol) {
k0=k0%nol; }
for(i=0;i<=nol-1;i++) {
x=res[0][i]; y=res[1][i]; l=(x-y); if(l<0) {
l=l+p; }
xy[i]=l; //printf("i=%d x-y=%d\n",i,xy[i]); }
for(i=0;i<=nn-1;i++) {
for(j=0;j<=nol-1;j++) {
if(xy[j]==kk[i] ) {
pm[0][i]=res[0][j]; pm[1][i]=res[1][j]; j=nol-1; }
}
printf("j=%d pmx=%d pmy=%d\n",i,pm[0][i],pm[1][i]);
}
for(i=0;i<=nn-1;i++)
{
printf("i=%d prov(pmx=%d
pmy=%d)=%d\n",i,pm[0][i],pm[1][i],prov(pm[0][i],pm[1][i],a,b,p)); }
for(i=0;i<=nol-1;i++) {
res0[0][i]=res[0][i]; res0[1][i]=res[1][i];
}
a1=res0[0][k0-1]; a2=res0[1][k0-1]; printf("kg=(%d %d) \n",a1,a2); //return 0;
for(j=0;j<=nn-1;j++) {
shifr[j][0]=a1; shifr[j][1]=a2; }
x0=pb[0][0]; y0=pb[1][0]; x2=xp(x0,y0,x0,y0,a,p); y2=yp(x0,y0,x0,y0,a,p);
res1[0][0]=x0; res1[1][0]=y0; res1[0][1]=x2; res1[1][1]=y2; x1=x0; y1=y0;
for(j=2;j<=k0-1;j++) {
xw=x2-x1; yw=y2-y1; if(xw==0 && y1+y2==p ) {
res1[0][j]=-1000; res1[1][j]=-1000;
```

```
printf("dot infinity"); res1[0][j+1]=res1[0][0]; res1[1][j+1]=res1[1][0];
res1[0][j+2]=res1[0][1]; res1[1][j+2]=res1[1][1]; y2=y0; x2=x0; x3= res1[0][1]; y3=
res1[1][1]; j=j+2; }
else if(!xw==0) {
x3=xp(x1,y1,x2,y2,a,p); y3=yp(x1,y1,x2,y2,a,p); res1[0][j]=x3; res1[1][j]=y3; }
x1=x0; x2=x3; y1=y0; y2=y3; }
printf("k0*pb xpb(%d * %d)=%d ypb(%d * %d)=%d\n",k0,pb[0][0],res1[0][k0-
1],k0,pb[1][0],res1[1][k0-1]); a1=res1[0][k0-1]; a2=res1[1][k0-1]; a3=562; a4=201;
printf("x2s=%d y2s=%d\n",xp(a3,a4,a1,a2,a,p ), yp(a3,a4,a1,a2,a,p ) ); for(j=0;j<=nn-
1;j++) {
x1=pm[0][j]; y1=pm[1][j]; x2=res1[0][k0-1];
y2=res1[1][k0-1]; xw=x2-x1; if(xw==0 && y1+y2==p) {
shifr[j][2]=-1000; shifr[j][3]=-1000; printf("dot=infinity\n");
}
else {
shifr[j][2]=xp(x1,y1,x2,y2,a,p); shifr[j][3]=yp(x1,y1,x2,y2,a,p); }
shifr[j][0]=shifr[0][0]; shifr[j][1]=shifr[0][1]; }
for(i=0;i<=nol-1;i++) {
if(res[0][i]==pb[0][0] && res[1][i]==pb[1][0]) {
nb=i+1; printf(" **** number open kluch i=%d ******\n",nb); }
}
x0=res[0][k0-1]; y0=res[1][k0-1]; x1=x0; y1=y0; de[0][0]=x0; de[0][1]=y0;
x2=xp(x0,y0,x0,y0,a,p); y2=yp(x0,y0,x0,y0,a,p); de[1][0]=x2; de[1][1]=y2;
for(j=2;j<=nb-1;j++) {
xw=x2-x1; if(xw==0 && y2+y1==p) {
de[j][0]=-1000; de[j][1]=-1000; printf("dot infinity"); de[j+1][0]=x0; de[j+1][1]=y0;
de[j+2][0]=de[1][0]; de[j+2][1]=de[1][1]; j=j+2; }
else {
x3=xp(x1,y1,x2,y2,a,p); y3=yp(x1,y1,x2,y2,a,p); de[0][j]=x3; de[1][j]=y3; }
x1=x0; x2=x3; y1=y0; y2=y3; }
for(j=0;j<=nn-1;j++) {
x1=de[0][nb-1]; y1=-de[1][nb-1]; x2=shifr[j][2]; y2=shifr[j][3]; xw=x2-x1; if(xw==0
&& y1+y2==p) {
otv[0][j]=-1000;
otv[1][j]=-1000; printf("dot infinity"); }
else {
otv[0][j]=xp(x1,y1,x2,y2,a,p); otv[1][j]=yp(x1,y1,x2,y2,a,p); }
}
printf("nb=%d\n",nb); // shifrovanie////////////////// for(j=0;j<=nn-1;j++) {
printf(" **** soobchenie xm=%d ym=%d *******\n",pm[0][j],pm[1][j]); printf("
******* begin shifrovanie ********   \n"); printf("       ***** infinity haz number = %d
********\n",nol+1); ///end shifrovanie////// ///deshifrovanie///////// xx51=otv[0][j];
```

```
xx52=otv[1][j]; printf("***** deshifrovanie *****\n"); printf("end x=%d end
y=%d\n",xx51,xx52); printf("proverka(%d
%d)=%d\n",xx51,xx52,prov(xx51,xx52,a,b,p)); des[j]=xx51-xx52; if(des[j]<0) {
des[j]=des[j]+p; }
printf("j=%d x=%d y=%d des=%d\n",j,xx51,xx52,des[j]); /// end shifrovanie////// }
printf("*****deshifrovanie****\n"); for(j=0;j<=nn-1;j++) {
jj=des[j]; printf("%c \n",str[jj]);
}
for(j=0;j<=nn-1;j++) {
jj=des[j]; printf("j=%d,jj=%d\n",j,jj);
}
printf("shifr of text is:\n"); for(i=0;i<=nn-1;i++) {
printf("[(%d,%d),(%d,%d)]\n",shifr[i][0],shifr[i][1],shifr[i][2],shifr[i][3]);
}
//////////////////// file.txt remove("balka1.txt"); remove("balka2.txt"); file*file;
ofstream out("balka2.txt", ios::app); file=fopen("balka2.txt","r"); out<<k0<<endl;
//fprintf(file,"%d",k0); fclose(file); printf("\n"); printf("//////////////////////"); int kk0=p;
printf("num(%d)=%d\n",kk0,nu(kk0));
printf("nachalo\n");////////////////////////////////////////////////////////////////////////////// //////////////
//return 0; for(i=0;i<=nn-1;i++) {
zyx(shifr[i][0],nol);
zyx(shifr[i][1],nol); zyx(shifr[i][2],nol); zyx(shifr[i][3],nol); }
fclose(file); int ind,sad[4*nn-1],mm; char arr00[nnn]; char rr[4*nn-1]; int
rf[4*nn+1][4],rs[4*nn+1]; ind=nu(nol); file=fopen("balka1.txt","r"); if(file==null) {
printf(" not open file");
}
else {
for(i=0; i<=4*nn-1;i++)
{
fgets(arr00,nnn,file); sad[i]=atoi(arr00);
}
}
fclose(file); for(i=0;i<=4*nn-1;i++) {
printf("i=%d l=%d\n", i ,sad[i]);
}
for(i=0;i<=4*nn-1;i++) {
t=i%4; t1=(i-i%4)/4;
shifr0[t1][t]=sad[i]; }
for(i=0;i<=nn-1;i++) {
printf("i=%d shifr0(%d %d),(%d
%d)\n",i,shifr0[i][0],shifr0[i][1],shifr0[i][2],shifr0[i][3]); }
```

```
x0=res[0][k0-1]; y0=res[1][k0-1]; x1=x0; y1=y0; de[0][0]=x0; de[0][1]=y0;
x2=xp(x0,y0,x0,y0,a,p); y2=yp(x0,y0,x0,y0,a,p); de[1][0]=x2; de[1][1]=y2;
for(i=0;i<=nol-1;i++) {
if(res[0][i]==pb[0][0] && res[1][i]==pb[1][0]) {
nb=i+1; printf(" **** number open kluch i=%d ******\n",nb); }
}
////////////////////////////////////
for(j=2;j<=nb-1;j++) {
xw=x2-x1; if(xw==0 && y2+y1==p) {
de[j][0]=-1000; de[j][1]=-1000; printf("dot infinity"); de[j+1][0]=x0;
de[j+1][1]=y0; de[j+2][0]=de[1][0]; de[j+2][1]=de[1][1]; j=j+2; }
else {
x3=xp(x1,y1,x2,y2,a,p); y3=yp(x1,y1,x2,y2,a,p); de[0][j]=x3; de[1][j]=y3; }
x1=x0; x2=x3; y1=y0; y2=y3; }
for(j=0;j<=nn-1;j++) {
x1=de[0][nb-1]; y1=-de[1][nb-1]; x2=shifr0[j][2]; y2=shifr0[j][3]; xw=x2-x1; if(xw==0
&& y1+y2==p) {
otv[0][j]=-1000; otv[1][j]=-1000; printf("dot infinity"); }
else {
otv[0][j]=xp(x1,y1,x2,y2,a,p); otv[1][j]=yp(x1,y1,x2,y2,a,p); }
}
printf("nb=%d\n",nb); // shifrovanie////////////////// for(j=0;j<=nn-1;j++) {
printf(" **** soobchenie0 xm0=%d ym0=%d *******\n",pm[0][j],pm[1][j]); printf("
******* begin shifrovanie0 ********   \n"); printf("       ***** infinity haz number =
%d ********\n",nol+1); ///end shifrovanie////// ///deshifrovanie///////// xx51=otv[0][j];
xx52=otv[1][j]; printf("***** deshifrovanie0 *****\n"); printf("end x0=%d end
y0=%d\n",xx51,xx52); printf("proverka0(%d
%d)=%d\n",xx51,xx52,prov(xx51,xx52,a,b,p)); des[j]=xx51-xx52; if(des[j]<0) {
des[j]=des[j]+p; }
printf("j=%d x0=%d y0=%d des0=%d\n",j,xx51,xx52,des[j]); /// end shifrovanie////// }
printf("*****deshifrovanie0****\n"); for(j=0;j<=nn-1;j++) {
jj=des[j]; printf("%c \n",str[jj]);
}
for(j=0;j<=nn-1;j++) {
jj=des[j]; printf("j=%d,jj=%d\n",j,jj);
}
printf("shifr0 of text is:\n"); for(i=0;i<=nn-1;i++) {
printf("[s=%d kg:(%d,%d),pm+k*pb:(%d,%d)]\n",i,shifr0[i][0],shifr0[i][1],shifr0[i][2],s
hifr0[i][3]); }
printf("end\n"); return 0;
}
```

ДОДАТОК В
Програма хеш – функції

```
#include<stdio.h> #include<math.h> int prog3( int t); int main() {
int m,n,m0,n0,res; n=123; n0=prog3(n); printf("n0=%d\n",n0); int bac[n0],k,i; k=n;
m=0; for(i=1;i<=n0;i++) {
bac[i]=k%10; printf("bac(%d)=%d\n",i,bac[i]); k=(k-k%10)/10;
m=m+bac[i]*pow(10,n0-i); }
printf("m=%d\n",m); if(n<=m) {
res=m%n;
}
else {
res=n%m;
}
printf("hesch(%d)=%d\n",n,res); }
int prog3( int t) {
int i,t1,sss; t1=t; sss=0; for(i=0;i<=100;i++)
{
if(t1>=10) {
t1=(t1-t1%10)/10; sss=sss+1; }
else if(t1<10) {
t1=t1; sss=sss+1; i=100; }
}
return sss; }
```

ДОДАТОК Г

Програма бібліотеки інтерфейса dll, виконано у програмному середовищі Compaq Visual Fortran 6.6

```
MODULE Primer USE Xeffort USE Xflogm USE DFWIN
USE KERNEL32
IMPLICIT NONE
INCLUDE 'Resource.fd'
INTEGER(4):: ICOLOR,ICOLOR1,IBKCOLOR,IBKCOLOR1,IBKCOLOR2
INTEGER(4):: a,b,p,z1,z2,k INTEGER(HANDLE):: HINSTANCE,HTHREADWORKER
!=============================================================
==============
CONTAINS
!=============================================================
==============
!PURPOSE: Initialization function called by XFT library on app start.
LOGICAL FUNCTION XInit(szCmdLine,nCmdShow) !DEC$IF (_DF_VERSION_.GE.650)
!DEC$ATTRIBUTES DEFAULT, DECORATE, ALIAS: 'XINIT':: XInit !DEC$ELSE
!DEC$ATTRIBUTES ALIAS: '_XINIT@12':: XInit !DEC$ENDIF
IMPLICIT NONE
CHARACTER(*), INTENT(IN)::    szCmdLine INTEGER, INTENT(IN)::        nCmdShow
LOGICAL::           bSt,FLAG IBKCOLOR=#CC3299
ICOLOR=#00FFFF
IBKCOLOR1=(#D7EBFA) !(#87B8DE)
ICOLOR1=#FF0000
IBKCOLOR2=(#EEEEAF)  !(#CBC0FF)  !(#D8BFD8)  !(#DDA0DD) !
!(#D7EBFA)
IF (XCreateDialogApp(IDD_MAIN, idIcon=IDI_ICON_MAIN)) THEN !Todo: Initialize dialog
controls here via XCtlSet(Sub)
FLAG=DLGSET(XW_FRAME,IDD_MAIN,IBKCOLOR,DLG_BKCOLOR)
!End Todo: Initialize dialog controls here via XCtlSet(Sub)
FLAG=DLGSETSUB(XW_FRAME,PUSK,PUSKSUB_THREAD)
FLAG=DLGSETSUB(XW_FRAME,PUSK2,STOP_SUB)
FLAG=DLGSETSUB(XW_FRAME,PUSK3,CLEAR_SUB)
FLAG=DLGSETSUB(XW_FRAME,IDC_EDIT2,SUB_EDIT2,DLG_CHANG
E)
FLAG=DLGSETSUB(XW_FRAME,IDD_MAIN,SUB_MAIN)
FLAG=DLGSET(XW_FRAME,IDD_MAIN,IBKCOLOR,DLG_BKCOLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC1,IBKCOLOR,DLG_BKCOLOR
)
FLAG=DLGSET(XW_FRAME,IDC_STATIC2,IBKCOLOR,DLG_BKCOLOR
)
FLAG=DLGSET(XW_FRAME,IDC_STATIC3,IBKCOLOR,DLG_BKCOLOR
)
FLAG=DLGSET(XW_FRAME,IDC_STATIC4,IBKCOLOR,DLG_BKCOLOR
)
FLAG=DLGSET(XW_FRAME,IDC_STATIC5,IBKCOLOR,DLG_BKCOLOR
)
```

```
FLAG=DLGSET(XW_FRAME,IDC_STATIC6,IBKCOLOR,DLG_BKCOLOR
)
FLAG=DLGSET(XW_FRAME,IDC_STATIC7,IBKCOLOR,DLG_BKCOLOR
)
FLAG=DLGSET(XW_FRAME,IDC_STATIC8,IBKCOLOR,DLG_BKCOLOR
)
FLAG=DLGSET(XW_FRAME,IDC_STATIC9,IBKCOLOR,DLG_BKCOLOR
)
FLAG=DLGSET(XW_FRAME,IDC_STATIC10,IBKCOLOR,DLG_BKCOLO
R)
FLAG=DLGSET(XW_FRAME,IDC_STATIC11,IBKCOLOR,DLG_BKCOLO
R)
FLAG=DLGSET(XW_FRAME,IDC_STATIC12,IBKCOLOR,DLG_BKCOLO
R)
FLAG=DLGSET(XW_FRAME,IDC_STATIC13,IBKCOLOR,DLG_BKCOLO
R)
FLAG=DLGSET(XW_FRAME,IDC_STATIC14,IBKCOLOR,DLG_BKCOLO
R)
FLAG=DLGSET(XW_FRAME,IDC_STATIC1,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC2,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC3,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC4,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC5,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC6,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC7,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC8,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC9,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC10,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC11,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC12,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_STATIC13,ICOLOR,DLG_COLOR)
63
FLAG=DLGSET(XW_FRAME,IDC_STATIC14,ICOLOR,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT1,ICOLOR1,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT2,ICOLOR1,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT3,ICOLOR1,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT4,ICOLOR1,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT5,ICOLOR1,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT6,ICOLOR1,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT7,ICOLOR1,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT8,ICOLOR1,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT9,ICOLOR1,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT10,ICOLOR1,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT11,ICOLOR1,DLG_COLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT1,IBKCOLOR2,DLG_BKCOLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT2,IBKCOLOR2,DLG_BKCOLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT3,IBKCOLOR2,DLG_BKCOLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT4,IBKCOLOR1,DLG_BKCOLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT5,IBKCOLOR1,DLG_BKCOLOR)
```

```
FLAG=DLGSET(XW_FRAME,IDC_EDIT6,IBKCOLOR1,DLG_BKCOLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT7,IBKCOLOR1,DLG_BKCOLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT8,IBKCOLOR1,DLG_BKCOLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT9,IBKCOLOR1,DLG_BKCOLOR)
FLAG=DLGSET(XW_FRAME,IDC_EDIT10,IBKCOLOR1,DLG_BKCOLOR
)
64
FLAG=DLGSET(XW_FRAME,IDC_EDIT11,IBKCOLOR2,DLG_BKCOLOR
)
!-------------------------------------------------
FLAG=DLGSET(XW_FRAME,IDC_EDIT2,'polotsk 2016')
FLAG=DLGSET(XW_FRAME,IDC_EDIT1,'96587')
FLAG=DLGSET(XW_FRAME,IDC_EDIT4,'-1')
FLAG=DLGSET(XW_FRAME,IDC_EDIT5,'188')
FLAG=DLGSET(XW_FRAME,IDC_EDIT6,'751')
FLAG=DLGSET(XW_FRAME,IDC_EDIT7,'201')
FLAG=DLGSET(XW_FRAME,IDC_EDIT8,'5')
FLAG=DLGSET(XW_FRAME,IDC_EDIT9,'17')
!FLAG=DLGSET(XW_FRAME,IDC_EDIT10,'12')
!-------------------------------------------------
IF (XModalDialog(XW_FRAME).EQ.IDOK) THEN !Todo: place logic after dialog is exited via OK
button here (XCtlGet)
END IF
END IF
!For a dialog-bassed application, the XInit may return .FALSE. !Alternatively, use XModelessDialog,
but return .TRUE. and set !handlers for IDOK/IDCANCEL buttons.
XInit = .FALSE.
END FUNCTION XInit
!=======================================================
===============
SUBROUTINE PUSKSUB_THREAD(DLG,C_NAME,CBTYPE)
IMPLICIT NONE
TYPE(DIALOG):: DLG
INTEGER(4),INTENT(IN):: C_NAME,CBTYPE
65
LOGICAL:: FLAG
TYPE (T_SECURITY_ATTRIBUTES),POINTER:: NULL_SA
INTEGER(4):: idTHREADWORKER HTHREADWORKER = CreateThread(
NULL_SA, 0,LOC(PUSKSUB),   & LOC(dlg), 0 , LOC(idTHREADWORKER) )
FLAG=SetThreadPriority(HTHREADWORKER,THREAD_PRIORITY_NOR MAL)
ENDSUBROUTINE PUSKSUB_THREAD
!=============================================================
===============
SUBROUTINE STOP_SUB(DLG,C_NAME,CBTYPE)
IMPLICIT NONE
TYPE(DIALOG):: DLG
```

```
INTEGER(4),INTENT(IN):: C_NAME,CBTYPE
LOGICAL:: FLAG
TYPE (T_SECURITY_ATTRIBUTES),POINTER:: NULL_SA
INTEGER(4):: idTHREADWORKER CALL  CLOSE_LIBRARY(HINSTANCE);
FLAG=TERMINATETHREAD(HTHREADWORKER,0)
IF(FLAG) THEN
FLAG=DLGSET(DLG,IDC_STATIC12,'Process terminated succesfully!') ELSE
FLAG=DLGSET(DLG,IDC_STATIC12,'Process not terminated!
Unsuccesfully. ') ENDIF
ENDSUBROUTINE STOP_SUB
!=============================================================
================
SUBROUTINE CLEAR_SUB(DLG,C_NAME,CBTYPE)
IMPLICIT NONE
TYPE(DIALOG):: DLG
INTEGER(4),INTENT(IN):: C_NAME,CBTYPE
LOGICAL:: FLAG
TYPE (T_SECURITY_ATTRIBUTES),POINTER:: NULL_SA
INTEGER(4):: idTHREADWORKER FLAG=DLGSET(DLG,IDC_EDIT1,' ')
66
FLAG=DLGSET(DLG,IDC_EDIT3,' ')
FLAG=DLGSET(DLG,IDC_STATIC12,' ')
FLAG=DLGSET(DLG,IDC_EDIT11,' ')
ENDSUBROUTINE CLEAR_SUB
!=============================================================
================
SUBROUTINE SUB_EDIT2(DLG,C_NAME,CBTYPE)
IMPLICIT NONE
TYPE(DIALOG):: DLG
INTEGER(4),INTENT(IN):: C_NAME,CBTYPE
LOGICAL:: FLAG
CHARACTER(LEN=1000):: STROKA
INTEGER(4):: LENTH
FLAG=DLGSET(DLG,IDC_EDIT10,' ')
FLAG=DLGGET(DLG,IDC_EDIT2,STROKA)
LENTH=LEN_TRIM(STROKA)
WRITE(STROKA,*) LENTH
STROKA=ADJUSTL(STROKA)
FLAG=DLGSET(DLG,IDC_EDIT10,TRIM(STROKA))
ENDSUBROUTINE SUB_EDIT2
!=============================================================
================
```

```
SUBROUTINE SUB_MAIN(DLG,C_NAME,CBTYPE)
IMPLICIT NONE
TYPE(DIALOG):: DLG
INTEGER(4),INTENT(IN):: C_NAME,CBTYPE
LOGICAL:: FLAG
CHARACTER(LEN=1000):: STROKA
INTEGER(4):: LENTH
CALL SUB_EDIT2(DLG,C_NAME,CBTYPE)
ENDSUBROUTINE SUB_MAIN
!=============================================================
===============
SUBROUTINE PUSKSUB(DLG,C_NAME,CBTYPE)
!USE ISO_C_BINDING
IMPLICIT NONE
TYPE(DIALOG):: DLG
INTEGER(4),INTENT(IN):: C_NAME,CBTYPE
67
LOGICAL:: FLAG
CHARACTER(LEN=80)::STRING
CHARACTER(LEN=10000)::STRING_OUT,STRING_CIFER
CHARACTER(LEN=1000)::STR
CHARACTER(LEN=1):: CHAR1
INTEGER(4):: J,I
INTEGER(4):: nDLL_C_PLUS_PLUS INTEGER(4):: a,b,p,z1,z2,k
CHARACTER(LEN=1000)::STR1
CHARACTER(LEN=1000)::STR_DLL
INTEGER(4)::shifr(0:1000,0:3),shifr0(0:1000,0:3) INTEGER(4):: n,nn,nnn
INTEGER(4)::n7,nn97,nnn7 INTEGER(4):: t INTERFACE
INTEGER FUNCTION NU_DLL(N1)
INTEGER ,INTENT(IN):: N1
ENDFUNCTION
END INTERFACE
INTERFACE
INTEGER FUNCTION fort_nu(N1) INTEGER ,INTENT(IN):: N1
ENDFUNCTION
END INTERFACE
INTERFACE
SUBROUTINE fnDLL_C_PLUS_PLUS ENDSUBROUTINE
END INTERFACE
INTERFACE
SUBROUTINE fn_main ENDSUBROUTINE
END INTERFACE
```

```
POINTER(FARPROC0,fnDLL_C_PLUS_PLUS) POINTER(FARPROC1,a)
68
POINTER(FARPROC2,b) POINTER(FARPROC3,p) POINTER(FARPROC4,z1)
POINTER(FARPROC5,z2) POINTER(FARPROC6,k) POINTER(FARPROC7,str1)
POINTER(FARPROC8,nn) POINTER(FARPROC9,shifr)
POINTER(FARPROC10,shifr0) POINTER(FARPROC15,NU_DLL)
POINTER(FARPROC20,nDLL_C_PLUS_PLUS) POINTER(FARPROC25,fn_main)
POINTER(FARPROC30,n7) POINTER(FARPROC35,nn97)
POINTER(FARPROC40,nnn7) POINTER(FARPROC45,fort_nu)
POINTER(FARPROC50,STR_DLL)
HINSTANCE=LoadLibrary("DLL_C_PLUS_PLUS"C) IF(HINSTANCE/=0) THEN
FLAG=DLGSET(XW_FRAME,IDC_STATIC12,'Длл успешно загружена!')
CONTINUE
ELSE
FLAG=DLGSET(XW_FRAME,IDC_STATIC12,'Длл не загружена!') CALL
CLOSE_LIBRARY(HINSTANCE); RETURN ;
ENDIF
FARPROC0=GetProcAddress(HINSTANCE,"?fnDLL_C_PLUS_PLUS@@Y
AXXZ"C)
FARPROC1=GetProcAddress(HINSTANCE,"?a@@3HA"C) !a
FARPROC2=GetProcAddress(HINSTANCE,"?b@@3HA"C) !b
FARPROC3=GetProcAddress(HINSTANCE,"?p@@3HA"C) !p
FARPROC4=GetProcAddress(HINSTANCE,"?z1@@3HA"C) !z1
FARPROC5=GetProcAddress(HINSTANCE,"?z2@@3HA"C) !z2
FARPROC6=GetProcAddress(HINSTANCE,"?k@@3HA"C)  !k
69
FARPROC7=GetProcAddress(HINSTANCE,"?str1@@3PADA"C) !str1
FARPROC8=GetProcAddress(HINSTANCE,"?nn@@3HA"C) !nn
FARPROC9=GetProcAddress(HINSTANCE,"?shifr@@3PAY03HA"C) !shifr
FARPROC10=GetProcAddress(HINSTANCE,"?shifr0@@3PAY03HA"C) !shifr0
FARPROC15=GetProcAddress(HINSTANCE,"?nu@@YAHH@Z"C) !shifr0
FARPROC20=GetProcAddress(HINSTANCE,"?nDLL_C_PLUS_PLUS@@3H A"C)
!shifr0 FARPROC25=GetProcAddress(HINSTANCE,"?fn_main@@YAXXZ"C)
!shifr0
FARPROC30=GetProcAddress(HINSTANCE,"?n7@@3HA"C) !shifr0
FARPROC35=GetProcAddress(HINSTANCE,"?nn97@@3HA"C) !shifr0
FARPROC40=GetProcAddress(HINSTANCE,"?nnn7@@3HA"C) !shifr0
FARPROC45=GetProcAddress(HINSTANCE,"?fort_nu@@YAHXZ"C) !shifr0
FARPROC50=GetProcAddress(HINSTANCE,"?str10@@3PADA"C) !shifr0
FLAG=DLGSET(DLG,IDC_STATIC12,' ')
FLAG=DLGGET(DLG,IDC_EDIT2,str1)
FLAG=DLGGET(DLG,IDC_EDIT4,STRING)
```

```
READ(STRING,'(I10)',ERR=1) a FLAG=DLGGET(DLG,IDC_EDIT5,STRING)
READ(STRING,'(I10)',ERR=1) b FLAG=DLGGET(DLG,IDC_EDIT6,STRING)
READ(STRING,'(I10)',ERR=1) p FLAG=DLGGET(DLG,IDC_EDIT7,STRING)
READ(STRING,'(I10)',ERR=1) z1 FLAG=DLGGET(DLG,IDC_EDIT8,STRING)
READ(STRING,'(I10)',ERR=1)z2 FLAG=DLGGET(DLG,IDC_EDIT9,STRING)
READ(STRING,'(I10)',ERR=1) k
70
FLAG=DLGGET(DLG,IDC_EDIT10,STRING)
READ(STRING,'(I10)',ERR=1) nn
IF(NN<1) THEN
FLAG=DLGSET(DLG,IDC_STATIC12,'Error parameter nn.Program terminated. ')
RETURN
ENDIF
FLAG=(FARPROC0/=0).AND.(FARPROC1/=0).AND.(FARPROC2/=0).AND.
(FARPROC3/=0).AND.(FARPROC4/=0).AND.(FARPROC5/=0).AND. &
(FARPROC6/=0).AND.(FARPROC7/=0).AND.(FARPROC8/=0).AND.(FARP
ROC9/=0).AND.(FARPROC10/=0).AND.(FARPROC15/=0) &
.AND.(FARPROC20/=0).AND.(FARPROC25/=0)
IF(FLAG) THEN
CONTINUE
ELSE
FLAG=DLGSET(XW_FRAME,IDC_STATIC12,'Не все адреса процедур
получены!') CALL  CLOSE_LIBRARY(HINSTANCE); RETURN ;
ENDIF
!FLAG=DLGGET(DLG,IDC_EDIT1,STRING)
!READ(STRING,'(I10)',ERR=1) nDLL_C_PLUS_PLUS
!I=fort_nu(nDLL_C_PLUS_PLUS) !WRITE(STRING,*) I;
STRING=ADJUSTL(STRING);
!FLAG=DLGSET(XW_FRAME,IDC_STATIC12,'NU='//TRIM(STRING))
!CALL fnDLL_C_PLUS_PLUS() nnn7=80 ; n7=800 ; !nn9=1000; CALL fn_main()
STRING_OUT=' ';STRING_CIFER=' ';
DO J=1,4*NN,1
71
WRITE(STR,*) shifr(J-1,0) STR=ADJUSTL(STR)
STRING_CIFER=TRIM(STRING_CIFER)//'  '//TRIM(STR)
WRITE(STR,*) shifr0(J-1,0) STR=ADJUSTL(STR)
STRING_OUT=TRIM(STRING_OUT)//'  '//TRIM(STR)
ENDDO
STRING_CIFER=ADJUSTL(STRING_CIFER)
STRING_OUT=ADJUSTL(STRING_OUT)
FLAG=DLGSET(XW_FRAME,IDC_EDIT1,TRIM(STRING_CIFER))
FLAG=DLGSET(XW_FRAME,IDC_EDIT3,TRIM(STRING_OUT))
```

```
STR_DLL=ADJUSTL(STR_DLL)
FLAG=DLGSET(XW_FRAME,IDC_EDIT11,TRIM(STR_DLL))
FLAG=DLGSET(XW_FRAME,IDC_STATIC12,'Finished!')
RETURN
1 FLAG=DLGSET(DLG,IDC_STATIC12,'Error input data') CALL
CLOSE_LIBRARY(HINSTANCE);
ENDSUBROUTINE PUSKSUB
!=============================================================
===============
SUBROUTINE CLOSE_LIBRARY(HINSTANCE)
USE KERNEL32
IMPLICIT NONE
INTEGER(HANDLE),INTENT(IN):: HINSTANCE
LOGICAL:: FLAG
FLAG=FreeLibrary(HINSTANCE) IF(FLAG) THEN
FLAG=CLOSEHANDLE(HINSTANCE)
FLAG=DLGSET(XW_FRAME,IDC_STATIC12,'Длл отключена!') !CALL
SLEEPQQ(100)
ENDIF
END SUBROUTINE CLOSE_LIBRARY
!=============================================================
===============
END MODULE Primer
```