МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

СУСЛІН Віталій Вікторович

Програмна підсистема модулярного експоненціювання для асиметричних криптосистем / The software subsystem of the modular exponentiation in asymmetric cryptosystems

спеціальність: 6.050102 - Комп'ютерна інженерія
освітньо-професійна програма - Комп'ютерні системи та мережі

Випускна кваліфікаційна робота

Виконав: студент групи КСМ-42/1
Суслін Віталій Вікторович
_____

Науковий керівник:
к.т.н., доцент Касянчук М.М.
_____

Випускну кваліфікаційну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
О. М. Березький

ТЕРНОПІЛЬ - 2019

РЕЗЮМЕ

Бакалаврська робота містить 86 сторінок пояснюючої записки, 18 рисунки, 20 таблиць, 3 додатки. Обсяг графічного матеріалу – 3 аркуші формату А3.

Метою дипломної роботи є розробка програмної підсистеми модулярного експоненціювання для асиметричних криптосистем.

Методи досліджень – методи програмування, методи захисту інформації, методи теорії графів.

У дипломній роботі розроблено програмну підсистему модулярного експоненціювання для асиметричних криптосистем. На основі аналізу криптографічних методів шифрування встановлено їх найбільш поширені арифметичні операції, визначено основні переваги та недоліки їх реалізації.. На основі векторно-модульного методу розроблено алгоритм пошуку залишку. На основі формування вимог до реалізації операцій модулярного множення та експоненціювання розроблено алгоритмічне забезпечення. На основі розробленого алгоритмічного забезпечення побудовано UML Use-case діаграму програмної підсистеми. На основі UML Use-case діаграм розроблено програмну підсистему модулярного множення та експоненціювання, тестування якого підтвердило збільшення швидкодії виконання вказаних операцій у порівнянні з класичними алгоритмами в асиметричних криптосистемах.

Аналіз спроектованої системи дозволяє зробити висновок, що розроблена програмна підсистема модулярного експоненціювання для асиметричних криптосистем задовольняє вимогам сучасного ринку, які ставляться користувачами, проте сучасна елементна база та програмне забезпечення розвиваються стрімкими темпами і внесення змін і доповнень до даного продукту є обов’язковою вимогою.

Ключові слова: ПРОГРАМНА ПІДСИСТЕМА, ЗАЛИШОК, МОДУЛЯРНЕ МНОЖЕННЯ, МОДУЛЯРНЕ ЕКСПОНЕНЦІЮВАННЯ, UML USE-CASE ДІАГРАМА.

RESUME

The Bachelor paper contains 86 pages of explanatory notes, 18 drawings, 20 tables, 3 appendices. The volume of graphic material - 3 sheets of A3 format.

The purpose of the thesis is to develop a software subsystem of modular exponentiation for asymmetric cryptosystems.

Methods of research - methods of programming, methods of information security, methods of graph theory.

In the thesis the software sub-system of modular exponentiation for asymmetric cryptosystems has been developed. Based on the analysis of cryptographic encryption methods, their most common arithmetic operations were determined, the main advantages and disadvantages of their implementation were determined. On the basis of the vector-modular method, an algorithm for finding the remainder was developed. Based on the formation of requirements for the implementation of operations of modular multiplication and exponentiation, algorithmic support is developed. Based on the developed algorithmic support, the UML Use-case diagram of the software subsystem was built. On the basis of the UML Use-case diagrams, a software subsystem of modular multiplication and exponentiation was developed, the testing of which confirmed the increase in the performance of these operations compared with the classical algorithms in asymmetric cryptosystems.

The analysis of the designed system allows us to conclude that the developed software sub-system of modular exponentiation for asymmetric cryptosystems meets the requirements of the modern market, which are put by users, but the modern elemental base and software are developing at a rapid pace and making changes and additions to this product is a mandatory requirement.

Keywords SOFTWARE SUBMISSION, SCALE, MODULAR MANNING, MODULAR EXPONENTATION, UML USE-CASE DIAGRAM.

ЗМІСТ

Вступ 9
1 Аналіз предметної області 10
1.1 Аналіз найбільш поширених асиметричних криптосистем 10
1.2 Алгоритми пошуку залишку 19
1.3 Аналіз технічного завдання та постановка задачі 22
2 Алгоритми модулярного екпоненціювання 24
2.1 Алгоритми модулярного множення 24
2.2 Алгоритми модулярного експоненціювання 29
2.3 Векторно-модульний метод модулярного експоненціювання 31
3 Програмна реалізація 34
3.1 Обгрунтування вибору мови програмування Java 34
3.2 Програмне вирішення задачі 35
3.3 Приклад модулярного експоненціювання для використання в криптосистемах RSA та Ель-Гамал. 40
4 Техніко-економічний розділ 44
4.1 Розрахунок витрат на розробку програмного додатку 44
4.2 Визначення експлуатаційних витрат 49
4.3 Визначення економічної ефективності і терміну окупності капітальних вкладень 53
Висновки 55
Список використаних джерел 56
Додаток А. Лістинг коду програми 59
Додаток Б. Світлокопії публікацій 83
Додаток В. Довідка про використання 86

ВСТУП

На даний час асиметричні криптосистеми відіграють дуже важливу роль під час захисту інформаційних потоків від несанкціонованого доступу. До найбільш поширених асиметричних відносяться криптосистеми *RSA*, Рабіна та Ель-Гамаля. Основними операціями асиметричних криптосистем є модулярне множення, модулярне експоненціювання та пошук оберненого елемента за модулем [1].

Проблемою сучасних крипто алгоритмів є проблематика швидкого обчислення і практично є досить громіздкими в обчисленні, що досить є зетратними у використанні ресурсів. Ресурсами зазвичай є час, пам’ять для зберігання даних, потужності центрального процесора для обробки обчислень та витраченого часу для досліджень працездатності алгоритмів.

Таким чином, актуальною є науково-технічна задача зменшення обчислювальної складності, підвищення швидкодії алгоритмів, спеціалізованого програмного і апаратного забезпечення при виконанні арифметичних операцій в асиметричних криптосистемах [2].

Розробка програмних рішень дозволяє здійснювати обчислення значно швидко, а реалізовування є досить складним, проте це дозволяє користувачам алгоритмів застосовувати їх у своїх потребах, без затрати часу на реалізацію обчилень. Тому для цього існує ряд мов програмування, для створення програмних рішень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз найбільш поширених асиметричних криптосистем

Відомо, що усім симетричним криптосистемам, у яких шифрування та розшифровування відбувається за допомогою одного і того самого ключа, притаманні такі основні недоліки [3]:

– принциповою є надійність каналу передачі ключа другому учаснику секретних переговорів. Інакше кажучи, ключ повинен передаватися по секретному каналу;

– до служби генерації ключів пред'являються підвищені вимоги, обумовлені тим, що для *j* абонентів при схемі взаємодії "кожен з кожним" потрібно *j*(*j*-1)/2 ключів, тобто залежність кількості ключів від кількості абонентів є квадратичною.

Для вирішення перерахованих вище проблем симетричного шифрування призначені системи з асиметричним шифруванням або шифруванням з відкритим ключем (рисунок 1.1), що використовують властивості функцій з секретом, розроблених Діффі і Хеллманом.

Рисунок 1.1 – Схема асиметричних криптосистем

Ці системи характеризуються наявністю у кожного абонента двох ключів: відкритого і закритого (секретного). При цьому відкритий ключ

відкрито передається всім учасникам секретних переговорів. Таким чином, вирішуються дві проблеми:

– немає потреби в секретній доставці;

– відсутня квадратична залежність кількості ключів від кількості користувачів – для *j* користувачів потрібно 2*j* ключів.

Першим шифром, розробленим на принципах асиметричного шифрування, є шифр RSA. Він названий так за першими літерами прізвищ його винахідників [4]: Рона Райвеста, Аді Шаміра і Леонарда Елдемана – засновників компанії RSA Data Secutity. RSA – не тільки найпопулярніший з асиметричних шифрів, але, мабуть, взагалі найвідоміший.

Математичне обгрунтування RSA таке: пошук дільників дуже великого натурального числа, яке є добутком двох простих чисел – дуже трудомістка процедура. Відповідно, за допомогою відкритого ключа дуже складно обчислити відповідний йому таємний ключ. Схема криптоалгоритму RSA представлена на рисунку 1.2.

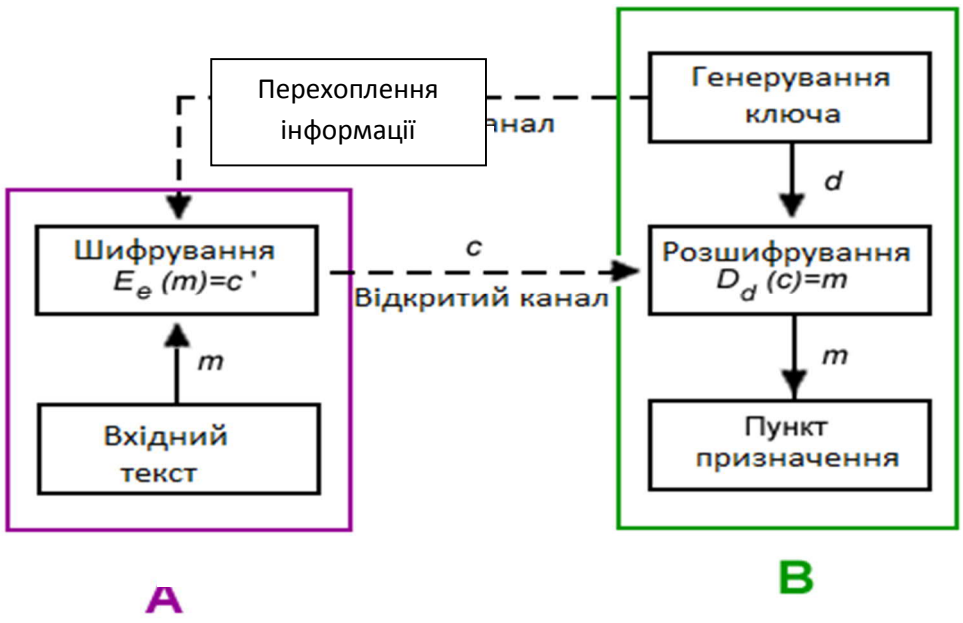


Рисунок 1.2 – Криптоалгоритм RSA

Шифр RSA всебічно вивчений і визнаний стійким при достатній довжині ключів. Наприклад, 512 біт для забезпечення стійкості не вистачає, а

1024 біти вважається прийнятним варіантом. З ростом потужності процесорів при даній довжині ключа RSA втрачає стійкість до атаки повним перебором, однак це дозволяє застосувати більш довгі ключі, що в свою чергу підвищить стійкість шифру [5].

Шифр працює за алгоритмом, який включає в себе генерацію ключів, шифрування та дешифрування.

Для того, щоби згенерувати пари ключів, виконуються такі дії:

– вибираються два великі прості числа p та q;

– обчислюється їх добуток $n=p*q$;

– обчислюється функція Ейлера $\varphi(n)=(p-1)(q-1)$;

– вибирається ціле число e таке, що$1<e<\varphi(n)$ та e - взаємно просте з $\varphi(n)$, тобто НСД(e, $\varphi(n)$)=1;

– на основі розширеного алгоритму Евкліда знаходиться число d таке, що$e\ d \pmod{\varphi(n)}=1$ або $d=e^{-1} \pmod{\varphi(n)}$.

Число n називається модулем, а числа e і d – відкритою та секретною експонентами відповідно. Пара чисел (n,e) є відкритою частиною ключа, а пара (n, d) – секретною. Числа p і q після генерації ключів можуть бути знищені, однак у жодному разі не повинні бути розкриті.

Для шифрування повідомлення припустимо, що перший абонент хоче відправити другому повідомлення A. Для початку він за допомогою узгодженого протоколу перетворення, відомого як доповняльні схеми або таблиці перетворення, перетворює A в ціле число a так, щоб $0 \le a \le n$. Опісля він обчислює зашифрований тест A', використовуючи відкритий ключ другого абонента e, за допомогою рівняння:

$$A' = a^e \bmod n. \qquad (1.1)$$

Це може бути зроблено досить швидко, навіть у випадку, коли текст перевищує 500-бітну розрядність.

Розшифрування відбувається наступним чином:

$$a=(A')^d \bmod n. \quad (1.2)$$

Відповідно при виконанні даної операції відновлюєтсья вихідне повідомлення:

$$(A')^d \equiv (a^e)^d \equiv a^{ed} \pmod{n}. \quad (1.3)$$

З умови

$$ed \equiv 1 \pmod{\varphi(n)} \quad (1.4)$$

випливає твердження, що $ed \equiv k_0 \varphi(n) + 1$ для деякого цілого k_0, отже:

$$a^{ed} \equiv a^{k0\varphi(n)+1} \pmod{n}. \quad (1.5)$$

Згідно з теоремою Ейлера:

$$a^{\varphi(n)} \equiv 1 \pmod{n}, \quad (1.6)$$

тому

$$a^{k0\varphi(n)+1} \equiv a \pmod{n}, (A')^d \equiv a \pmod{n}. \quad (1.7)$$

Для шифрування необхідно знати пару чисел *e, n*, для дешифрування – *d, n*. Перша пара – відкритий ключ, друга – закритий. Знаючи відкритий ключ, можна обчислити значення закритого ключа. Необхідною проміжною дією цього перетворення є знаходження множників *p* і *q*, для чого потрібно

розкласти n на множники – ця процедура займає дуже багато часу. Саме з величезною обчислювальною складністю пов'язана криптостійкість RSA [6].

Криптосистема Рабіна стала першою асиметричною криптосистемою, яка грунтується на складності знаходження квадратичного лишку. Вона, як і будь-яка асиметрична криптосистема, використовує відкритий і закритий ключі. Відкритий ключ використовується для шифрування повідомлень і може бути опублікований для загального огляду. Закритий ключ необхідний для розшифрування і повинен бути відомий тільки одержувачу зашифрованих повідомлень [7]. Схема шифрування згідно криптосистеми Рабіна наведена на рисунку 1.3.

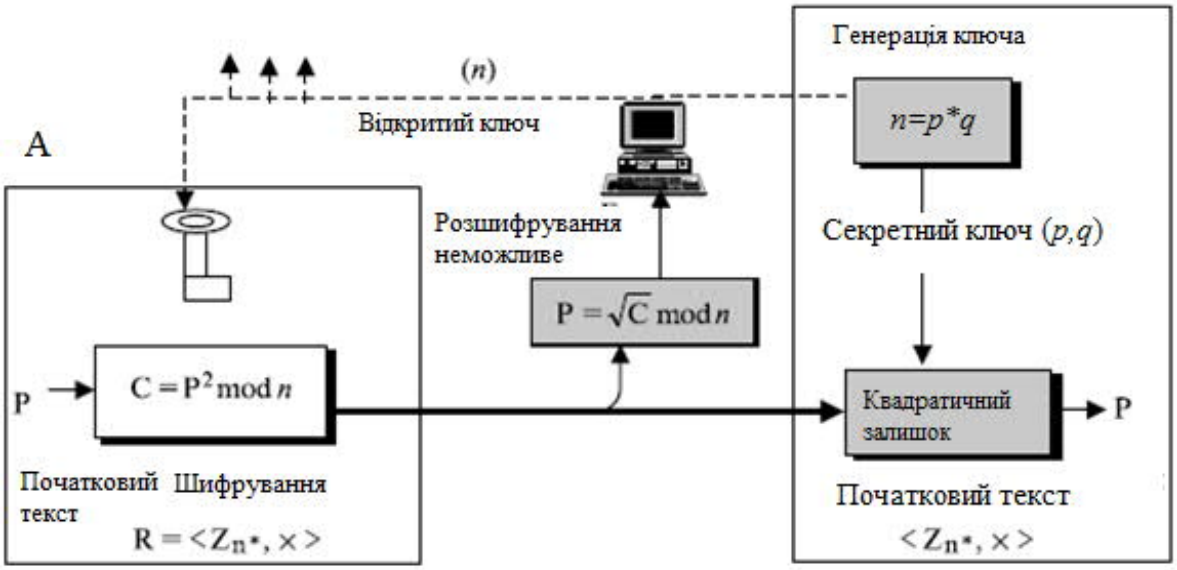


Рисунок 1.3 – Схема шифрування згідно криптосистеми Рабіна

Для генерації ключів вибираються два випадкових числа p та q із урахуванням таких вимог:

– числа повинні бути великими;

– числа повинні бути простими;

– повинна виконуватися умова: $p \equiv q \equiv 3 \bmod 4$.

Виконання цих вимог сильно прискорює процедуру знаходження коренів за модулем p і q. Далі обчислюється число $n=p\cdot q$, тоді число n – відкритий ключ, числа p і q - закритий.

Початкове повідомлення A (текст) шифрується з допомогою відкритого ключа-числа n – за такою формулою:

$$A' = A^2 \bmod n. \tag{1.8}$$

Завдяки використанню операції піднесення до квадрату за модулем швидкість шифрування системи Рабіна більша, ніж швидкість шифрування за методом RSA, навіть якщо в останньому випадку значення експоненти вибрати невелике.

При дешифруванні криптограми A' для зручності вводяться додаткові допоміжні величини c_1 і c_2:

$$c_1 = \sqrt[\circ]{A'} \bmod \sqrt[\circ]{p},\ c_2 = \sqrt[\circ]{A'} \bmod \sqrt[\circ]{q}, \tag{1.9}$$

Для знаходження A необхідно знайти квадратичні лишки c_1, c_2 відповідно за модулями p і q:

$$x^2 \equiv c_1 (\bmod\ p),\ y^2 \equiv c_2 (\bmod\ q) \tag{1.10}$$

В результаті можна записати чотири системи порівнянь:

$$\begin{cases} A_1 \equiv x (\bmod\ p); \\ A_1 \equiv y (\bmod\ q); \end{cases} \begin{cases} A_2 \equiv x (\bmod\ p); \\ A_2 \equiv -y (\bmod\ q); \end{cases} \begin{cases} A_3 \equiv -x (\bmod\ p); \\ A_3 \equiv y (\bmod\ q); \end{cases} \begin{cases} A_4 \equiv -x (\bmod\ p); \\ A_4 \equiv -y (\bmod\ q); \end{cases} \tag{1.11}$$

Одне з рішень (1.11), яке ґрунтується на використанні китайської теореми про залишки, буде шуканим повідомленням A.

Розшифрування тексту, крім правильного, приводить ще до трьох хибних результатів.

Це і є головною проблемою незручності криптосистеми Рабіна, а також одним із факторів, що перешкоджали тому, щоб вона знайшла широке практичне використання [8-9].

Якщо вихідний текст являє собою текстове повідомлення, то визначення правильного тексту не складним. Однак якщо повідомлення є потоком випадкових бітів (наприклад, для генерування ключів чи цифрового підпису), то тоді визначення потрібного тексту стає реальною проблемою. Одним із способів усунути цей недолік є додавання до повідомлення перед шифруванням відомого заголовка або якоїсь мітки[10-12].

Однак у класичній криптосистемі Рабіна блок відкритого тексту обмежується величиною відкритого ключа ($A<n$). Тому для досить довгих повідомлень потрібно кожен блок шифрувати окремо.

Приблизно такою ж обчислювальною складністю, як і факторизація, володіє операція дискретного логарифмування у скінченному полі, на якій ґрунтується асиметрична криптосистема Ель-Гамаля (Elgamal) [13-14]. Вона включає в себе алгоритм шифрування та алгоритм цифрового підпису.

Схема була запропонована Тахером Ель-Гамалем у 1985 році і являється одним із удосконалених варіантів алгоритму Діффі-Хеллмана, які використовуються для шифрування та забезпечення аутентифікації [14].

Алгоритм виконується в такій послідовності:

Крок перший – генерується випадкове просте число p;

Крок другий – вибирається ціле число g - первісний корінь p;

Крок третій – вибирається випадкове ціле число a таке, що $1<a<p-1$;

Крок п’ятий – обчислюється $h=g^a \bmod p$;

Крок шостий – відкритий ключ становлять три числа (p, g, h), таємний ключ-число a.

Повідомлення A, яке повинно бути менше від числа p, шифрується таким чином:

1) вибирається сесійний ключ – випадкове ціле число r таке, що $1<r<p-1$;

2) вираховуються числа $c_1=g^r \bmod p$ і $c_2=h^rA \bmod p$.

Пара чисел (c_1, c_2) є шифротекстом.

Одним з недоліків криптосистеми Ель-Гамаля є те, що довжина шифротексту вдвічі більша від вихідного повідомлення A.

Знаючи закритий ключ , вихідне повідомлення можна обчислити із шифртексту (c_1, c_2) за такою формулою:

$$A = c_2 (c_1^{-1})^a \bmod p \qquad (1.12)$$

При цьому дуже легко перевірити, що $(c_1^{-1})^a \bmod p=g^{-ra} \bmod p$ і тому $c_2(c_1^{-1})^a \bmod p=(h^rA)g^{-ra} \bmod p=(g^{ra}A)g^{-ra} \bmod p=A \bmod p$.

Для практичних обчислень більше підходить наступна формула:

$$A = c_2 (c_1^{-1})^a \bmod p = c_2 (c_1)^{p-1-a} \bmod p \qquad (1.13)$$

Схема шифрування на основі криптоалгоритму Ель-Гамаля представлена на рисунку 1.4.

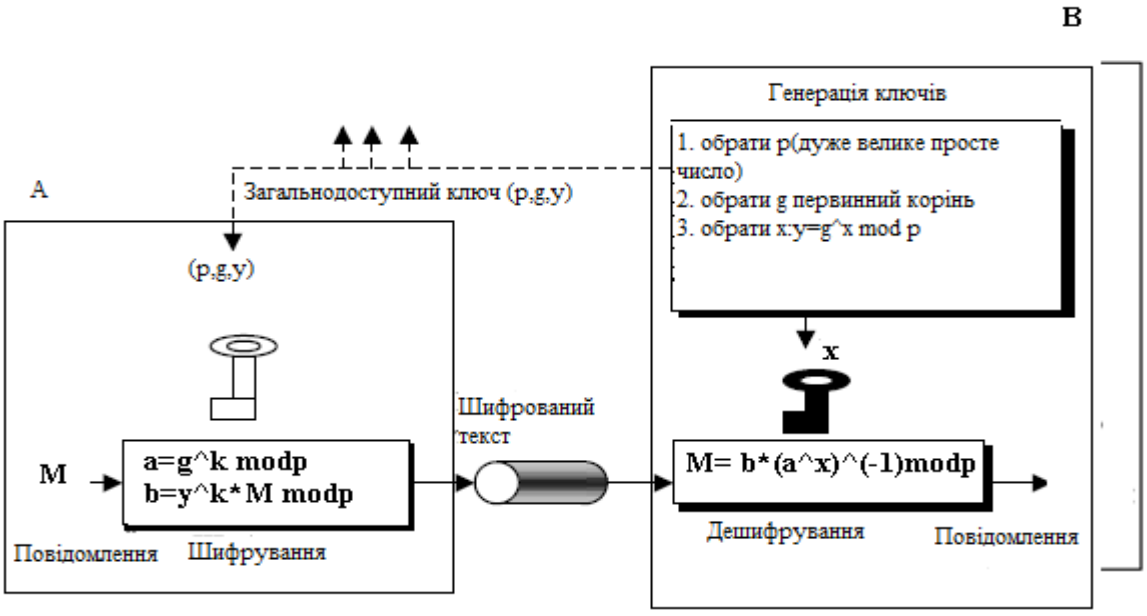


Рисунок 1.4 – Схема шифрування на основі криптоалгоритму Ель-Гамаля

Оскільки в схему Ель-Гамаля вводиться випадкова величина *r*, то шифр Ель-Гамаля можна назвати шифром багатозначної заміни. Через випадковість вибору числа *r* така схема ще називається схемою імовірнісного шифрування. Імовірнісний спосіб шифрування є перевагою для криптоалгоритму Ель-Гамаля, тому що у таких схемах спостерігається більша стійкість у порівнянні зі схемами із певним процесом шифрування [15].

Недоліком схеми шифрування Ель-Гамаля є подвоєння довжини зашифрованого тексту у порівнянні із початковим текстом. Для схеми імовірнісного шифрування саме повідомлення а ключ не визначать, однозначно, шифротекст. В схемі Ель-Гамаля необхідно використовувати різні значення випадкової величини щоб шифрування різних повідомлень . Якщо використовувати однакові , то для відповідних шифротекстів (c_1, c_2) та (c_1', c_2') виконується співвідношення $c_2(c_2')^{-1}= A(A')^{-1}$. З цього виразу можна легко обчислити *A*', якщо відомо .

На цей час криптосистеми із відкритим ключем вважаються найбільш перспективними. До них належить також схема Ель-Гамаля, криптостійкість якої ґрунтується на обчислювальній складності проблеми дискретного логарифмування. Крім того, існує велика кількість інших алгоритмів, які базуються на схемі Ель-Гамаля, зокрема: *DSA*, *ECDSA*, *KCDSA*, схема Шнорра та інші. Також заслуговують на увагу криптоалгоритми, котрі використовують арифметику еліптичних кривих, визначених над простими полями Галуа[19-21].

Однак усі операції у проаналізованих криптоалгоритмах відбуваються в ПСЧ (двійковій або десятковій системі числення), у яких відсутня можливість розпаралелення процесу обчислень.

1.2 Алгоритми пошуку залишку

У всіх асиметричних алгоритмах шифрування інформаційних потоків найважливішими операціями є модулярне множення та модулярне експоненціювання багаторозрядних чисел. Для підвищення ефективності їх обчислення багатьма авторами запропоновано різні підходи та алгоритми, які розроблені, в основному, з використанням десяткової системи числення, що значно збільшує час виконання програми. Одним з підходів щодо підвищення швидкодії знаходження результату модулярного множення є метод Карабуци, але він практично не використовується через складність його реалізації. В описано алгоритм Шенхаге – Штрассена з використанням швидкого перетворення Фур'є, який потребує $O(n_0 \log(n_0) \log(\log(n_0))$ бітових операцій [22].

В циклі робіт запропоновано матричні та векторні методи виконання основних арифметичних операцій для знаходження залишку, множення та піднесення до степеня на основі розмежованої двійкової системи числення.

Зокрема, для пошуку залишку $a \bmod p$ число a необхідно записати в двійковій системі числення $a=a_{n_0-1}2^{n_0-1}+ \ldots +a_i2^i+ \ldots +a_12+a_0$, де n_0 – розрядність числа a. Тоді:

$$a \bmod p = \left(\sum_{i=0}^{n_0-1} (a_i 2^i \bmod p)\right) \bmod p = \left(\sum_{i=0}^{n_0-1} (a_i a_{1i})\right) \bmod p \qquad (1.14)$$

де a_i=0 або 1, $a_{1i}=2^i \bmod p$.

З (1.14) випливає, що шуканий залишок дорівнюватиме сумі тих степенів двійки (або a_{1i}), для яких відповідно a_i=1, що проілюстровано в таблиці 1.1.

Таблиця 1.1 – Таблиця знаходження залишку $a \bmod p$

i	n_0-1	n_0-2		2	1	0
a_i	$a_{n}0$-1	$a_{n}0$-2	…	a_2	a_1	a_0
$2^i \bmod p$	$2^{n_0-1} \bmod p$	$2^{n_0-2} \bmod p$		$2^2 \bmod p$	$2^1 \bmod p$	$2^0 \bmod p$
a_{1i}	$a_{1\ n_0-1}$	$a_{1\ n_0-2}$	…	a_{12}	a_{11}	a_{10}

Слід зазначити також, що два послідовні значення a_{1i} та $a_{1\ i+1}$ пов'язані таким рекурентним співвідношенням:

$$a_{1i+1} = \{2 \cdot a_{1i}, 2 \cdot a_{1i}\, p \tag{1.15}$$

Отже, щоб знайти залишок за модулем в (1.15) не обов'язково виконувати обчислювально витратну операцію ділення з остачею, а можна обмежитися тільки відніманням. Ще, множення на 2 дуже просто реалізується за допомогою дописування нуля в кінці двійкового запису числа. В таблиці 1.2 наведено приклад пошуку 171 mod 31.

Таблиця 1.2 – Пошук залишку 171 mod 31

i	7	6	5	4	3	2	1	0
a_i	1	0	1	0	1	0	1	1
$2^i \bmod 31$	$2^7 \bmod 31$	$2^6 \bmod 31$	$2^5 \bmod 31$	$2^4 \bmod 31$	$2^3 \bmod 31$	$2^2 \bmod 31$	$2^1 \bmod 31$	$2^0 \bmod 31$
a_{1i}	4	2	1	16	8	4	2	1

Отже, 171 mod 31=(4+1+8+2+1) mod 31=16. Зазначимо, що результат отриманий є на основі додавання мало розрядних залишків степенів двійки за відповідним модулем.

Іншим напрямком підвищення швидко дії асиметричних криптосистем є використання нових підходів до виконання арифметичних операцій модулярного множення та експоненціювання.

В запропонована організація паралельного виконання модулярного експоненціювання і встановлено, що двопотоковий паралелізм при цьому є найбільш доцільною формою. Для практичної реалізації такої можливості

запропонована організація прискореного обчислення модулярної експоненти на двоядерному процесорі.

Теоретично та експериментально доведено, що розроблена організація забезпечує практично двократне прискорення виконання операції модулярного експоненціювання для розрядностей 2048 і 4096. Більш значне прискорення виконання модулярного експоненціювання може бути досягнуто при переході на рівень операцій процесорного множення та хмарних технологій [23].

В показано, що застосування СЗК в методі Монтгомері є ефективним способом збільшення швидкодії модулярного множення, проте його часова складність залишається високою при опрацюванні багаторозрядних чисел.

Незважаючи на існуючі ефективні алгоритми опрацювання інформаційних потоків у СЗК [24], що реалізуються на основі програмно-апаратних спецпроцесорів та забезпечують захист від помилок і певний рівень захисту інформації від несанкціонованого доступу, потрібно їх додаткове дослідження, аналіз і ефективне застосування нових методів та алгоритмів при застосуванні різних форм СЗК в асиметричних криптосистемах[25-26].

Для множення двох n-розрядних чисел $a = \sum_{i=0}^{n-1} a_i \cdot 2^i$ та $b = \sum_{j=0}^{n-1} b_j$, де a_i, b_j=0, 1, n-розрядність модуля p за допомогою векторно-модульного методу, потрібно побудувати два вектор-рядки, в першому з яких записуються елементи $c = 2^0 \bmod p$, $c = 2 \cdot c_{i-1} \bmod p$ другий - з a_i, як показано в таблиці 1.3

Таблиця 1.3 – Представлення вектор-рядків медулярного множення

c		c	…	c	c
a_{n-1}	…	a_j	…	a_1	a_0

Результатом модулярного множення двох n – розрядних чисел знаходиться згідно формули:

$$a \cdot b \bmod p = (\sum_{i=0}^{n-1} a_i \cdot c_i) \bmod p \qquad (1.16)$$

Розроблений метод характеризується меншою часовою та апаратною складністю порівняно з матрично-модульним.

1.3 Аналіз технічного завдання та постановка задачі

Метою даної роботи є розробка програмного рішення для аналізу та перевірки класичного та пропонованого вдосконалого методів медулярного експоненціювання. На рисунку 1.5 зображено дерево рішень розробки проекту.

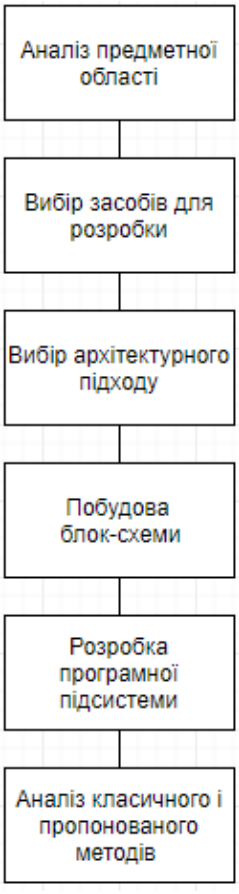


Рисунок 1.5 – Дерево рішень розробки проекту

Для досягнення поставленої мети потрібно вирішити ряд наступних завдань:

– проаналізувати передметну область;

– розробити математичну модель вдосконалого алгоритму модулярного експоненціювання;

– визначити набір інструментів для побудови програмного рішення;

– розробити блок-схему виконання роботи програмної підсистеми;

– реалізувати програмну підсистему для здійснення перевірки та працездатності класичного і пропонованого методів;

– провести ряд обчислень для отримання результатів;

– на основі отриманих даних побудувати графіки залежностей;

– проаналізувати роботу програмного рішення та методів обчислень за графіками;

– зробити висновок з отриманих результатів.

2 АЛГОРИТМИ МОДУЛЯРНОГО ЕКПОНЕНЦІЮВАННЯ

2.1 Алгоритми модулярного множення

Нехай маємо два числа із розрядністю n_0: $a=a_{n_0-1}2^{n_0-1}+\ldots+a_i2^i+\ldots+a_12+a_0$ та $b=b_{n_0-1}2^{n_0-1}+\ldots+b_i2^j+\ldots+b_12+b_0$, де a_i, $b_j=0$ або 1. Для знаходження результату їх множення за модулем p потрібно побудувати матрицю, представлену в таблиці 2.1, де $c_{ij}=2^{i+j} \bmod p$. Тоді добуток чисел a та b за модулем p отримувався згідно такої формули:

$$a \cdot b \bmod p = (\sum_{i=0}^{n_0-1}(\sum_{j=1}^{n_0-1} a_i b_j)c_{ij}) \bmod p = (\sum_{i=1}^{n_0-1}(\sum_{j=1}^{n_0-1} a_i b_j)2^{i+j} \bmod p) \bmod p \quad (2.1)$$

Це означає, що шуканий результат отримується у вигляді суми за модулем p тих c_{ij}, для яких відповідні a_i та b_j дорівнюють 1.

Таблиця 2.1 – Матриця для модулярного множення

	b_{n_0-1}	…	b_j	…	b_1	b_0
a_{n_0-1}	$c_{n_0-1\ n_0-1}$	…	$c_{n_0-1\ j}$	…	$c_{n_0-1\ 1}$	$c_{n_0-1\ 0}$
…	…	…	…	…	…	…
a_i	$c_{i\ n_0-1}$	…	c_{ij}	…	c_{i1}	c_{i0}
…	…	…	…	…	…	…
a_1	$c_{1\ n_0-1}$	…	c_{1j}	…	c_{11}	c_{10}
a_0	$c_{0\ n_0-1}$	…	c_{0j}	…	c_{01}	c_{00}

Слід зазначити, що $c_{ij} = c_{ji}$ і аналогічно до попереднього випадку кожне наступне значення c_{ij} визначається за допомогою рекурентних співвідношень:

$$a_{ij+1} = \{2 \cdot c_{ij}, 2 \cdot c_{ij} < p; a_{i+1\,j} = \{2 \cdot c_{ij}, 2 \cdot c_{ij} < p \quad (2.2)$$

Враховуючи, що $25_{10}=11001_2$ та $21_{10}=10101_2$ і побудувавши відповідну матрицю, можна побачити, що 25·21 mod 29=(24+12+16+6+3+4+16+8+1) mod 29= 90 mod 29=3.

Таблиця 2.2 – Матриця для модулярного множення 25*21 mod 29

25 / 21	1	1	0	0	1
1	24	12	6	3	16
0	12	6	3	16	8
1	6	3	16	8	4
0	3	16	8	4	2
1	16	8	4	2	1

Даний метод дозволяє замінити операцію множення, яка має квадратичну обчислювальну складність $O1(n_0) = n_0^2$, матрично-модульною операцією сумування, яка характеризується лінійно-логарифмічною складністю $O2(n_0)=n_0 \log n_0$

Для удосконалення даного методу можна побудувати матрицю, представлену в таблиці 2.3, де $a_{1j}=2^j \bmod p$.

Таблиця 2.3 – Матриця для удосконаленого матричного методу модулярного множення

	$a_{1\,2n_0-1}$	…	$a_{1\,n_0+1}$	$a_{1\,n_0}$	…	$a_{1\,i}$	…	a_{11}	a_{10}
b_{n_0-1}	a_{n_0}	…	a_1	a_0	…	…	…	0	0
…	…	…	…	…	…	…	…	0	0
b_j	…	…	…	…	…	…	…	0	0
…	0	…	…	…	…	…	…	0	0
b_1	0	…	a_{n_0}	a_{n_0-1}	…	a_{i-1}	…	a_0	0
b_0	0	…	0	a_{n_0}	…	a_i	…	a_1	a_0

Множення відбувається таким чином:

$$a \cdot b \bmod p = a_0 b_0 a_{10} + (a_0 b_1 + a_1 b_0) a_{11} + (a_0 b_2 + a_1 b_1 + a_2 b_0) a_{12} + ... + \\ + (a_0 b_{n_0-1} + a_1 b_{n_0-2} + ... + a_{n_0-1} b_0) \cdot a_{1 \cdot 2 n_0 - 1} \quad (2.3)$$

Введемо позначення:

$$A_l = (a_0 b_{n_0-1} + a_1 b_{n_0-2} + ... + a_{n_0-1} b_0) a_{1l} \quad (2.3)$$

Тоді

$$a \cdot b \bmod p = (\sum_{l}^{2n_0-1} A_l) \bmod p \quad (2.4)$$

З врахуванням (2.3) співвідношення (2.4) набуде такого вигляду:

$$a \cdot b \bmod p = (\sum_{l}^{2n_0-1} A_l \mid n_0 - 1 b_j) a_{1l} \bmod p \quad (2.5)$$

де $i+j=n_0-1$.

Причому, якщо $b_j=1$ та деякі a_s і $a_q=1$, то можна спростити обчислення за допомогою наступного співвідношення:

$$(a_s+a_q)a_{1i}= a_{1\,i+1}. \quad (2.6)$$

В таблиці 2.4 наведено приклад вдосконаленого матричного методу для пошуку значення виразу 21*25 mod 29. Знову числа 21 і 25 представляються в двійковій системі числення і на основі таблиці 2.5 будується відповідна матриця.

З урахуванням співвідношень (2.5) та (2.6) одержуємо

$$(24+12+6+3+16+16+8+4+1)\bmod 29=(19+8+4+1)\bmod 29=3. \quad (2.7)$$

Таблиця 2.4 – Приклад множення 21*25 mod 29

	19	24	12	6	3	16	8	4	2	1
1	0	1	0	1	0	1	0	0	0	0
1	0	0	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0
0	0	0	0	0	1	0	1	0	1	0
1	0	0	0	0	0	1	0	1	0	1

Таким чином, отримано удосконалений новий метод заміни операції множення, яка має квадратичну обчислювальну складність $O1\cdot(n_0)=n_0^2$, матричномодульною операцією сумування з лінійно-логарифмічною складністю $O3=\frac{1}{2}n_0\log_2 n_0$

Отже, при використанні виразу (2.7) обчислювальна складність виконання операції модулярного множення суттєво зменшується, що дозволяє ефективно використовувати запропонований метод в алгоритмах захисту інформаційних потоків, побудованих на асиметричних криптосистемах.

Для зменшення об’єму пам’яті, в якій мають зберігатися проміжні результати матричних обчислень, можна використати векторно-модульний метод модулярного множення чисел $a\cdot=\sum_{i=0}^{n_0-1}a_i\cdot 2^i$ та $b\cdot=\sum_{j=0}^{n_0-1}b_i\cdot 2^j$. В цьому випадку будується два вектор-рядки (c_i та a_i), перший з яких складається з елементів $c=2\cdot c_{i-1}\bmod p, c=2^0\cdot b\bmod p$ другий - з a_i (таблиця 2.5).

Таблиця 2.5 – Представлення вектор-рядків модульного множення

i	n_0-1	…	2	1	0
c	c	…	c	c	c
a_i	a_n0-1	…	a_2	a_1	a_0

Слід зазначити, що кожне наступне значення c_i обчислюється за рекурентною формулою, аналогічною:

$$c_{i+1} = \{2 \cdot c_i, 2 \cdot c_i < p \tag{2.8}$$

Результат модулярного множення двох чисел отримується згідно такої формули:

$$a \cdot b \bmod p = (\sum_{i=0}^{n_0-1} a_i \cdot c_i) \bmod p \tag{2.9}$$

тобто відбувається сумування тих c_i, для яких відповідні a_i дорівнюють 1.

В таблиці 2.6 наведено приклад векторно-модульного методу модулярного множення для пошуку значення виразу 21*25 mod 29. Число 21 представляється в двійковій системі числення: $21_{10}=10101_2$ і на основі таблиці 2.6 будується відповідна матриця.

Таблиця 2.6 – Приклад множення 21*25 mod 29 векторно-модульним методом

i	4	3	2	1	0
a_i	1	0	1	0	1
c_i	2*26mod29=23	2*13mod29=26	2*21mod29=13	2*25mod29=21	2^0*25mod29=25

Отже, 21*25 mod 29=(23+13+25) mod 29=3.

Розроблений метод характеризується меншою часовою та апаратною складністю порівняно з матрично-модульним за рахунок зменшення кількості операцій додавання з $2 \cdot \log_2 n_2$ до $\log_2 n_0$ тобто в два рази. При вирішенні задач криптографії збільшення швидкодії в два рази є суттєвим і значно розширює функціональні можливості апаратного забезпечення, а також спрощує реалізацію відповідних спец процесорів, процесів реалізації програмних рішень для втілення використання алгоритмів у реальних проектах.

2.2 Алгоритми модулярного експоненціювання

Для модулярного експоненіювання $a^x \bmod p$ (вважається, що $x \leq \varphi(p)$, $\varphi(p)$ – значеня функції Ейлера від модуля p) потрібно використати проміжну матрицю, представлену в таблиці 2.7. Її розмірність дорівнює розрядності n_0 модуля p. В стовбцях матриці записані величини $A_i = a^{2^i} \bmod p$ у двійковій системі числення, тобто a_{ij}=0, 1. Тоді будь–який степінь числа a записується за степенями двійки і шуканий результат можна отримати, перемноживши значення у стовбцях, для яких відповідні x_i у розкладі $x = \sum_{i=0}^{n_0-1} x_i \cdot 2^i$ дорівнюють одиниці, за допомогою такого виразу:

$$a^x \bmod p = \left(\prod_{i=0}^{n_0-1} a^{x_i 2^i}\right) \bmod p = \prod_{i=0}^{n_0-1} \left(\left(\sum_{j=0}^{n_0-1} a_{ij} 2^j\right)^{x_i}\right) \bmod p \qquad (2.10)$$

де a_{ij} – біти двійкового запису числа $a^{2i} \bmod p = \sum_{j=0}^{n_0-1} a_{ij} 2^j$

Таблиця 2.7 – Матриця піднесення до степеня

x_{n_0-1}	…	x_i	…	x_1	x_0
$a_{n_0-1\ n_0-1}$	…	$a_{i\ n_0-1}$	…	$a_{1\ n_0-1}$	$a_{0\ n_0-1}$
…	…	…	…	…	…
$a_{n_0-1\ j}$	…	$a_{i\ j}$	…	$a_{1\ j}$	$a_{0\ j}$
…	…	…	…	…	…
$a_{n_0-1\ 1}$	…	$a_{i\ 1}$	…	$a_{1\ 1}$	$a_{0\ 1}$
$a_{n_0-1\ 0}$	…	$a_{i\ 0}$	…	$a_{1\ 0}$	$a_{0\ 0}$
$a^{2^{n_{i_0}-1}} \bmod p$	…	$a^{2^i} \bmod p$	…	$a^{2^1} \bmod p$	$a^{2^0} \bmod p$

Основними перевагами даного методу - це здійснення операцій над залишками, а не із великими числами, які дозволяють пришвидшити алгоритм модулярного експоненціювання. Слід зазначити, що для заповнення матриці доцільно скористатися таким рекурентним співвідношенням:

$$a^{2^{i+1}} \bmod p = (a^{2^i} \bmod p)^2 \bmod p\,. \qquad (2.11)$$

В таблиці 2.8 наведено приклад пошуку значення $23^{19} \bmod 29$ на основі матричного методу модулярного експоненціювання.

Таблиця 2.8 – Приклад пошуку значення $23^{19} \bmod 29$ на основі матричного методу модулярного експоненціювання

1	0	0	1	1
19				
0	1	1	0	1
0	0	0	0	0
1	1	1	1	1
1	1	0	1	1
1	1	0	1	1
7	23	20	7	23
$23^{24} \bmod 29$	$23^{23} \bmod 29$	$23^{22} \bmod 29$	$23^{21} \bmod 29$	$23^{20} \bmod 29$

Отже, 23^{19} mod 29=(7·7·23) mod 29=25. Операцію множення можна виконувати методами, описаними в п. 2.2.

2.3 Векторно-модульний метод модулярного експоненціювання

Для зменшення матриці, представленої у таблиці 2.9, доцільно використати векторно-модульний метод модулярного експоненціювання, у якому не визначаються біти двійкового запису чисел A_i, яке записується в десятковому вигляді (таблиця 2.9).

В таблиці 2.10 наведено приклад використання даного методу для пошуку 23^{19} mod 29.

Таблиця 2.9 – Матриця векторно-модульного методу модулярного експоненціювання

i	n_0-1	…	2	1	0
x_i	x_n0_{-1}	…	x_2	x_1	x_0
$a^{2^i} \mathrm{mod} p$	$a^{2^{n_0-1}} \mathrm{mod} p$	⋯	$a^{2^2} \mathrm{mod} p$	$a^{2^1} \mathrm{mod} p$	$a^{2^0} \mathrm{mod} p$
A_i	A_n0_{-1}	…	A_2	A_1	A_0

Таблиця 2.10 – Приклад пошуку 23^{19} mod 29 векторно-модульним методом

i	4	3	2	1	0
x_i	1	0	0	1	1
$a^{2^i} \mathrm{mod} p$	$a^{2^4} \mathrm{mod} 29$	$23^{2^3} \mathrm{mod} 29$	$23^{2^2} \mathrm{mod} 29$	$23^{2^1} \mathrm{mod} 29$	$23^{2^0} \mathrm{mod} 29$
A_i	7	23	20	7	23

Звідси слідує – 23^{19} mod 29= (7·7·23) mod 29=25.

Розрахунки демонструють, що запропонований алгоритм піднесення до степеня двійкового числа будь-якої розрядності за модулем p дозволяє зменшити складність з $O(n_0^3)$, або $O1(n_0^2 \log_2 n_0)$ (Монтгомері метод) до

$O2(\frac{n_0^2 \log_2 n_0}{4})$, тобто ефективність зростає в 4 рази.

На БР.КСМ.07125/15.00.00.000 А2 зображено алгоритм класичного методу у вигляді блок-схеми.

Як ми можемо спостерігати, даний алгоритм містить досить багато розгалужень для корегування обчислення, що призводить до його складності реалізації. Корегування обчислень необхідні для захисту від помилок.

На БР.КСМ.07125/15.00.00.001 А2 зображено алгоритм роботи запропонованого методу у вигляді блок схеми. Як ми спостерігаємо, в запропонованому алгоритмі використано менше корегувальних елементів обчислення, що дозволяє спростити його реалізацію, відсутня велика кількість перевірок, що дозволяє обчислювальній системі використовувати менше ресурсів для обчислень. Так як ми зберігаємо не всі результати, а тільки беремо до уваги ті, що по ітерації з бінарного ряду числа *X*, де одиниця співпадає, а змінна яка нам потрібна для наступних ітерацій, тимчасово міститиме значення і буде перезаписана.

Звідси випливає доцільність використання запропонованого методу в асиметричних криптографічних алгоритмах захисту інформації для зменшення складності обчислень, при генеруванні ключів, шифруванні/дешифруванні повідомлень, спрощує можливість реалізації і зменшує кількість корегувань обчислень що досить може навантажити систему обробки інформації, запису та зчитування.

Для простішого пояснення виконання роботи програмної підсистеми потрібно розробити UML діаграму послідовностей, що довзолить простіше зрозуміти виконання роботи, що зображено на рисунку 2.1.

Виконання роботи над алгоритмом здійснюється за наступною послідовністю:

1) потрібно підібрати відповідні дані для дієздатності виконання роботи методів;

Рисунок 2.1 – UML діаграма послідовностей

2) підібрані дані вносяться у програмну підсистему для обробки

3) програмна підсистема аналізує введені дані та викликає функції, що дозволяють конвертувати дані для зручного обчислення даних

4) за обраним користувачем методом, здійснюється обробка даних

5) отримані результати виводяться користувачу для здійснення подальшого використання.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Обгрунтування вибору мови програмування Java

Java – це об’єктно орієнтованою мовою програмування випущена в 1995 році компанією Sun Microsystem. У 2009 році компанію викуплено компанією Oracle. Код написаний на Java компілюється в байт-код, що підчас виконяння інтерпретується віртуальною машиною для потрфбної платформи. Віртуальною машиною що запускає байт-код згенерованих мовою Java називається Java Virtual Machine (JVM) [33].

Синтаксис мови запозичений із таких мов як C і C++. За основу взяли об’єктну модель *C++* і модифікували. Були усунуті конфліктні ситуації які міг допустити розробник при розробці програми, такі як унаслідування одного батьківського класу, очищення пам’яті тепер перекладено на JVM, де не потрібно вручну описувати дистуктори і звільнення пам’яті.

Java Virtual Machine(JVM) – це пакет комп’ютерних програм і структур даних, які використовують модель віртуальної машини щоб виконанти інші комп’ютерні програми. Віртуальна машина працює із байт-кодом який можк бути згенерованим з інших мов програмування. JVM є основним компонентом Java платформ, доступна для сучасних платформ, тому про програми, що скомпільовані у Java байт-код запускаються на різних платформах.

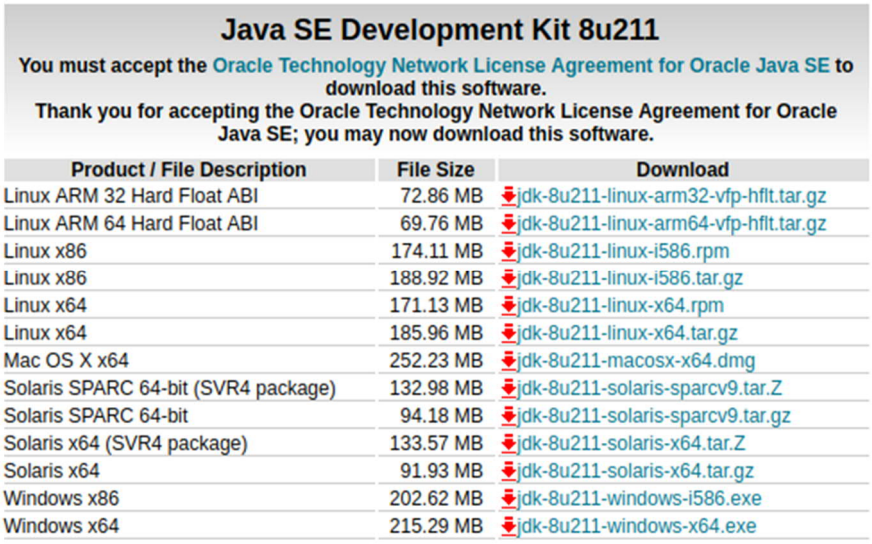
Байт-код – це машинно-незалежний код низького рівня, що генерується транслятором та виконується інтерпретатором. Більшість інструкцій еквіваленті одній або кільком командам асамблера. Трансляція в байт-код відбувається проміжним положенням між компіляцією у машинний код та інтерпретацією.

Програма на байт-коді зазвичай виконується за допомогою робті інтерпретатора із байт-кодом. Перевагою є мобільність, тобто один і той же байт-код може працювати на різних платформах та архітектурах. Оскільки

байт-код зазвичай менше абстрагований, компактний та більше схожий до машиного коду ніж початковий, байт-коду зазвичай ефективніший, ніж чиста інтерпретація початкового коду, призначеного для корегування людиною.

3.2 Програмне вирішення задачі

Розробка програмного рішення здійснюється вибором інструментів. Для роботи на мові програмування Java нам потрібно встановити Java Development Kit (на далі JDK) із основного сайту компанії Oracle,як це показано на рисунку 3.1, що є повноцінним пакетом для розробки.



Java SE Development Kit 8u211

You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.
Thank you for accepting the Oracle Technology Network License Agreement for Oracle Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.86 MB	jdk-8u211-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.76 MB	jdk-8u211-linux-arm64-vfp-hflt.tar.gz
Linux x86	174.11 MB	jdk-8u211-linux-i586.rpm
Linux x86	188.92 MB	jdk-8u211-linux-i586.tar.gz
Linux x64	171.13 MB	jdk-8u211-linux-x64.rpm
Linux x64	185.96 MB	jdk-8u211-linux-x64.tar.gz
Mac OS X x64	252.23 MB	jdk-8u211-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	132.98 MB	jdk-8u211-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	94.18 MB	jdk-8u211-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	133.57 MB	jdk-8u211-solaris-x64.tar.Z
Solaris x64	91.93 MB	jdk-8u211-solaris-x64.tar.gz
Windows x86	202.62 MB	jdk-8u211-windows-i586.exe
Windows x64	215.29 MB	jdk-8u211-windows-x64.exe

Рисунок 3.1 – Вибір потрібного JDK

Після встановлення основних пакетів, також потрібне і середовище, у якому зручно розробляти, тому хорошим вибором буде InteliJ IDEA від компанії JetBrains(див. рис. 3.2).

Рисунок 3.2 – Середовище розробки InteliJ IDEA

Наступним кроком нам потрібно організувати архітектурний підхід до розробки програмного рішення, блок-схема якого представлена на БР.КСМ.07125/15.00.00.002.А2. Вибором архітектурного підходу буде шаблон проектування Model View Presenter (надалі MVP), що є досить зручним для організації роботи програм, вигляд побудови архітектурного підходу зображено на рисунку 3.3.

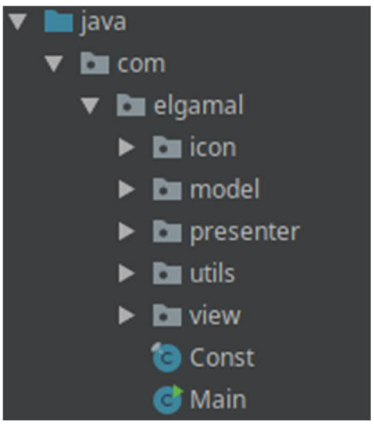


Рисунок 3.3 – Архітектурна структура проекту

Далі ми описуємо клас який відповідатиме за побудову та здійснення роботи графічного інтерфейсу, він міститиме в собі конструктор, що

відповідатиме за генерування зображення і підключатиме клас реалізатора логіки виконання програми, згідно архітектурного підходу(див. рисунок 3.4).

```
public MainFrame() {

    mPresenter = new MainPresenterImpl( view: this);

    mPanel = new JPanel();
    mPanel.setVisible(true);
    mPanel.setLayout(null);

    initMenu();
    initControllers();
    initLabels();
    initTextInputAreas();
    initProgressBar();
    invisibleLoadingButtons();

    mPanel.setBackground(Color.WHITE);
    mPanel.setSize(Const.PANEL_WIDTH, Const.PANEL_HEIGHT);
    mPanel.setBackground(Color.decode(Const.LIGHT_BLUE));
    setTitle("El Gamal");
    setIconImage(new ImageIcon( filename: "com/elgamal/icon/logo_icon.png").getImage());
    getContentPane().add(mPanel);
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    setSize(Const.PANEL_WIDTH, Const.PANEL_HEIGHT);
    setResizable(false);
    setVisible(true);
}
```

Рисунок 3.4 – Основна функція налаштування графічного інтерфейсу

Наступним кроком буде побудова класів та функцій, що реалізовуватимуть класичний і запропонований метод для здійснення однакових умов перевірок працездатності.

```
public class OldWay {
    public static String encrypt(BigInteger keyX, BigInteger keyP, BigInteger keyQ) {
        BigInteger keyY = modPowFunction2(keyQ, keyX, keyP);
        return keyY.toString();
    }
    private static BigInteger modPowFunction2(BigInteger keyQ, BigInteger keyX, BigInteger keyP) {
        BigInteger q1 = new BigInteger( val: "1");
        BigInteger q2 = new BigInteger( val: "1");
        BigInteger zero = new BigInteger( val: "0");
        BigInteger one = new BigInteger( val: "1");
        BigInteger two = new BigInteger( val: "2");
        int exponent = 2;
        if (keyX.equals(new BigInteger( val: "0"))) { return new BigInteger( val: "1"); }
        if (keyX.equals(new BigInteger( val: "1"))) { return keyQ.mod(keyP); }
        if (!keyX.mod(two).equals(BigInteger.ZERO)) {
            q1 = q1.multiply(keyQ);
            q2 = q2.multiply(keyQ);
            q2 = q2.pow(exponent);
            q2 = q2.mod(keyP);
            keyX = keyX.subtract(one).divide(two);
        } else {
            q2 = q2.multiply(keyQ).pow(exponent).mod(keyP);
            keyX = keyX.divide(two);
        }
        while (keyX.compareTo(zero) > 0) {
            if (keyX.equals(BigInteger.ONE)) { break; }
            if (!keyX.mod(two).equals(BigInteger.ZERO)) {
                if (!q1.equals(q2)) { q1 = q1.multiply(q2).mod(keyP); }
                q2 = q2.pow(exponent);
                q2 = q2.mod(keyP);
                keyX = keyX.subtract(one).divide(two);
            } else {
                q2 = q2.modPow(two, keyP);
                if (!keyX.equals(two)) { keyX = keyX.divide(two); }
                else { break; }
            }
        }
        return q1.multiply(q2).mod(keyP);
    }
}
```

Рисунок 3.5 – Реалізація класичного методу

На рисунку 3.5 зображено код реалізації класичного методу, на рисунку 3.6 – реалізацію пропонованого.

```
public class NewWay {
    public static String encrypt(BigInteger keyX, BigInteger keyP, BigInteger keyQ) {
        char[] decimalKeyXRadixArray = radixCharArray(keyX.toString( radix: 2), keyP.toString( radix: 2).length());
        List<BigInteger> iterationOneY = firstPart(decimalKeyXRadixArray, keyP, keyQ);
        BigInteger keyY = new BigInteger( val: "1");
        for (BigInteger item : iterationOneY) {
            keyY = keyY.multiply(item);
        }
        keyY = keyY.mod(keyP);
        return keyY.toString();
    }

    private static char[] radixCharArray(String keyXInBinary, int keyPDecimalLength) {
        if (keyXInBinary.length() != keyPDecimalLength) {
            StringBuilder stringBuilder = new StringBuilder();
            for (int j = 0; j < keyPDecimalLength - keyXInBinary.length(); j++) {
                stringBuilder.append('0');
            }
            stringBuilder.append(keyXInBinary);
            keyXInBinary = stringBuilder.toString();
        }
        keyXInBinary = new StringBuilder(keyXInBinary).reverse().toString();
        return keyXInBinary.toCharArray();
    }

    private static List<BigInteger> firstPart(char[] charRadixArray, BigInteger keyP, BigInteger keyQ) {
        List<BigInteger> tempResultArray = new ArrayList<>();
        BigInteger tempResult = keyQ;
        for (int i = 0; i < charRadixArray.length; i++) {
            if (i == 0) {
                tempResult = tempResult.pow(1).mod(keyP);
            } else {
                tempResult = (tempResult.pow(2)).mod(keyP);
            }
            if (charRadixArray[i] != '0') {
                tempResultArray.add(tempResult);
            }
        }
        return tempResultArray;
    }
}
```

Рисунок 3.6 – Реалізація запропонованого методу

Надалі описується ряд кодів, що відповідатимуть реалізації варіацій перевірки працездатності старого і пропонованого методу для створення однакових умов і запускається проект. На рисунку 3.7 зображено головне вікно програми.

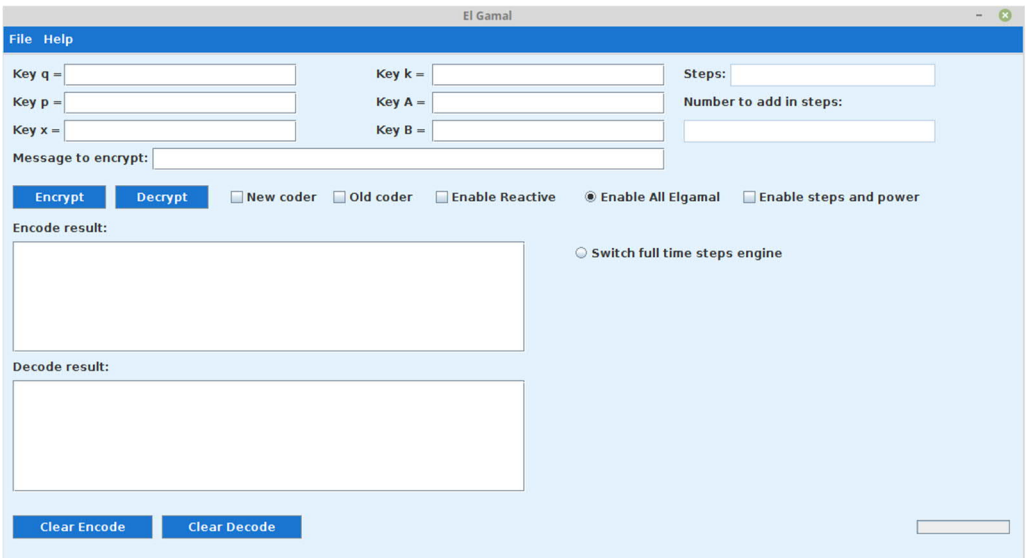


Рисунок 3.7 – Головне вікно програми

Тепер, маючи графічний інтерфейс і реалізовану логіку виконання, здійснимо наступний ряд дій, для перевірки працездатності реалізованого програмного рішення.

Згідно загальної формули медулярного експоненціювання $y = q^x \bmod p$ здійснюється запис у відповідні поля, де приклад запису зображено на рисунку 3.8.

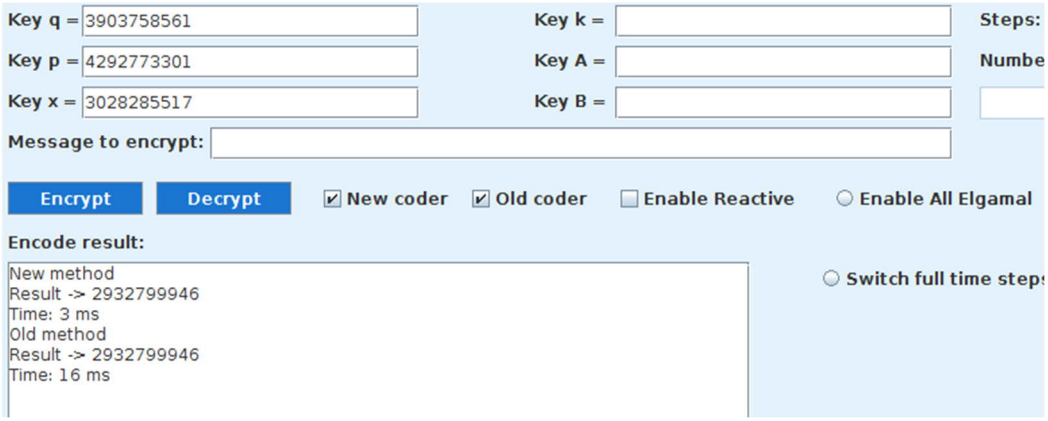


Рисунок 3.8 – Внесення даних у поля та виведення результату

Після внесення даних у поля, вибору методів обчислень за допомогою встановлення “пташки”, натискаємо кнопку “Encrypt” після чого, отриманий результат обчислення виводиться у поле “Encrypt result”.

Отримані результати можна експортувати в файли із розширенням *Excel*, які зберігатимуться після експорту в папці розташування програми, як це зображено на рисунку 3.9.

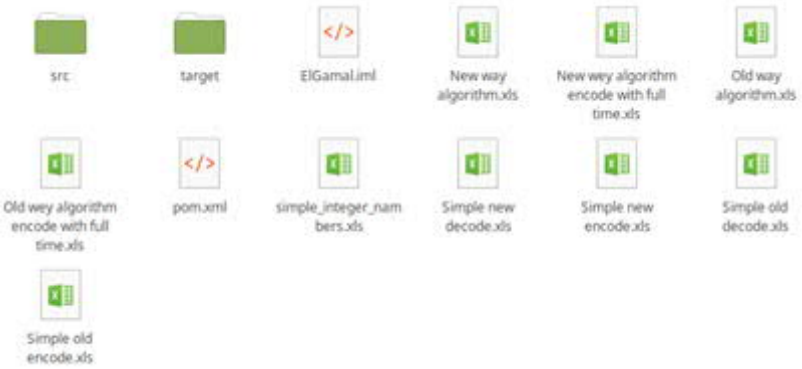


Рисунок 3.9 – Розташування екпортованих результатів

Так як отримані дані екпортовані у Excel, то по отриманих даних легко побудувати графік і порівняти результат.

3.3 Приклад модулярного експоненціювання для використання в криптосистемах RSA та Ель-Гамал.

Щоб зрозуміти перевагу пропонованого методу ми вирішимо наступний ряд обчислень для порівняння. Для дослідження обиралися числа розрядностей по зростанню від 32 бітних і до 62 бітних чисел.

Здійснимо обчислення де показник степеня змінюватиметься за правилом 2^k-1

Дано такі вхідні дані:

1) p=18446744073709551557

2) q = 18446744073709551

3) x=2^k-1, k – розрядність числа

З результату обчислення, яке зображено на рисунку 3.10 запропонований метод, після зростання числа розрядності показника степеня, є набагато швидшим за класичний спосіб.

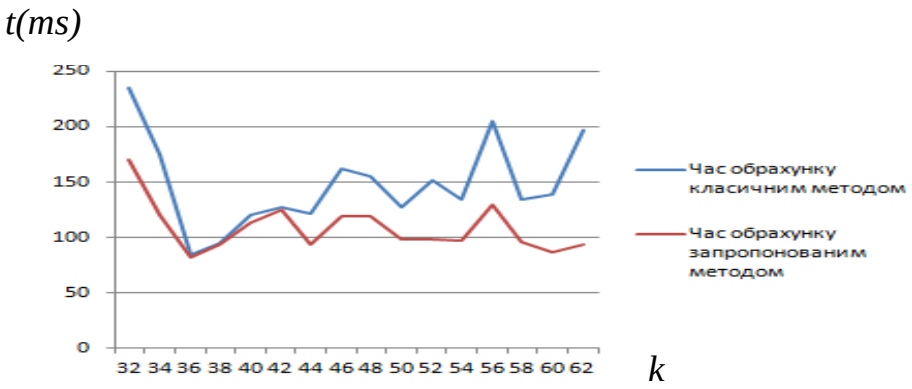


Рисунок 3.10 – Часові характеристики обчислення класичним та запропонованим методами, де x=2^k-1

Здійснимо обчислення де показник степеня змінюватиметься за правилом 2^k+1.

Дано такі вхідні дані:

1) p=18446744073709551557

2) q=18446744073709551

3) $x=2^k+1$, k – розрядність числа

У результаті обчислення, яке є зображено на рисунку 3.11 запропонований метод, після зростання числа розрядності показника степеня, є також набагато швидшим за класичний спосіб.

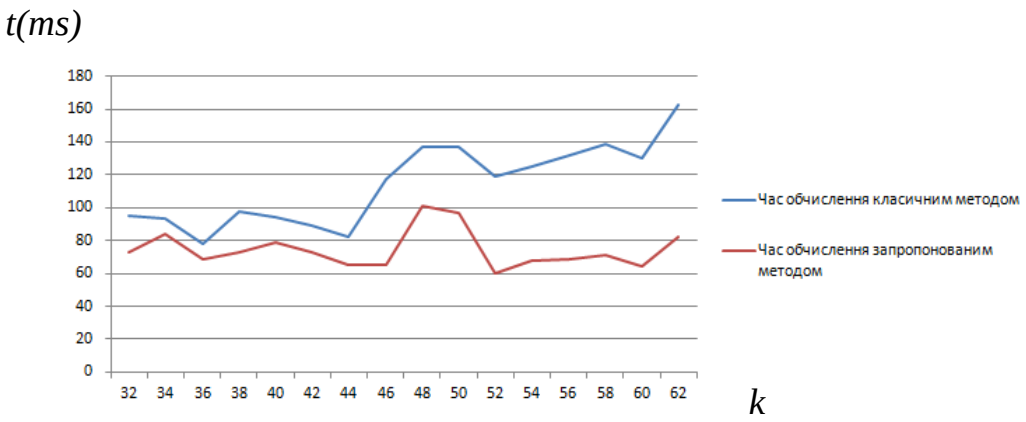


Рисунок 3.11 – Часові характеристики обчислення класичним та запропонованим методами, де $x=2^k+1$

Здійснимо обчислення де показник степеня змінюватиметься за правилом $(k/2)+1$.

Дано такі вхідні дані:

1) p =18446744073709551557

2) q = 18446744073709551

3) $x = (k/2)+1$, k – розрядність числа

В даному результаті, що на рисунку 3.12 також демонструє перевагу запропонованого методу.

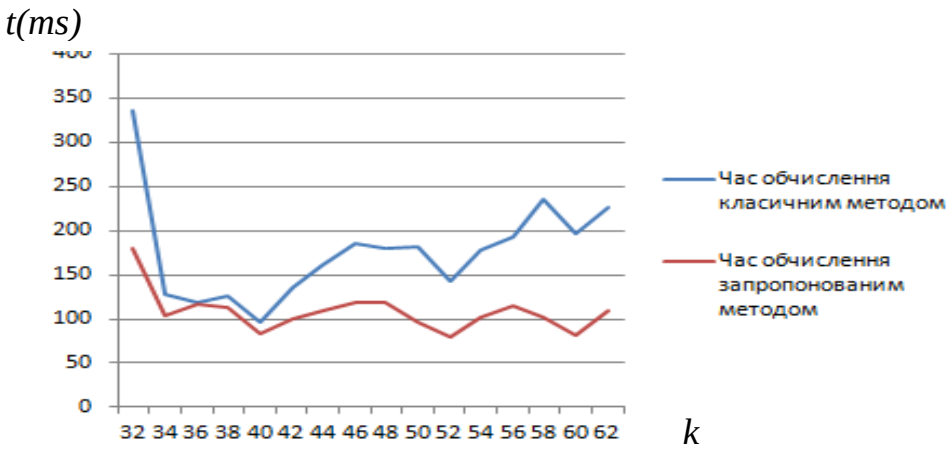


Рисунок 3.12 – Часові характеристики обчислення класичним та запропонованим методами, де $x=(k/2)+1$

Здійснимо обчислення де показник степеня змінюватиметься за правилом, де для числового показника степеня використовується Вага Хемінга.

Дано такі дані:

1) p = 18446744073709551557

2) q = 18446744073709551

3) k – розрядність показника

Результат з рисунку 2.4 демонструє досить велику різницю в часовій характеристиці роботи методів.

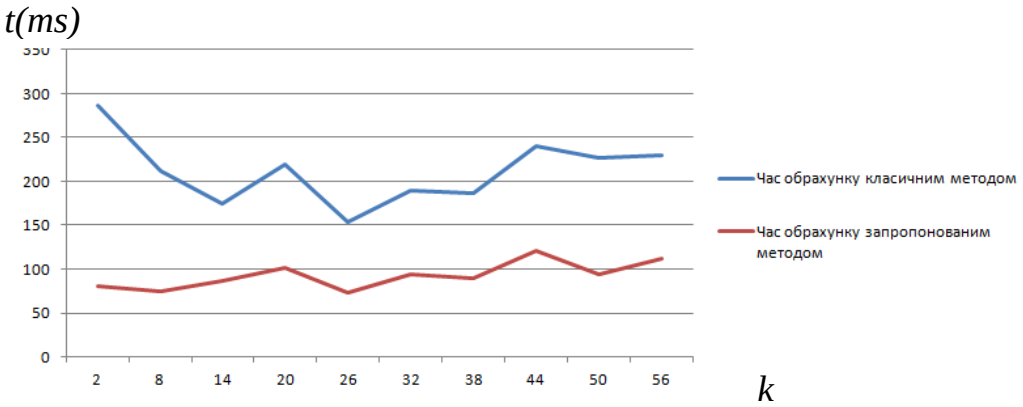


Рисунок 2.4 – Часові характеристики обчислення класичним та запропонованим методами

Отже, взявши до уваги всі отримані результати обчислень ми можемо зробити висновок, що запропонований метод є набагато ефективнішим для використання його у різноманітних криптосистемах. Пропонований метод дозволив продемонструвати свою швидкодію при різній розрядності показника степеня, що є вагомим для обчислення, навантаження та й в загальному розробки самого програмного рішення. Так як під час розробки двох методів, виникли ряд проблем для корегування обчислень. Щоб класичний метод давав потрібний результат, доводиться додавати значну кількість коригувальних перевірок, що навантажує обчислювальний процес і процес розробки, натомість пропонований метод потребував перевірки на значення розрядності.

4 ТЕХНІКО-ЕКОНОМІЧНИЙ РОЗДІЛ

Метою техніко – економічного розділу бакалаврської роботи є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності програмної підсистеми модулярного експоненціювання для асиметричних криптосистем та прийняття рішення про його подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки. Для проведення даного дослідження необхідно провести ряд розразунків.

4.1 Розрахунок витрат на розробку програмного додатку

Витрати на розробку і впровадження програмної підсистеми модулярного експоненціювання для асиметричних криптосистем (K) включають:

$$K = K_1 + K_2,$$

де K_1 – витрати на розробку програмного забезпечення грн.;

K_2 – витрати на відлагодження і дослідну експлуатацію програми рішення задачі на комп’ютері, грн.

Витрати на розробку програмних засобів включають:

- витрати на оплату праці розробників ($B_{ОП}$);
- витрати на відрахування у спеціальні державні фонди ($B_ф$);
- витрати на матеріали та комплектуючі ($П_в$);
- накладні витрати (H);
- інші витрати ($I_в$)

– витрати на використання комп’ютерної техніки($B_{КТ}$).

Витрати на оплату праці включають заробітну плату (ЗП) всіх категорій працівників, безпосередньо зайнятих на всіх етапах проектування. Розмір ЗП обчислюється на основі трудоємності відповідних робіт у людино-днях та середньої ЗП відповідних категорій працівників.

У розробці проектного рішення задіяні наступні спеціалісти-розробники, а саме: керівник проекту; студент-дипломант; консультант техніко-економічного розділу (таблиця 4.1).

Таблиця 4.1 – Вихідні дані для розрахунку витрат на оплату праці

№п/п	Посада виконавців	Місячний оклад, грн.
1	Керівник ДП, викладач	7293
2	Консультант техніко-економічного розділу, доцент	7293
3	Студент	1400

Витрати на оплату праці розробників проекту визначаються за формулою, що наведена нище (4.1):

$$B_{ОП} = \sum_{i=1}^{N} \sum_{j=1}^{M} n_{ij} \cdot t_{ij} \cdot C_{ij} \ , \qquad (4.1)$$

де n_{ij} – чисельність розробників i-ої спеціальності j-го тарифного розряду, осіб;

t_{ij} – затрачений час на розробку проекту співробітником i-ої спеціальності j-го тарифного розряду, год;

C_{ij} – годинна ставка працівника i-ої спеціальності j-го тарифного розряду, грн..

Середньо годинна ставка працівника може бути розрахована за формулою:

$$C_{ij} = \frac{C_{ij}^{0}(1+h)}{РЧ_i}, \qquad (4.2)$$

де C_{ij} – основна місячна заробітна плата розробника *i*-ої спеціальності *j*-го тарифного розряду, грн.;

h – коефіцієнт, що визначає розмір додаткової заробітної плати (при умові наявності доплат);

$РЧ_i$ – місячний фонд робочого часу працівника *i*-ої спеціальності *j*-го тарифного розряду, год. (приймаємо 168 год.).

Коефіцієнт *h*, який визначає розмір додаткової заробітної плати, який включає для керівника 0,25 та консультанта техніко-економічного розділу дорівнює 0,47.

Результати розрахунку записують до таблиці 4.2.

Таблиця 4.2 – Розрахунок витрат на оплату праці

№ п/п	Посада виконавців	Час розробки, год.	Погодинна заробітна плата, грн/год.	Витрати на розробку, грн.
1	Керівник ДП, старший викладач	16	63,8	1020,8
2	Консультант техніко-економічного розділу, доцент	2	63,8	127,6
3	Студент	144	8,33	1199,52
Разом				2347,92

Величну відрахувань у спеціальні державні фонди визначають у відсотковому співвідношенні від суми основної та додаткової заробітних плат. Згідно діючого нормативного законодавства сума відрахувань у спеціальні державні фонди складає 20,5% від суми заробітної плати:

$$B_{Ф} = \frac{20,5}{100} \cdot 2347,92 = 481,32 \text{ грн.}$$

Загальна сума витрат на матеріальні ресурси (B_M) визначається за формулою (4.3):

$$B_M = \sum_{i=1}^{n} K_i \cdot Ц_i, \qquad (4.3)$$

де K_i – витрата i-го типу матеріалу, натуральні одиниці вимірювання;

$Ц_i$ – ціна за одиницю i-го типу матеріалу, грн.;

i – тип матеріального ресурсу;

n – кількість типів матеріальних ресурсів.

Таблиця 4.3 – Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Факт. витрачено матеріалів	Ціна за одиницю, грн.	Сума, грн	Транспортні витрати (10% від суми)	Загальна сума, грн
1	Допоміжна література	шт	1	600	600	60	660
2	Папір (формат А4)	уп	2	80	160	16	176
3	Ручка кулькова	шт	2	10	20	2	22
4	Олівець простий	шт	2	10	20	2	22
5	Диски CD-R	шт	2	15	30	3	33
6	Зошит, 96 арк	шт	1	50	50	5	55
7	Тонер для принтера	уп	1	90	90	9	99
8	Канцелярські маркери	шт	2	20	40	4	44
	Р а з о м						1111,00

Витрати на використання комп'ютерної техніки($B_{КТ}$) включають витрати на амортизацію комп'ютерної техніки, витрати на користування програмним забезпеченням, витрати на електроенергію, що споживається комп'ютером. За даними обчислювального центру ТНЕУ для комп'ютера

типу IBM PC/ATX вартість години роботи становить 6 грн. Середній щоденний час роботи на комп'ютері – 2 години.

Розрахунок витрат на використання комп'ютерної техніки приведений в таблиці 4.4.

Таблиця 4.4 – Розрахунок витрат на використання комп'ютерної техніки

№ п/п	Назва етапів робіт, при виконанні яких використовується комп'ютер	Час використання комп'ютера, год.	Витрати на використання комп'ютера грн.
1	Проведення досліджень та оформлення їх результатів	60	360
2	Оформлення техніко-економічного розділу	8	48
3	Оформлення БР	12	72
Разом		80	480

Накладні витрати проектних організацій включають три групи видатків: витрати на управління, загальногосподарські витрати, невиробничі витрати. Вони розраховуються за встановленими відсотками до витрат на оплату праці. Середньостатистичний відсоток накладних витрат приймемо 150% від заробітної плати:

$$H = 1{,}5 \cdot 2347{,}92 = 3521{,}88 \text{ грн.}$$

Інші витрати є витратами, які не враховані в попередніх статтях. Вони становлять 10% від заробітної плати:

$$I_B = 2347{,}92 \cdot 0{,}1 = 234{,}79 \text{ грн.}$$

Витрати на розробку програмного забезпечення складають:

$$K_1 = B_{ОП} + B_Ф + B_M + H + I_B + B_{КТ},$$

K_1 = 2347,92+481,32+1111,00+3521,88+234,79+480,00=8176,91 грн.

Витрати на відлагодження і дослідну експлуатацію програмного продукту визначаємо за формулою (4.4):

$$K_2 = S_{м.г.} \cdot t_{від} \qquad (4.4)$$

де $S_{м.г.}$ – вартість однієї машино-години роботи ПК, грн./год;

$t_{від}$ – комп'ютерний час, витрачений на відлагодження і дослідну експлуатацію створеного програмного продукту, год.

Загальна кількість днів роботи на комп'ютері дорівнює 30 днів. Середній щоденний час роботи на комп'ютері – 2 години. Вартість години роботи комп'ютера дорівнює 6 грн., тому K_2= 6·60 = 360 грн.

4.2 Визначення експлуатаційних витрат

Для оцінки економічної ефективності проектованого програмного засобу слід порівняти його з аналогом, тобто існуючим програмним забезпеченням ідентичного функціонального призначення.

Експлуатаційні одноразові витрати по програмному забезпеченню та аналогу включають вартість підготовки даних і вартість роботи комп'ютера (за час дії програми):

$$E_П = E_{1П} + E_{2П},$$

де $E_П$ – одноразові експлуатаційні витрати на ПЗ (аналог), грн.;

$E_{1П}$ – вартість підготовки даних для експлуатації ПЗ (аналогу), грн.;

$E_{2П}$ – вартість роботи комп'ютера для виконання проектного рішення (аналогу), грн.

Річні експлуатаційні витрати $B_{ЕП}$ визначаються за формулою:

$$B_{ЕП} = E_{П} * N_{П},$$

де $N_{П}$ – періодичність експлуатації ПЗ (аналогу), раз/рік.

Вартість підготовки даних для роботи на комп'ютері визначається за формулою:

$$E_{1П} = \sum_{l=1}^{n} n_i t_i c_i,$$

де i – категорії працівників, які приймають участь у підготовці даних (i=1,2,...n);

n_i – кількість працівників i-ої категорії, осіб.;

t_i – трудомісткість роботи співробітників i-ої категорії по підготовці даних, год.;

c_i – середнього годинна ставка працівника i-ої категорії з врахуванням додаткової заробітної плати, що знаходиться із співвідношення:

$$c_i = \frac{c_i^0 (1+b)}{m},$$

де c_i^0 – основна місячна заробітна плата працівника i-ої категорії, грн.;

b – коефіцієнт, який враховує додаткову заробітну плату (приймемо 0,57);

m – кількість робочих годин у місяці, год.

Для роботи з даними як для проектного рішення так і аналогу потрібен один працівник, основна місячна заробітна плата якого складає: c = 4173 грн.

Тоді:

$$c_1 = \frac{4173(1+0{,}57)}{22*8} = \frac{6551{,}61}{176} = 37{,}23\,\text{грн/год.}$$

Трудомісткість підготовки даних для проектного рішення складає 1 год., для аналога 1,5 год.

Таблиця 4.5 – Розрахунок витрат на підготовку даних та реалізацію проектного рішення на комп'ютері

№	Час роботи співробітників, год.	Середньогодинна заробітна плата, грн./год.	Витрати , грн.
	Проектне рішення		
1	1	37,23	37,23
	Аналог		
1	1,5	37,23	55,85

Витрати на експлуатацію комп'ютера визначається за формулою:

$$E_{2П} = t * S_{МГ}.$$

де t – витрати машинного часу для реалізації рішення (аналогу), год.;

$S_{МГ}$ – вартість однієї години роботи комп'ютера, грн./год.

$E_{1П}$ = 1* 6 = 6 грн.; $E_{1А}$=1,5*6 = 9 грн.

$E_{2П}$=37,23+6=43,23 грн. ; $E_{2А}$=55,85+9 = 64,85 грн.

$B_{ЕП}$= 43,23*252=10893,96 грн. ; $B_{ЕА}$=64,85 *252=16342,2 грн.

Накладні витрати пов'язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 60–100 % від суми основної та додаткової заробітної плати працівників.

$$H_B = 0{,}7 * B_{ОП}\,, \tag{4.7}$$

де H_B – накладні витрати.

$$H_B=0{,}7*6551{,}61=4586{,}13 \text{ грн.}$$

Складання кошторису витрат та визначення собівартості. Результати проведених розрахунків зведемо у таблицю 4.6.

Таблиця 4.6 – Кошторис витрат

№ п/п	Найменування витрат	Сума витрат, грн.
1	Витрати на оплату праці ($B_{оп}$)	2347,92
2	Відрахування у спеціальні державні фонди ($B_ф$)	481,32
3	Витрати на матеріали та комплектуючі ($B_м$)	1111,00
4	Накладні витрати на розробку (H)	3521,88
5	Інші витрати ($I_в$)	234,79
6	Витрати на відлагодження і дослідну експлуатацію програмного продукту	480
7	Накладні витрати на експлуатацію ($H_в$)	4586,13
8	Річні експлуатаційні витрати	16342,2
Разом		29105,24

Договірна ціна ($Ц_Д$) для проектних рішень розраховується за формулою, що неведена нище (4.8):

$$Ц_Д = B_{КС} \cdot \left(1 + \frac{p}{100}\right), \tag{4.8}$$

де $B_{КС}$ – кошторисна вартість, грн.;

p – середній рівень рентабельності, % (приймаємо 27% за погодженням з керівником).

$$Ц_Д = 29105{,}24 \cdot (1+0{,}27) = 36963{,}65 \text{ грн.}$$

4.3 Визначення економічної ефективності і терміну окупності капітальних вкладень

Економічна ефективність (E_P) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E_P = \frac{П}{B_{КС}}\ , \qquad (4.9)$$

де $П$ – прибуток, грн.;

$B_{КС}$ – кошторисна вартість, грн..

$$Ep = 5684{,}6 \text{ грн.} / 36963{,}65 \text{ грн.} = 0{,}2.$$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень (Tp):

$$T_P = \frac{1}{E_P}\ . \qquad (4.10)$$

Тобто:

$$T_P = 1/0{,}2 = 5\text{р.}$$

Прийнятним вважається термін окупності близький до 7 років. Розраховані економічні показники проекту занесемо до таблиці 4.7.

Таблиця 4.7 – Економічні показники розробки

№ п/п	Показник	Значення
1.	Собівартість, грн.	29105,24
2.	Плановий прибуток, грн.	5684,6
3.	Ціна, грн.	33891,83
4.	Економічна ефективність	0,2
5.	Термін окупності, рік	5

Враховуючи основі економічні показники з таблиці 4.7, можна зробити висновок, що при економічній ефективності 0,2 та терміні окупності – 5 роки проводити роботи по впровадженню даного програмного є доцільним та економічно вигідним.

ВИСНОВКИ

1. Відбувся аналіз існуючих криптосистем, їх недоліки та переваги.

2. На основі теоретичної бази пропонованого методу асиметричного медулярного експоненціювання зроблено аналітичний приклад із дослідженням дієздатності алгоритмічного підходу.

3. Беручи до основи розглянуто алгоритмічний підхід до вирішення завданння

4. Побудовано блоксхему класичного та пропонованого методу із врахуванням корекції частини, щоб виконання алгоритму видавав очікуваний результат, без помилок.

4. На основі розробленого алгоритмічного забезпечення побудовано блок-схему програмної підсистеми, що дозволило визначити основні копоненти що потрібно внести для розробки програмного рішення, щоб дослідити роботу класичного і пропонованого методів.

5. На основі блок-схеми розроблено програмний засіб для дослідження працездатності методів.

6. Здійснено дослідження методів за допомогою розробленого програмного засобу.

7. Отримані результати які були експортовані у вихідний файл проаналізовано та побудовано графіки.

8. Дослідження показало, що пропонований метод є набагато ефективнішим, як у розробці, так і у швидкодійності обчислення, на відмінну класичного, що потребую нагромадженої кількості додаткових перевірок, для корегування обчислення, що також має і вплив на його швидкодію.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Якименко І.З., Касянчук М.М., Кінах Я.І., Власюк І.М., Суслін В.В. Удосконалення реалізації асиметричних криптоалгоритмів на основі системи залишкових класів. Матеріали VI Всеукраїнської школи-семінару молодих вчених і студентів «Сучасні комп'ютерні інформаційні технології» (АСІТ), 2018. С. 79.

2. Касянчук М.М., Якименко І.З., Івасьєв С.В., Момотюк О.В. Експериментальне дослідження програмної реалізації методів пошуку оберненого елемента за модулем. Інформатика та математичні методи в моделюванні. 2017. С. 178.

3. Романец Ю.В., Тимофеев П.А., Шаньгин В.Ф. Защита информации в компьютерных системах и сетях. М.: Радио и связь, 2001. 376 с.

4. Воробьев И.В. Защита информации в персональных ЭВМ. М.: Мир, 2008. 312 с.

5. Мафтик С. Механизмы защиты в компьютерных сетях. М.: Мир, 2013. 219 с.

6. Ростовцев А.Г., Маховенко Е.Б. Теоретическая криптография. М.: Профессионал, 2005. 480 с.

7. Рябко Б.Я., Фионов С.В. Криптографические методы защиты информации: Учебное пособие для вузов. М.: Телеком, 2005. 229 с.

8. Панасенко С.П. Алгоритмы шифрования. М.: Академический Проект, 2006. 529 с.

9. Петров В.П., Петров С.В. Информационная безопасность человека и общества. М.: ЭНАС, 2007. 334 с.

10. Шнаер Б. Практическая криптография. М.: Вильямс, 2007. 424 с.

11. Задірака В.К., Олексюк О.С. Комп'ютерна криптологія. Тернопіль, Київ, 2002. 504 с.

12. Вербіцький О.В. Вступ до криптології. Львів: ВНТЛ, 1998. 247 с.

13. Иванов И.И. Криптология. М.: Высшая школа, 2004. 230 с.

14. Винокуров А.Ю. Блочные алгоритмы. М.: Монитор, 2005. 132 с.

15. Голубицкая Е.А., Жигульская Г.М. Экономика связи. М.: Радио и связь, 2003. 318 с.

16. Айерленд К. Классическое введение в современную теорию чисел. М.: Мир, 2007. 416 с.

17. Акушский И.Я., Юдицкий Д.М. Машинная арифметика в остаточных классах. М.: Сов. радио, 1968. 440с.

18. Вариченко Л.В. Абстрактные алгебраические системы и цифровая обработка сигналов. К.: Наука думка, 2006. 247 с.

19. Сидельников В.М. Криптография и теория кодирования. М.: ФИЗМАТЛИТ, 2008. 324 с.

20. Амербаев В.М. Теоретические основы машинной арифметики. Алма-Ата: Наука, 2006. 324 с.

21. Ворона В.А., Тихонов В.А. Системы контроля и управления доступом. М.: Горячая линия - Телеком, 2010. 272 с.

22. Деднев М.А., Дыльнов Д.В. Защита информации в банковском деле и электронном бизнесе. М.: КУДИЦ ОБРАЗ, 2004. 321 с.

23. Казарин О.В. Безопасность программного обеспечения компьютерных систем. М.: Высшая школа, 2013. 243 с.

24. Ярочкин В.И. Информационная безопасность. М.: Академический Проект, 2008. 544 с.

25. Барсуков В.С. Безопасность: технологии, средства, услуги. М.: КУДИЦ-ОБРАЗ, 2001. 496 с.

26. Белов Е.Б., Лось В.П., Мещеряков Р.В. Основы информационной безопасности. М.: Горячая линия - Телеком, 2006. 544 с.

27. Краснобаев В.А. Методы повышения надежности специализированных ЭВМ систем и средств связи. Харьков: ХВВКИУ РВ, 2000. 172с.

28. Ковалевский В. Криптографические методы. М.:Компьютер Пресс, 2013. 312 с.

29. Севастьянова Б.А. Совершенные шифры. М.: Гелиос АРВ, 2003. 160 с.

30. Методичні рекомендації до виконання дипломного проекту з освітньо-кваліфікаційного рівня “Бакалавр” напряму підготовки 6.050102 «Комп’ютерна інженерія» фахового спрямування «Комп’ютерні системи та мережі» / О.М. Березький, Л.О.Дубчак, Р.Б. Трембач, Г.М. Мельник, Ю.М. Батько, С.В. Івасьєв / Під ред. О.М. Березького. - Тернопіль: ТНЕУ, 2016.–65 с.

31. Об’єктно орієнтована мова програмування *Java*: веб-сайт URL: https://uk.wikipedia.org/wiki/JAVA (дата звернення 5.02.2019)

32. Методичні вказівки до написання техніко-економічного розділу для дипломних проектів на здобуття освітньо-кваліфікаційного рівня “Бакалавр” напряму підготовки 6.050102 «Комп’ютерна інженерія» / І.Р.Паздрій. - Тернопіль: ТНЕУ, 2015.– 36 с.

33. Методичні вказівники до оформлення курсових проектів, звіт про проходження практики, випускних кваліфікаційних робіт Студентів спеціальності «Клмп’ютерна інженерія» / І.В. Гураль, Л.О. Дубчак /
Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019.– 33с.