НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
УКРАЇНИ "КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО"
Факультет інформатики та обчислювальної

техніки Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
Сергій СТІРЕНКО
"__"_____2023 р.
(підпис)

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою "Комп'ютерні системи та мережі"

спеціальності 123 "Комп'ютерна інженерія"

на тему: *Метод прискореного модулярного піднесення до квадрату*
Виконала: студентка 4 курсу, групи ІВ-93
(шифр групи)
Островська Богдана Валеріївна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н. Марковський О.П. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль)ст.викл Виноградов Ю.М _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент декан ФПМ, д.т.н., проф. Дичка І.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ "КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО"

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

"Комп'ютерні системи та мережі"

спеціальності 123 "Комп'ютерна інженерія"

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
"__" _____2023 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Островської Богдани Валеріївни

1. Тема проєкту *Метод прискореного модулярного піднесення до квадрату*
керівник проєкту Марковський Олександр Петрович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від *31.05.2023.*
2. Термін здачі студентом закінченого проєкту *08.06.2023*
3. Вихідні дані до проєкту *див. технічне завдання*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються) *Аналіз існуючих методів модулярного піднесення до квадрату та дослідження можливостей їх реалізації. Розробка та дослідження методу прискореного модулярного піднесення до квадрату. Розробка програмних засобів для тестування дослідження методу прискореного модулярного піднесення до квадрату.*
5. Перелік графічного матеріалу (назви креслень з точним позначенням обов'язкових креслень) *електрична схема двопотокового обчислювача модулярної експоненти (структурна схема), блок-схема алгоритму прискореного модулярного піднесення до квадрату (принципова схема), блоку модулярного піднесення до квадрату(функціональна схема)*

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	ст.викл Виноградов Ю.М		

7. Дата видачі завдання *2 6 . 0 9 . 2 0 2 2*

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми* проєкту	*26.09.2022-15.12.2022*	
2.	*Вивчення та аналіз завдання*	*15.12.2022-15.03.2023*	
4.	*Розробка алгоритму прискореного модулярного піднесення до квадрату*	*15.03.2023-25.03.2023*	
5.	*Програмна реалізація системи*	*25.03.2023-20.05.2023*	
6.	*Оформлення пояснювальної записки*	*5.05.2023-10.06.2023*	
7.	*Захист програмного продукту*	*26.05.2023*	
8.	*Передзахист*	*10.06.2023*	
9.	*Захист*	*22.06.2023*	

Студент-дипломник _____

(підпис)

Керівник проєкту _____

(підпис)

Анотація

В бакалаврському проекті розроблено та теоретично обґрунтовано метод прискорення модулярного піднесення до квадрату, який базується на комбінації кількох технік, зокрема суміщення в часі множення на різні розряди множника та групової редукції Монтгомері з передобчисленням. Експериментально доведено, що цей метод забезпечує значне прискорення виконання мультиплікативних операцій модулярної арифметики. Зокрема, виявлено, що швидкість виконання базових операцій, які широко використовуються в сучасних алгоритмах криптографічного захисту даних, зростає в 3-4 рази.

Abstract

This paper proposes a new method for accelerating modular exponentiation, based on a combination of several techniques, including time interleaving of multiplication by different digits of the multiplier and grouped Montgomery reduction with precomputation. Experimental results demonstrate that this method significantly speeds up the execution of multiplicative operations in modular arithmetic. Specifically, it has been found that the speed of executing basic operations widely used in modern cryptographic data protection algorithms increases by 3-4 times.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екзем-пляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.468243.001 ОА***	Метод прискореного модулярного	1		
			піднесення до квадрату			
			Опис альбому			
	А4	***ІАЛЦ.468243.002 ТЗ***	Метод прискореного модулярного	5		
			піднесення до квадрату			
			Технічне завдання			
	А4	***ІАЛЦ.468243.003 ПЗ***	Метод прискореного модулярного	63		
			піднесення до квадрату			
			Пояснювальна записка			
	А1	***ІАЛЦ.468243.004 Е1***	Метод прискореного модулярного	1		
			піднесення до квадрату			
			електрична схема двопотокового			
			обчислювача модулярної експоненти			
			(структурна схема)			
	А1	***ІАЛЦ.468243.005 Д1***	Метод прискореного модулярного	1		
			піднесення до квадрату			
			блок-схема алгоритму			
			прискореного модулярного			
			піднесення до квадрату			
			(принципова схема)			
	А1	***ІАЛЦ.468243.006 Е2***	Метод прискореного модулярного	1		
			піднесення до квадрату			
			блоку модулярного піднесення до			
			квадрату(функціональна схема)			
	А4	***ІАЛЦ.468243.007 Д2***	Метод прискореного модулярного	8		
			піднесення до квадрату			
			Текст програмного коду			

Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	***ІАЛЦ.468243.001 ОА***			
Розроб		Островська Б. В.			*Метод прискореного модулярного піднесення до квадрату* Опис альбому	Літ.	Аркуш	Аркушів
Перев		Марковський О.П.					1	1
Н. контр.		Виноградов Ю.М				***КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-93***		
Затв.		Стіренко С.						

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ
на тему: «Метод прискореного модулярного піднесення до квадрату»

Київ – 2023 р.

ЗМІСТ

Стр.

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ 2
2. ПІДСТАВИ ДЛЯ РОЗРОБКИ 2
3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ 2
4. ДЖЕРЕЛА РОЗРОБКИ 3
5. ТЕХНІЧНІ ВИМОГИ 3
5.1 Вимоги до продукту, що розробляється 3
5.2 Вимоги до програмного забезпечення 4
5.3. Вимоги до апаратної частини 4
6. ЕТАПИ РОЗРОБКИ 5

					ІАЛЦ.468243.002 ТЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Островська Б.В			*Метод прискореного модулярного піднесення до квадрату* Технічне завдання	Літ.	Аркуш 1 / Аркушів 5
Перевірив		Марковський О.П					
Н. Контр.		Виноградов				КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-93	
Затвердив		Стіренко С.					

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання поширюється на розробку методу та програмних засобів прискореного модулярного піднесення до квадрату чисел, довжина яких значно перевищує розрядність процесора.

Область застосування – засоби захисту інформації в системах віддаленого моніторингу стану об’єктів реального світу та комп'ютерного дистанційного управління ними з використанням Інтернету в якості середовища обміну даних.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проєкту за освітньо-професійною програмою “Комп’ютерні системи та мережі” спеціальності 123 “Комп’ютерна інженерія”, затверджене кафедрою Обчислювальної техніки Національного технічного Університету України “Київський Політехнічний інститут імені Ігоря Сікорського”.

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Мета проекту полягає в прискоренні програмної реалізації широкого кола алгоритмів криптографічного захисту інформації, в основі яких лежить операція модулярного експоненціювання, за рахунок виключення операційної надмірності при обчисленні операції модулярного піднесення до квадрату. Розробка призначена для підвищення ефективності систем комп’ютерного дистанційного управління об’єктами реального світу з використанням Інтернету в якості середовища обміну даними за рахунок прискорення реалізації в таких системах функцій захисту інформації.

4. ДЖЕРЕЛА РОЗРОБКИ

4.1 Ostrovska. B Method of accelerated modular multiplication with Montgomeri group reduction / I. Boyarshin, O. Markovskyi, B. Ostrovska // Proceeding of International Conference Security, Fault Tolerance, Intelligence. ICSFTI-2022,- Kyiv.- P.40-45.

4.2 Ostrovska. B Organization of parallel execution of modular multiplication to speed up the computational implementation of public-key cryptography / I. Boyarshin, O. Markovskyi, B. Ostrovska //Information, Computing and Intelligent systems. – 2022. – № 3. – P. 77.

4.3 Кабір Л.А. Метод прискорення модулярного множення за технологією Монтгомері / Л.А. Кабір, О.В. Русанова, І.О. Гуменюк //Альманах науки № 1(52).- 2022.- С.44-46.

4.4 Трибунська К.Є. Метод прискорення модулярного множення з використанням групової редукції /К.Є. Трибунська // Актуальні питання розвитку науки та освіти: матеріали М Міжнародної науково-практичної конференції м.Львів, 30-31 березня 2022 року.-Львів: Львівський науковий форум, 2022.- С.38- 46.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до продукту, що розробляється

5.1.1 Програмні засоби повинні реалізувати операції модулярного піднесення до квадрату з застосуванням редукції Монтгомері. Розрядність операндів, над якими здійснюється операція модулярного піднесення до квадрату має гнучко змінюватися від 1024 до максимального значення – 4096.

5.1.2 Програмні засоби повинні реалізувати операцію модулярного піднесення до квадрату за час, що не перевищує 0.1 мс. на процесорі Intel з тактовою частотою 3.6 GHz. Граничний об’єм пам'яті таблиць

передобчислень не має перевищувати 65535 байт

5.1.3 Для забезпечення високої швидкості програмної реалізації операції модулярного піднесення до квадрату в програмі має суміщуватися формування суми часткових добутків та модулярної редукції за методом Монтгомері. Для підвищення швидкодії потрібно застосувати групову редукцію на основі попередньо виконаних передобчислень. При цьому об'єм таблиці передобчислень має гнучко змінюватися для забезпечення можливості експериментального підбору оптимального значення.

5.1.4 Програмні засоби мають реалізувати функціональне, часове та статистичне моделювання методу прискореного модулярного піднесення до квадрату задля визначення показників швидкодії та проведення об'єктивного порівняльного аналізу.

5.1.5 Програма повинна працювати в режимі автоматичної генерації чисел, над якими здійснюється модулярне піднесення до квадрату і в режимі ручного інтерактивного їх задання з використанням символьних рядків.

5.2 Вимоги до програмного забезпечення

- Операційна система Windows 10
- Мова програмування C++11

5.3. Вимоги до апаратної частини

- Процесор рівня Intel i5 і вище.
- Оперативна пам'ять не менше 500 МБ.
- Вільне місце на жорсткому диску не менше 100 МБ

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення та аналіз завдання	26.09.2022-02.12.2022
Створення та узгодження технічного завдання	02.12.2022-15.12.2022
Вивчення літературних джерел	15.12.2022-15.03.2023
Розробка алгоритму шифрування	15.03.2023-25.03.2023
Розробка програмної моделі	25.03.2023-20.04.2023
Відлагодження програми та виправлення помилок	20.04.2023-5.05.2023
Оформлення документації дипломного проєкту	5.05.2023-10.06.2023

Пояснювальна записка
до дипломного проєкту
на тему «Метод прискореного модулярного піднесення до квадрату»

Київ – 2023 р.

ЗМІСТ

Стр.

ВСТУП 4

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ТЕХНОЛОГІЙ МОДУЛЯРНОГО МНОЖЕННЯ ЯК БАЗОВОЇ СКЛАДОВОЇ КРИТПОГРАФІЧНИХ АЛГОРИТМІВ З ВІДКРИТИМ КЛЮЧЕМ 6

1.1 Аналіз вимог до технологій модулярного піднесення до квадрату в сучасних системах комп'ютерного керування 6

1.2 Огляд методів прискорення операції модулярного множення 8

1.3 Аналіз можливостей прискорення програмної реалізації модулярного піднесення до квадрату 10

Висновки до розділу 1 19

РОЗДІЛ 2
РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДУ ПРИСКОРЕННЯ МОДУЛЯРНОГО ПІДНЕСЕННЯ ДО КВАДРАТУ 21

2.1 Аналіз особливостей обчислювальної реалізації операції модулярного піднесення до квадрату 22

2.2 Метод прискореного модулярного піднесення до квадрату на базі редукції Монтгомері 27

2.3 Організація розпаралелювання операції модулярного піднесення до квадрату на багатоядерних обчислювальних платформах 35

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ТЕСТУВАННЯ ТА ДОСЛІДЖЕННЯ МЕТОДУ ПРИСКОРЕНОГО МОДУЛЯРНОГО ПІДНЕСЕННЯ ДО КВАДРАТУ 46

3.1 Організація даних 47
3.2 Розробка функцій і методів 48
3.3 Експериментальне дослідження запропонованого методу прискореного модулярного піднесення до квадрату з використанням розробленої програми 50
Висновки до розділу 3 58
ВИСНОВКИ 60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64
ДОДАТОК 1 2
ДОДАТОК 2 2
ДОДАТОК 3 2
ДОДАТОК 4 0

ВСТУП

Процес модульного піднесення до степеня, який виконується над числами, що значно перевищують розрядність процесора, є фундаментальним для широкого спектру криптографічних алгоритмів, які базуються на аналітично невирішуваних проблемах теорії чисел, зважаючи на це, він незмінно займає чільне місце серед ряду актуальних досліджень протягом останнього десятиліття. Він забезпечується операцією, що ґрунтується на особливій організації розбиття компонентів модулярного множення на незалежні обчислювальні процеси. Зокрема, ця операція є основою обчислення для реалізації RSA, EL-Gamal, стандарту цифрового підпису DSA, схеми FESIS строгого ідентифікування віднесених користувачів, шо забезпечує цим технологіям криптографічний захист даних. Збільшення розрядності процесора, зважаючи на кубічну залежність між розрахунковою складністю алгоритмів та ємністю, кардинально знизить швидкість виконання пов'язаних протоколів.

Рівень захисту криптографічних механізмів безпеки даних, які базуються на операції модульного множення, повністю визначається кількістю розрядів модуля. На сьогоднішній день 2048 біт вистачає для більшості практичних задач, що передбачають захист даних за допомогою криптографічних алгоритмів з відкритим ключем. Проте, технології продовжують розвиватись і суттєво підвищувати складність, що дає можливість зловмисникам віддалено використовувати обчислювальні потужності у власних корисних потребах.

У той же час аналіз динаміки розвитку прикладних задач, в яких криптографічні механізми захисту даних з відкритим ключем практично використовуються, вказує на те, що значна частина з них виконується в реальному часі і диктує нагальність потреби у швидкій реалізації відповідних обчислень[4]. Ще одна ключова особливість використання модульної арифметики на сучасному етапі розвитку криптографії з відкритим ключем - збільшення розрядності чисел, що використовуються в реальних практичних

задачах. Динаміка розвитку технологій хмарних обчислень потенційно надає зловмисникам можливість віддаленого доступу до великої обчислювальної потужності, яка може використовуватися для зламу криптографічних механізмів захисту. Це каталізує потребу в адекватному збільшенні рівня забезпечення захисту даних, що для криптографічних механізмів з відкритим ключем може бути досягнене за рахунок збільшення розрядності бітів. Це призводить до помітного збільшення часу реалізації обчислень механізмів криптографічної безпеки даних.

Отже, основною задачею наукових досліджень є прискорення обчислень, які реалізують криптографічні механізми захисту даних з відкритим ключем, за допомогою багатопроцесорних можливостей сучасних комп'ютерних систем. Основним способом вирішення цієї проблеми є розпаралелювання основної операції - модулярного множення, тобто сумісне одномоментне виконання певної кількості складових.

Наукова проблема прискорення модульного множення для криптографічних систем забезпечення інформаційної безпеки є актуальною на сучасному етапі розвитку інформаційно-комп'ютерних технологій.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ТЕХНОЛОГІЙ МОДУЛЯРНОГО МНОЖЕННЯ ЯК БАЗОВОЇ СКЛАДОВОЇ КРИТПОГРАФІЧНИХ АЛГОРИТМІВ З ВІДКРИТИМ КЛЮЧЕМ

В сучасному науковому світі криптографія з відкритим ключем заснована та ґрунтується на операції модульного піднесення до степеня – це основна розрахункова операція в широкому спектрі криптографічних алгоритмів на відкритому ключі. Проблема прискорення обчислення цієї операції є першорядною для підвищення надійності комплексів та систем захисту інформації та інформаційної безпеки.

1.1 Аналіз вимог до технологій модулярного піднесення до квадрату в сучасних системах комп’ютерного керування

Для комп'ютерів персонального використання та потужних обчислювальних систем це нагальне питання можна залагодити, включивши до системи на рівні апаратного забезпечення криптопроцесори. Сучасні реалії дозволяють виробляти всеохоплюючий асортимент криптопроцесорів [1] комерційно, майже всі з них застосовують операцію модульярного експоненціювання на рівні апаратного обслуговування. Але для значної ніші, до якої входять мобільні пристрої, мікроконтролери терміналів систем, що призначені для дистанційного керування реальними об'єктами, в яких Інтернет забезпечує чільну роль засобу обміну даними, питання прискорення реалізації обчислення операції модульного піднесення до степеня є першочерговим та незмінно актуальним. Для багатьох критичних прикладних задач застосування криптопроцесорів неприпустиме з огляду впливу на інформаційну безпеку.

Існує загальновідома поширена думка, що класичний алгоритм модульного піднесення до степеня має дуже жорстку структуру є непорушно послідовним і є практично та теоретично непридатним для паралельної обробки. [2]. Єдиний спосіб підвищити швидкість його виконання - це

мінімізувати час виконання його компонентів - модульного знаходження добутку та модульної зменшування. Існує можливий підхід для прискорення обчислювальної реалізації модульного добутку - це суміщення в часі виконання кількох операцій множення та зменшування. Отже, загальною проблемою, є питання збільшення швидкості обчислення модульного піднесення до степеня, в основі якого лежить паралельна обробка його основної операції - модульного множення.

Обчислення операції модульного множення чисел, розрядність яких перевищує розрядність модуля $I \cdot J \bmod K$ включає дві частини: множення компонентів $M=I \cdot J$ та знаходження залишку від результату після ділення добутку $I \cdot J$ на модуль K. У криптографії з відкритим ключем модуль K є складовою відкритого ключа, тому його можна вважати константою[3].

Існує диференціювання алгоритмів модулярного множення на два напрямки: ті, в яких множення $I \cdot J$ та послідовне обчислення залишку від ділення на модуль виконуються поступово в часі, тобто послідовні та ті, в яких операції множення та знаходження залишку від ділення чергуються в контексті часу тобто змішані[4].

Алгоритми модульного множення першої групи мають можливість використовувати вбудовані інструкції множення процесора. Для цього, L чисел, що беруть участь у операції модульного множення, розбиваються на X фрагментів, довжина Y яких відповідає ємності процесора. Це дозволяє зменшити обчислювальну складність за допомогою попарного множення фрагментів з поступовим додаванням результатів. Використання цього методу дозволяє ефективно використовувати апаратне забезпечення сучасних процесорів, включаючи вбудовані схеми для прискорення множення [5].

Базовий алгоритм знаходження модульного добутку передбачає обчислення модульного залишку в ході операції знаходження цілочисельного ділення кратного, довжина якого $2 \cdot y$ біт на дільник довжиною y біт для

обчислення частки та залишку [6]. Очевидно, що операція редукції при виконанні базового алгоритму потребує x·(x+2,5) операцій знаходження добутку та x операцій цілочисельного ділення, адже ділення L-бітових чисел на процесорі розрядність якого y є дуже низько ефективним. Сьогодні загальновідомими є широкий ряд алгоритмів, основна задача який підвищити продуктивність програмної реалізації обчислення операції знаходження модулярного добутку. Левова частка із цих алгоритмів передбачає забезпечення покращення продуктивності у відповідь на підвищення ефективності прискорення модулярної редукції за допомогою операції цілочисленного ділення, що є частиною базового алгоритму.

1.2 Огляд методів прискорення операції модулярного множення

Найбільш відомі та поширені прикладні технології знаходження модулярного залишку від ділення на сьогодні це алгоритм Баррета та алгоритм Монтґомері. Перший відповідно заснований на обчисленні максимального малого значення e, для якого I·J - e·K < K. Тоді як залишок від ділення визначається з формули F = I·J-e ·K. Звідси зрозуміло, що алгоритм Баррета виконується з використанням двох знаходжень добутків за бітністю y, в той час, коли модулярне знаходження добутку I·J mod K з використанням алгоритму Баррета виконується з використанням трьох знаходжень добутку.

Друга згадана вище технологія для знаходження залишку від ділення на модуль, яка має властивості пристосування до архітектури універсальних процесорів – це широко розповсюдженний алгоритм Монтґомері. В основі алгоритму закладено принцип, відповідно до якого він заміщує виконання операції ділення з довільним модулем K на реалізацію ділення за степенем числа 2, що має виску ефективність при виконанні зсувів. Відповідно, модулярне зменшення в згаданому алгоритмі передбачає x·(x+1) знаходжень добутків.

Використання алгоритму Монтґомері при множені L-бітних чисел на y-

бітному процесорі має загальну обчислювальну складність реалізації визначається 2·x2+y знаходження добутку процесора та 4·x2+4·x+2 операціями знаходження суми процесора. Алгоритм Монтґомері має значну перевагу у тому, що його легко інтегрувати в процес множення відносно простим способом в часі. Саме це дає можливість обмежувати довжину часткових результатів у процесі знаходження модулярного добутку до L+1, а це слугує каталізатором до зменшення кількості розрахунків порівняно з послідовним алгоритмом, де довжина проміжних результатів досягає 2·L-біт.

Знаходження модульного добутку можна здійснити розділивши операцію знаходження добутку та зменшення. Цей окремий випадок дозволяє використовувати алгоритми Карацуби підквадратичного або квазі-лінійного множення [3,4], що дозволяють пришвидшити множення в 1,4 рази. У випадку окремого виконання операцій множення та редукції, редукція виконується за допомогою технології Баррета [5], що дозволяє зменшити процес знаходження залишку від ділення до двох операцій множення.

Для підвищення швидкості знаходження модулярного добутку найчастіше застосовуються такі методи:

- суміщення в часі обробки кількох розрядів операнда;

- попередні розрахунки, які зберігаються окремо в спеціальній пам'яті таблиць та залежать тільки від значення модуля;

- паралелізація обчислення операції додавання частин модульного множення.

В прикладних задачах має місце використання суміщень цих методів між собою для покращення кінцевих результатів.

З поміж усіх завдань сучасних систем захисту інформації [1], що э невід’ємними складовими нинішніх комп’ютерних систем і мереж, можна виокремити такі найважливіші:

- Застосування різноманітних дій та інструментів для запобігання

раптовому несанкціонованому доступу (читанню) до інформації, потік якої постійно відбувається в комп'ютерних мережах, і яка здатна зберігатись у системі як файли в пам’яті різних носіїв цільових машин у вбудованих системах обробки інформації.

- Застосування дій та інструментів для збереження автентичності та змістової цілісності згаданих вище наборів даних, які користувачі передають використовуючи мережу, а також файлів напряму, простіше кажучи, здійснюють захист інформації від змін неавторизованими користувачами, в тому числі у злочинних цілях.

- Застосування дій та інструментів для гарантованого закритого користувацького доступу до вбудованих систем роботи з інформацією та наявних баз даних, іншими словами – створення системи надійного розпізнавання користувачів.

1.3 Аналіз можливостей прискорення програмної реалізації модулярного піднесення до квадрату

Для розв’язання цієї насушної проблеми створена низка рішень: складний набір протоколів (у вигляді зв’язаних засобів організації), численний набір алгоритмів засобів для прикладної реалізації. Особливої уваги в контексті всіх рішень потребують алгоритми, що належать до розділу криптографії , вони в повній мірі застосуються для вирішення згаданого списку потенційних задач для забезпечення інформаційної безпеки у сучасних комп’ютерних системах та мережах. Варто також згадати, що технічні засоби захисту програють криптографічним алгоритмам у гнучкості, а також у контексті змін, простіше кажучи, криптографічні методи захисту придатні до швидких змін захисних властивостей, як наприклад, ключ, а також менш вразливі до впливу технічних засобів злому.

На основі головних показників якості алгоритму формується оцінка та проводиться аналіз змінних криптографічних алгоритмів. Важливість кожного

показника визначається протоколом та специфікою випадків застосування кожного з відомих алгоритмів. Крім того, існує значна різноманітність криптографічних алгоритмів, і для кожного з них існують унікальні вимоги щодо якості, які мають різняться в залежності від представленого класу алгоритмів. Для алгоритмів з відкритим ключем чільним критерієм є час, що необхідний для генерацій відкритих і закритих ключів, в той же час цей критерій не займає однаково значущого місця в оцінці якості для інших класів криптографічних алгоритмів. Тож, існує така загальна градація вимог до криптографічних алгоритмів:

- рівень захищеності інформації, що передається у вигляді повідомлення , який може бути забезпечений вибраним алгоритмом. Якщо підходити з іншого боку та спростити уявлення, це безпосередньо вартість затрачених обчислювальних ресурсів для злому вибраного алгоритму.

- швидкодія обробки даних при використанні криптографічного алгоритму на наявних ресурсах процесора.

- для успішної реалізації ефективного розрахункового процесу за алгоритмом необхідні спеціалізовані та проблемно-орієнтовані засоби розрахунку, а також наявність відповідних умов;

- забезпечення гнучкості обробки інформаційних повідомлень, а також захисту від навмисних і ненавмисних перешкод у каналах передачі даних і під час зберігання зашифрованих елементів інформації на аналогових носіях.

Очевидно, що між названими критеріями існує певна суперечливість і забезпечення одного з них каталізує погіршення показників інших, тому для остаточної оцінки якості для криптографічних алгоритмів варіант компромісної відповідності визначених раніше критеріїв визнаний найбільш оптимальним при врахуванні усіх особливостей умов і протоколів використання поточного алгоритму. Інтегральна оцінка якості кожного з алгоритмів складається з вирахуваних окремо компонентів якості на основі конкретних деталей прикладного застосування того чи іншого алгоритму.

Значимість елементів параметра якості у випадку комерційної криптографії можна описати за допомогою таких складових [2]:

1. В першу чергу, рівень захищеності інформації при використанні будь-якого алгоритму має бути таким, щоб здобуття цієї інформації шляхом несанкціонованого злому не було фінансово вигідним для злочинця, який має намір заволодіти передаваними даними, простіше кажучи, прибуток, отриманий від здобутої інформації не має перевищувати вартість обчислювальних ресурсів використаних для проходження крізь алгоритмічний захист.

2. Вартість реалізації криптографічного алгоритму повинна бути оптимальною, щоб застосування криптографічного захисту було економічно вигідним. Вона включає складність реалізації алгоритму, яка вимагає обчислювальних ресурсів для передачі інформаційного повідомлення з криптографічним захистом. Крім того, супутні витрати, такі як витрати на захоплення лінії зв'язку в разі, коли швидкість криптоперетворення нижча за швидкість передачі інформації по лінії, повинні бути меншими, ніж збитки, що можуть виникнути внаслідок несанкціонованого доступу до захищеної інформації.

З урахуванням основних принципів організації захисту інформації криптографічними засобами для недержавних комерційних установ, очевидно, що потрібно розробити групу добре досліджених і перевірених криптоалгоритмів, які мають різні рівні безпеки і, відповідно, швидкості реалізації. Ця політика активно реалізується в практиці багатьма компаніями, включаючи всесвітньо відомого виробника комп'ютерів IBM. Вони розробляють сімейства алгоритмів, які відрізняються рівнями безпеки та вартістю практичної реалізації, для вирішення конкретних типів завдань у сфері захисту інформації [3].

У результаті в залежності від способу криптографічного перетворення всі алгоритми шифрування криптографічних даних поділяються на такі групи:

- оборотні, для яких можна алгоритмічно визначити процедури як прямого, так і зворотного перетворень.

- алгоритми, які називаються криптографічно незворотними або односторонніми. Вони дозволяють алгоритмічно визначити лише пряме перетворення, що призводить до отримання цифрового підпису повідомлення. Проте, зворотне перетворення стає алгоритмічно нерозв'язною задачею, тобто такою, що в основному вирішується лише за допомогою складних обчислювальних методів, наприклад, шляхом перебору або використання спеціальних алгоритмів сортування.

Асиметричні криптографічні алгоритми, які використовують відкриті ключі, є ще одним важливим типом оборотних алгоритмів. Більшість сучасних алгоритмів цього класу базуються на задачах підвищеної складності теорії чисел. Відмінністю несиметричних алгоритмів від симетричних є значно ширші можливості для розробки ефективних протоколів захисту інформації, де один з ключів може бути відкритим, незалежно від того, чи є він закритим чи відкритим. Це дає можливість застосовувати такі протоколи у ситуаціях, коли один ключ залишається таємним. У кінці 70-х років виникла "нова ера розвитку криптографії" завдяки технології відкритих ключів, що сприяла виникненню і розвитку багатьох інноваційних принципів організації захисту інформації.

RSA (Rivest-Shamir-Adleman) є одним з найвідоміших асиметричних криптографічних алгоритмів, який отримав широке застосування. Основна ідея RSA полягає у факторизації великих чисел. В алгоритмі RSA використовуються два ключі: відкритий (шифрувальний) ключ, позначений парою чисел (exp, num), і закритий ключ, пов'язаний з цими числами значенням t. Інформаційне повідомлення, представлене як блок довжиною N бітів, трактується як двійкове число a, і процес шифрування перетворює його в кодовий код довжиною n бітів, тоді як процес дешифрування описується такими рівняннями:

$$code = a^{\exp} \bmod num\ , \quad a = \text{code}^{integer} \bmod num\ . \tag{1.1}$$

Ці рівняння дозволяють зашифрувати повідомлення за допомогою відкритого ключа і розшифрувати його за допомогою відповідного закритого ключа. Результатом шифрування є шифрований текст c, який може бути розшифрований тільки за допомогою відповідного закритого ключа.

Окрім RSA, існують й інші відомі асиметричні алгоритми, такі як алгоритм Мерклі-Хеллмана, алгоритм Ель-Гамаля і Рабіна, які також засновані на складних математичних задачах теорії чисел. Кожен з цих алгоритмів має свої особливості і застосування в різних сферах криптографії**.**

Завдяки використанню великих чисел і складних операцій, які виконуються над ними, RSA має вагомо більшу обчислювальну складність порівняно з симетричними алгоритмами, такими як DES.

Розрядність чисел, використовуваних в RSA, зазвичай розраховується з урахуванням потрібного рівня безпеки. Зазвичай для RSA використовують ключі довжиною 1024, 2048 або навіть 4096 біт. Більша довжина ключа забезпечує більшу безпеку, але також призводить до більшої обчислювальної складності.

Щодо продуктивності RSA на сучасних мікропроцесорах загального призначення, дійсно, його алгоритм не є найкраще пристосованим до їх архітектури. Виконання операцій з великими числами може бути витратним з точки зору швидкодії. Однак, існують різні методи оптимізації та розпаралелювання RSA-обчислень, що дозволяють досягти кращої продуктивності.

Також важливо зазначити, що апаратна реалізація RSA на спеціалізованих пристроях (наприклад, криптографічних процесорах або апаратних модулях безпеки) може значно покращити продуктивність RSA-шифрування. Спеціалізовані пристрої можуть ефективно виконувати операції з великими числами і прискорити обчислення RSA в порівнянні з загальнопризначеними мікропроцесорами.

Отже, хоча RSA має свої особливості, такі як більша обчислювальна

складність і менша продуктивність на загальнопризначених мікропроцесорах, він залишається одним з найбільш використовуваних асиметричних криптографічних алгоритмів, завдяки своїй безпеці та широкому застосуванню в різних областях.

Модуль представляє число num, яке отримується шляхом перемноження простих чисел x і k. В алгоритмі також використовується натуральне число exp, яке задовольняє умову

$$НОД(exp, \varphi(num)) = 1. \quad (1.2)$$

Щоб обчислити φ(num) за відомими x і k, використовується формула

$$\varphi(num) = (x - 1)(k - 1). \quad (1.3)$$

Пара (num, exp)[10] є відкритим ключем, який використовується для кодування в криптосистемі RSA. Припустимо, що blck - це ціле число, яке представляє блок повідомлення і задовольняє умову $0 \leq blck \leq n-1$. Зашифрований блок повідомлення, позначений як E(blck), обчислюється шляхом взяття залишку від ділення $blck^{exp}$ на num.

Щоб зашифрувати повідомлення методом RSA, спочатку його потрібно підготувати. Повідомлення повинно бути розбите на послідовні блоки, кожен з яких є числом, меншим за n (модуль RSA).

Кожен блок повідомлення шифрується окремо. Це означає, що кожний блок кодується індивідуально. Результатом шифрування є набір зашифрованих блоків, які також залишаються окремими елементами. Важливо зазначити, що ці блоки не повинні бути об'єднані в одне велике число, оскільки це запобігає остаточному розшифруванню повідомлення.

Щоб переконатися в надійності системи RSA як криптографічного шифрування, розглянемо можливості її злому, коли відома лише пара значень (num, exp).

Для розшифрування блоку, зашифрованого з використанням RSA,

потрібне значення t > 0, яке розкладається на множники за модулем φ(num). Однак основна перешкода полягає у тому, що єдиний відомий метод заснований на використанні розширеного алгоритму Евкліда для exp і φ(num). Знаходження значення φ(num) за формулою (1.3) вимагає знання x і k, що вимагає розкладання num на його прості множники. Ураховуючи складність такого розкладу, систему RSA вважають безпечною.

Допустимо, що в один момент хтось відкриває алгоритм, який обчислює φ(num) без знання множників num. Це може статися, якщо буде створено ефективний спосіб обчислення φ(num) безпосередньо з num і exp. Фактично, це прихований спосіб розкладання num на множники. Можна навіть стверджувати, що в разі, якщо відомі значення num = xk і φ(num) = (x - 1) (k - 1), достатньо просто обчислити x і k.

Формула φ(num) = (x - 1) (k - 1) = xk - (x + k) + 1 = num - (x + k) + 1 показує, що сума простих дільників x + k = num - φ(num) + 1 є відомою. З цього можна отримати: $(x + k)^2 – 4num = (x^2 + k^2 + 2 xk) – 4 xk = (x-k)^2$

Отже, різниця (x - k) також відома. Розв'язавши цю систему рівнянь, ми можемо легко обчислити x і k, розкладаючи num на його множники. З цього можна зробити висновок, що алгоритм, який обчислює φ(num), по суті, також розкладає число num на його множники.

Припустимо, що існує алгоритм, який може безпосередньо визначити значення t за допомогою num і exp. Але враховуючи вираз exp * t (mod φ(num)), навіть якщо ми знаємо num, exp і t, це недостатньо для розкриття кіль. Існує алгоритм, який може привести до такого розкладу, але його ефективність для великих значень num ще не досліджена. Тому, на сьогоднішній день, єдиним варіантом є систематичний перебір всіх можливих варіантів для пошуку блоку. Зазначено, що безпека RSA ґрунтується на розкладі числа n на його прості складові множники, але доведення цього твердження немає. Це означає, що правильний вибір простих чисел x і k є важливим для забезпечення безпеки криптосистеми RSA. Необхідно уникати невеликих множників, але також

враховувати достатню різницю між ними, оскільки навіть великі множники можуть бути порушені алгоритмом Ферма.

В багатьох країнах використовується алгоритм Ель-Гамаля, який базується на складній задачі дискретного логарифмування. Цей алгоритм виявляє більшу стійкість до розкриття порівняно з RSA. Для шифрування використовуються цілі числа m, k і l як відкриті ключі шифрування. Секретний ключ для розшифровки - це ціле число s, яке пов'язане з m, k і l за формулою $k \equiv ls \pmod m$, де s менше за x.

Для шифрування блоку інформаційного повідомлення h необхідно обчислити дві величини: v і w. Ці значення використовуються для створення зашифрованого тексту разом із випадковим цілим числом i. Обчислення здійснюються наступним чином:

$$v = l^i \bmod m, \qquad w = k^i \bmod m. \tag{1.4}$$

Декодування даних з зашифрованих компонентів v і w здійснюється шляхом віднімання:

$$h = v/w^s \bmod m. \tag{1.5}$$

Алгоритм Ель-Гамаля має серйозну перевагу - його можна використовувати як для шифрування, так і для цифрового підпису. У реальних застосуваннях часто використовуються числа з великою кількістю цифр, наприклад, 512, 768 або 1024. Швидкість шифрування в цьому алгоритмі майже в 1,5 рази вища, ніж швидкість дешифрування. Загалом, час обчислення для алгоритму Ель-Гамаля майже порівнянний з RSA.

Існує ще одна група асиметричних алгоритмів, яка називається ECC (або Elliptic Curve Cryptography), і вона базується на особливостях додавання точок на еліптичних кривих. Ці особливості створюють складність задачі дискретного логарифмування на еліптичних кривих над скінченими полями Галуа. Для шифрування використовуються певні параметри еліптичної кривої та значення

координат початкової точки. Секретним ключем є короткий ключ t, за допомогою якого обчислюються значення координат іншої характерної точки.

Алгоритм ECC має свої переваги, такі як висока безпека при використанні коротших ключів, а також ефективність у використанні ресурсів, що забезпечує швидку обробку даних. Це робить його популярним в застосуваннях з обмеженими обчислювальними ресурсами, наприклад, в мобільних пристроях або безпроводових сенсорних мережах.

Основною перевагою алгоритму ESS (Elliptic Curve Cryptography) порівняно з раніше згаданими алгоритмами є малий розмір ключа. В той час, як для RSA або ElGamal мінімальна довжина ключа становить 512 біт, в ESS досягається той самий рівень стійкості до розкриття з ключем всього лише 106 біт. Це означає, що ESS забезпечує високий рівень безпеки з меншим обсягом ключа.

Що стосується швидкості генерації ключів, то в алгоритмі ECC вона значно нижча, ніж у RSA. Генерація ключів в алгоритмі ECC займає на три порядки менше часу, порівняно з RSA. Це сприяє більш ефективному використанню ресурсів і забезпечує швидшу обробку даних.

Крім алгоритмів шифрування, модульні арифметичні операції часто використовуються в протоколах аутентифікації для віддалених користувачів комп'ютерних систем. Ці операції дозволяють перевірити достовірність повідомлень та ідентифікувати користувачів за допомогою математичних перевірок та обчислень.

Висновки до розділу 1

З За результатами виконання першого розділу дипломного проекту, спрямованого на аналіз можливостей прискорення операції модулярного піднесення до квадрату багаторозрядних чисел, що використовуються в криптографічних системах з відкритим ключем, можна зробити наступні висновки:

1. Операція модулярного піднесення до квадрату є критичною з позиції програмної реалізації в реальному часі на термінальних мікроконтролерах Інтернету речей. Велика розрядність чисел, що використовується в сучасних протоколах, разом з обмеженою потужністю та компактністю термінальних процесорів мікроконтролерів, становить серйозну перешкоду для реалізації криптографічних алгоритмів в реальному часі.
2. Проведений аналіз показав, що найбільш перспективним шляхом прискорення програмної реалізації базової операції криптографічних алгоритмів з відкритим ключем - модулярного піднесення до квадрату є пошук шляхів прискорення виконання саме операції піднесення до квадрату, яка становить дві третини обчислень модулярного піднесення до степеня.
3. В результаті детального вивчення операції модулярного піднесення до квадрату було виявлено, що швидкодію можна підвищити приблизно вдвоє шляхом виключення зайвих операцій. Іншим резервом прискорення операції модулярного піднесення до квадрату є поєднання в часі операцій піднесення до квадрату та модулярної редукції.
4. Порівняльний аналіз редукції Баррета та Монтгомері показав, що застосування останньої забезпечує суттєво більші можливості для прискорення виконання операції модулярного піднесення до квадрату.

5. Детальний аналіз показав, що важливим чинником прискорення операції модулярного піднесення до квадрату є організація одночасного множення та редукції отриманого часткового результату з обробкою декількох розрядів множника та використання таблиць передобчислень, які залежать лише від значення модуля.

РОЗДІЛ 2
РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДУ ПРИСКОРЕННЯ МОДУЛЯРНОГО ПІДНЕСЕННЯ ДО КВАДРАТУ

В оглядовому розділі бакалаврської роботи показано, що операція модулярного експоненціювання є базовою для практично всіх сучасних криптографічних алгоритмів з відкритим ключем, зокрема для механізмів несиметричного шифрування, формування та перевірки цифрового підпису, автентифікації учасників віддаленої інформаційної взаємодії з застосуванням прогресивної криптографічної концепції нульового розголошення.

Показано також, що обчислювальна реалізація операції модулярного експоненціювання може бути здійснена за двома алгоритмами, строго послідовний характер яких практично виключає можливість розпаралелювання. Теоретично доведено, що критичний шлях алгоритму модулярного експоненціювання зі старших розрядів коду експоненти включає $1.5 \cdot n$ операцій модулярного множення. Це означає, що всі операції модулярного множення належать критичному шляху, тобто не можуть бути розпараленими в принципі. Критичний шлях різновиду алгоритму модулярного експоненціювання з молодших розрядів коду експоненти складається з n операцій модулярного множення. Іншими слова, третина із загальної кількості операцій модулярного множення може виконуватися одночасно з обчисленням 2/3 операцій, що належать критичному шляху. Відповідно, доведено, що єдина можливість прискорення важливої для практичних застосувань операції модулярного експоненціювання полягає в розпаралелювання її базової компоненти – модулярного множення чисел, довжина яких на порядок перевищує розрядність процесору.

Проведений аналіз показав, що алгоритм модулярного експоненціювання зі старших розрядів коду експоненти складається з n операцій модулярного піднесення до квадрату і $0.5 \cdot n$ операцій модулярного

множення на постійне число. Це означає, що ще одна можливість прискорення обчислювальної реалізації модулярного експоненціювання полягає в урахуванні особливостей двох базових операцій:

- модулярного піднесення до квадрату;
- модулярного множення на постійне число;

Аналіз алгоритму модулярного експоненціювання з молодших розрядів коду експоненти засвідчив, що він складається з n операцій модулярного піднесення до квадрату і $0.5 \cdot n$ операцій модулярного множення. Відповідно, прискорення при використанні цього типу алгоритму модулярного експоненціювання може бути досягнене лише за рахунок використання спеціальної процедури модулярного експоненціювання, яка враховує специфічні особливості цього різновиду модулярного множення.

2.1 Аналіз особливостей обчислювальної реалізації операції модулярного піднесення до квадрату

Цілком очевидно, що основним резервом прискорення операція модулярного піднесення є виключення операційної надлишковості, зумовленої симетричністю модулярного множення. що виконується над однаковими числами.

Якщо модулярне піднесення до квадрату n-розрядних чисел здійснюється на m-розрядному процесорі, то вони розділяються на k секцій довжиною m, рівній розрядності процесора, тобто $k=n/m$. При модулярному множенні двох різних чисел $A \cdot B \bmod M$ здійснюється процесорне множення кожної секції множника B на кожну із k секцій множимого A. Відповідно, кількість η_m операцій процесорного множення, потрібних для множення двох n-розрядних чисел складає:

$$\eta_m = k^2. \qquad (2.1)$$

Для формування добутку $A \cdot B$, крім η_m операцій процесорного множення, потрібно здійснити η_a процесорних операцій додавання. Зрозуміло, що для

додавання одного результату процесорного множення потрібно реалізувати 2 операції процесорного додавання в силу того, що розрядність процесорного добутку вдвічі перевищує розрядність процесору. Крім того, зі ймовірністю 0.5 при додаванні до суми часткових добутків поточного результату процесорного добутку, виникає перенос, тобто сумарна середня кількість η_a процесорних операцій додавання обчислюється за формулою:

$$\eta_a = 2.5 \cdot k^2 . \qquad (2.2)$$

Таким чином, загальний час виконання операції T_m множення двох різних чисел A·B визначається наступною формулою:

$$T_m = k^2 \cdot (t_m + 2.5 \cdot t_a), \qquad (2.3)$$

де t_m -час виконання процесором команди цілочисельного множення, а t_a - час виконання процесором команди додавання.

Враховуючи, що для більшості сучасних процесорів і мікроконтролерів час виконання команди множення приблизно в 30 раз більший від часу здійснення команди додавання, тобто $t_m \approx 30 \cdot t_a$ [14], формула (2.3) може бути представлена в більш зручному для аналізу вигляді:

$$T_m = 1.083 \cdot k^2 \cdot t_a . \qquad (2.4)$$

При виконанні операції піднесення до квадрату n-розрядного числа також здійснюється його поділення на k секцій, довжина яких дорівнює розрядності процесора. Проте, в силу того, що фактично однойменні секції множимого і множника однакові, то кількість η_s операцій процесорного множення при піднесенні до квадрату зменшується до рівня, який визначається наступною формулою:

$$\eta_s = k + \frac{(k-1)^2}{2} = k + \frac{k^2}{2} - k + \frac{1}{2} \approx \frac{k^2}{2}. \qquad (2.5)$$

Для формування квадрату A^2, крім η_s операцій процесорного множення, потрібно здійснити η_{aS} процесорних операцій додавання. Оскільки при обчисленні квадрату до суми часткових добутків додаються часткові добутки зсунуті на один розряд ліворуч, ця операція фактично розповсюджується на три секції. Виходячи з цього, додавання одного результату процесорного множення потрібно реалізувати три операції процесорного додавання. таким чином, загальна кількість η_{aS} процесорних операцій додавання обчислюється за формулою:

$$\eta_{aS} = \frac{3}{2} \cdot k^2. \qquad (2.6)$$

Таким чином, загальний час виконання операції T_S піднесення до квадрату n-розрядного числа A^2 визначається наступною формулою:

$$T_S = \frac{k^2}{2} \cdot (t_m + 3 \cdot t_a) \approx 0.55 \cdot k^2 \cdot t_m, \qquad (2.7)$$

Відповідно, порівняльний аналіз формул (2.7) та (2.4) дозволяє зробити висновок про те, що операція піднесення до квадрату теоретично може виконуватися в 1.969 раз швидше в порівнянні з множенням різних чисел, за умови, що їх довжина значно перевищує розрядність процесора.

Проте, наведені оцінки стосуються лише операції множення багаторозрядних чисел. Добре відомо, що операція віднаходження залишку від ділення на заданий модуль, яка називається модулярною редукцією, потребує мінімум вдвоє більше обчислювальних ресурсів для реалізації [16]. Операція редукції обчисленого квадрату A^2, тобто отримання залишку від ділення квадрату A^2 на модуль M вимагає суттєво більших обчислювальних ресурсів,

оскільки неможливо використовувати для цієї мети процесорну операцію цілочисельного ділення. Тому основні зусилля досліджень останніх років спрямовані на прискорення редукції.

Тут використовують два основних напрямки: використання особливостей використання операції модулярного експоненціювання в реальних протоколах інформаційної безпеки та заміна неефективної операції ділення до більш ефективної операції множення. Перший напрямок базується на тому, що в реальних системах інформаційної безпеки на базі криптографії з відкритим ключем, модуль M є його частиною і тому змінюється відносно рідко. Тому в рамках цього напряму підвищення швидкості виконання модулярної редукції досягається за рахунок якнайширшого застосування передобчислень, які залежать тільки від модуля.

Найбільш відомі технології цього напрямку сформульовані у вигляді методу Бунімова- Шиммлєра [19]. Цей метод передбачає попереднє виконання передобчислень $t_1=2^{n+1} \bmod M$, $t_2= 2^{n+2} \bmod M$,…,$t_n= 2^{2n} \bmod M$. Потім табличні значення, що відповідають одиницям не редукованого квадрату додаються і отримана в результаті S знову рекурсивно редукується. Щоб зменшити величину суми S можна у таблиці зберігати половину позитивних, а половину негативних значень. Суттєвим недоліком розглянутого підходу до реалізації модерної редукції є те, що доводиться працювати з числами, розрядність яких вдвічі перевищує розрядність *n* модуля M.

Як зазначалося вище, альтернативний підхід передбачає використання для реалізації операції цілочисельного ділення декількох операцій множення довгих чисел.

Зокрема, при використанні для обчислювальної реалізації модулярної редукції технології Баррета [17], вона виконується у вигляді двох операцій множення *n*–розрядних чисел. Причому, при модулярній редукції за цим методом два додаткових множення здійснюються над різними числами. Це призводить до того, що лише одне із трьох множень, які реалізують операцію

модулярного піднесення до квадрату можуть бути виконані по описаній вище скороченій схемі. В цій схемі доводиться працювати з числами, розрядність яких вдвічі перевищує розрядність *n* модуля M. Це різко знижує ефективність програмної реалізації такої модулярної редукції.

Найбільш прогресивний метод реалізації модулярної редукції запропоновано П. Монтгомері. Пітер Монтгомері [18] показав як "розгорнути" класичний алгоритм модулярного експоненціювання багаторозрядних чисел таким чином, що поширення переносів розповсюджується у напрямку від бітів з більшою вагою в сторону розрядів з меншою вагою. Фактично це дозволяє замінити операцію цілочисельного ділення на порядки більш простою в технологічному плані операцією логічного зсуву. Все це помітно зменшує необхідну площу на кристалі та скорочує критичний шлях. Незважаючи на те, що складність двох алгоритмів здається ідентичною за часом і площею на чіпі, показники для алгоритму Монтгомері на практиці кращі. Монтгомері використовує молодшу цифру добутку, що акумулюється, щоб визначити, яке кратне M число додати, а не віднімати. Він вибирає цифри для множення у зворотному порядку, тобто з молодших до старших розрядів множника і на кожній ітерації виконує операцію ділення суми часткових добутків на два, яка технологічно реалізується простим логічним зсувом коду праворуч.

Важлива перевага модерної редукції Монтгомері полягає в тому, що операція множення інкапсульована в процес множення. Це має наслідком те, що не відбувається зростання розрядності суми часткових добутків, оскільки обчислені часткові добутки відразу редукуються.

Відповідно, постає задача створення такої модифікації модулярної редукції Монтромері, яка б врахувувала і використовувала для прискорення обчислень особливості модулярного піднесення до квадрату чисел, довжина яких значно перевищує розрядність процесора. Іншими словами, потрібно створити технологію, яка трансформує операційну надмірність операції модулярного піднесення до квадрату в прискорення її обчислювальної реалізації.

Причому, при модулярній редукції за цим методом два додаткових множення здійснюються над різними числами. Це призводить до того, що лише одне із трьох множень, які реалізують операцію модулярного піднесення до квадрату можуть бути виконані по описаній вище скороченій схемі. В цій схемі доводиться працювати з числами, розрядність яких вдвічі перевищує розрядність *n* модуля M. Це різко знижує ефективність програмної реалізації такої модулярної редукції.

2.2 Метод прискореного модулярного піднесення до квадрату на базі редукції Монтгомері

Задача полягає в розробці такої організації обчислень модулярного квадрату A mod M, яка б в рамках застосування суміщеної з множенням редукції Монтгомері дозволяла виключити дублювання операцій, зумовлених однаковістю множимого та множника.

Число A, яке підноситься до квадрату, задається множиною значень його двійкових розрядів $\{a_{n-1},a_{n-2},\ldots,a_1,a_0\}$ так, що

$$A=\sum_{j=0}^{n-1} a_j \cdot 2^j , \qquad (2.8)$$

де $\forall i \in \{0,1,2,\ldots,n\text{-}1\}: \ a_i \in \{0,1\}$.

Відповідно, квадрат A^2 числа A може бути представлений як сума попарних добутків його двійкових розрядів з урахуванням їх ваги:

$$A^2=\sum_{l=1}^{n}\sum_{j=1}^{n} a_l \cdot a_j \cdot 2^{l+j-2} . \qquad (2.9)$$

Класична схема множення [13] виходить із такого представлення формули (2.9):

$$A^2 = \sum_{l=1}^{n} a_i \cdot 2^{l-1} \cdot \sum_{j=1}^{n} a_j \cdot 2^{j-1} = \sum_{j=1}^{n} a_l \cdot 2^{i-1} \cdot A. \qquad (2.10)$$

Відповідно, ця схема передбачає накопичення суми часткових добутків, кожне з яких являє собою добуток числа *A* на бінарне значення *l*-го біту числа A. В залежності від напрямку проходу суми в формулі (2.10) можна розглядати множення зі старших або молодших розрядів двійкового коду множника. Операція множення на ступінь 2 в формулі (2.10) співвідноситься зі зсувом суми часткових добутків або множимого.

Враховуючи, що $a_i \cdot a_j = a_j \cdot a_i$, та $a_i \cdot = a_i$ отримана формула (2.10) може бути перетворена до такого вигляду:

$$A^2 = 2 \cdot \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} a_i \cdot a_j \cdot 2^{i+j-2} + \sum_{l=1}^{n} a_j \cdot 2^{2 \cdot l-2} =$$

$$= \sum_{l=1}^{n} \sum_{i=1}^{n-1} (a_i \cdot 2^{i-1} \cdot 2 \cdot \sum_{j=i+1}^{n} a_j \cdot 2^{j-1}) + \sum_{l=1}^{n} a_l \cdot 2^{2 \cdot (l-1)} \quad . \qquad (2.11)$$

Неважко помітити, що друга компонента в отриманій формулі (2.11) являє собою "розріджений" двійковий код числа A, утворений вставкою нулів між двійковими розрядами числа A. При застосуванні множення без переносів в результаті піднесення до квадрату числа A залишається лише друга компонента, тобто "розріджений" двійковий код числа A. Ця властивість широко використовується при прискорення піднесення до квадрату на кінцевих полях Галуа для викосокопродуктивної реалізації криптографії еліптичних кривих.

З формули (2.11) випливає, що в процедурному плані обчислення її першої компоненти може бути здійснене шляхом послідовного аналізу бітів числа A, причому, якщо *i*-тий біт цього числа дорівнює одиниці: a_i=1 - здійснюється додавання до часткового добутку зсунутого на один розряд ліворуч коду A в якому *i*+1 молодших розряди дорівнюють нулю.

Приклад обчислення першої компоненти суми часткових добутків в формулі (2.11) для схеми множення з молодших розрядів множника зі зсувом суми часткових добутків для числа $A=11_{10} = 1011_2$ показано в таблиці 2.1.

Для отримання значення $A^2 = 11^2$ потрібно до одержаного значення першої компоненти суми в формулі (2.11) : $1\ 1\ 0\ 1\ 0\ 0_2 = 52_{10}$ додати другу компоненту - "розріджений" двійковий код числа А: $1000101 = 69_{10}$: 52+69 = 121. Дуже перспективним напрямком в плані прискорення обчислення модулярної експоненти є організація реалізації піднесення до степені кратної двом, але більшої за два.

Таблиця 2.1 Приклад обчислення першої компоненти суми часткових добутків в формулі (2.11) для числа $A=11_{10} = 1011_2$

Номер біту *i*	Значення *i*–го біту числа А	Множене	Сума часткових добутків
1	1	10100	10100
2	1	10000	110100
3	0	00000	0110100
4	1	00000	00110100

Попередній аналіз практичної реалізації цього показує, що відповідна процедура піднесення до степені 4 потребує $n\cdot(n-1)/2$ операцій додавання при формуванні суми часткових добутків. Цілком очевидно, така кількість виконання операцій додавання не є прийнятною для задач практики. Більш детально дослідження можливості піднесення до степені більше двох може бути представлене у наступному вигляді:

Секції складаються з 2-х розрядів. Нехай виконується піднесення до 4-ї степені числа 9, яке займає дві секції тобто $A = a_2\cdot 2^2 + a_1$, де $a_2 = 2$, $a_1=1$. На першому етапі рахується $\alpha=a_1^2=1$; $\beta = a_1\cdot a_2 = 2\cdot 2^2 = 2^3$, $\gamma= a_2^2= 4\cdot 2^4 = 2^6$.

На другому етапі рахується $\alpha^2 = 1$, $\gamma^2 = 16\cdot 2^8 = 2^{12}$; $\beta^2 = 4\cdot 2^4 = 2^6$; $\alpha\cdot\beta = 2\cdot 2^2= 2^3$; $\gamma\cdot\beta = 2^6\cdot 2^3= 2^9$. Сума рахується у вигляді: 1 + 4096 + 64 + 8 + 512 =

$9^4 = 6561$. $A^2 = (\alpha+2\cdot\beta+\gamma) = 1+16+64 = 81$

Значення четвертої степені обчислюється у наступному вигляді:

$A^4 = (\alpha^2 + 4\cdot\alpha\cdot\beta + 2\cdot\alpha\cdot\gamma + 4\cdot\beta^2 + 4\cdot\beta\cdot\gamma + \gamma^2) = 1+4\cdot8 + 2\cdot2^6 + 4\cdot2^6 + 4\cdot2^3\cdot2^6 + 2^{12} = 6561$. Розглянемо обчислення експоненти числа $A = \{a_2,a_1,a_0\} = 2^2\cdot a_2 + 2\cdot a_1 + a_0$. При обчисленні модулярної експоненти через адитивне розкладання отримаємо наступну розрахункову формулу: $A^E \bmod M = (b+d)^E \bmod M = b^E + C^E_1\cdot b^{E-1}\cdot d + C^E_2\cdot b^{E-2}\cdot d^2 + \ldots + d^E$

Зокрема, за умови, що E = 3 можна ввести наступні позначення для складових числа A, що підноситься до експоненти: $A= b+d$, де d = e + f. Тоді

$(b+d)^3 = b^3 + 3\cdot b^2\cdot d + 3\cdot b\cdot d^2 + d^3 =$

$= b^3 + 3\cdot b^2\cdot e + 3\cdot b^2\cdot f + 3\cdot b\cdot e^2 + 6\cdot b\cdot e\cdot f + 3\cdot b\cdot f^2 + e^3 + 3\cdot e^2\cdot f + 3\cdot e\cdot f^2 + f^3 =$

$= 2^6\cdot b + 2^5\cdot 3\cdot b\cdot e + 2^4\cdot(3\cdot b\cdot f + 3\cdot b\cdot e) + 2^3\cdot(6\cdot b\cdot e\cdot f + e) + 2^2\cdot(3\cdot e^2\cdot f + 3\cdot b\cdot f^2) + 2\cdot 3\cdot e\cdot f + f$

Справедливість отриманого розкладання степені можна перевірити для випадку b=1, e=0, f=1. За цих умов $(b+e+f)^3 = 2^6\cdot b + 2^4\cdot 3\cdot b\cdot f + 2^2\cdot 3\cdot b\cdot f + f = 2^6\cdot b + 3\cdot b^2\cdot f + 3\cdot b\cdot f + f = 2^6 + 2^4\cdot 3 + 2^2\cdot 3 + 1$. Очевидно, що $(2^6 + 2^4\cdot 3 + 2^2\cdot 3 + +1) \bmod 19 = 11$. Це співпадає з прямим обчисленням $5^3 \bmod 19 = 11$.

Очевидна складність цього підходу при обчисленні модулярної експоненти ступені більшої ніж 2 полягає в тому, що:

- Потрібно перебрати всі можливі комбінації одиничних компонентів числа A
- Для кожної із комбінацій вибрати ваговий коефіцієнт
- Скористатися таблицями передобчислень степенів двійки
- Додати всі адитивні складові.

Як зазначалося вище, основна перепона полягає в практичному здійсненні першого пункту описаної послідовності дій. Виходить, що найбільш ефективний варіант – обчислення модулярного квадрату, а потім ще раз обчислення модулярного квадрату.

При класичній схемі піднесення до квадрату виконується звичайне

множення. Якщо вважати, що підноситься до квадрату 4-розрядне число A = $a_4 \cdot 2^3 + a_3 \cdot 2^2 + a_2 \cdot 2 + a_1$, то процес обчислення може бути відображено структурою, представленною на рис. 2.1.

		$a_4 \cdot a_1$	$a_3 \cdot a_1$	$a_2 \cdot a_1$	a_1
	$a_4 \cdot a_2$	$a_3 \cdot a_2$	a_2	$a_2 \cdot a_1$	
$a_4 \cdot a_3$	a_3	$a_2 \cdot a_3$	$a_1 \cdot a_3$		
a_4	$a_3 \cdot a_4$	$a_2 \cdot a_4$	$a_1 \cdot a_4$		

Рис.2.1 Структура обчислень квадрату при використанні схеми множення

Аналіз наведеної структури показує, що на бітовому рівні виконується 12 логічних операцій і 14 арифметичних операцій додавання. На рівні слів, виконується 4 операції логічного множення та 4 операції арифметичного додавання. З урахуванням особливостей операції піднесення до квадрату структура відповідних обчислень може бути оптимізована до вигляду, показаному на рис. 2.2.

	$a_4 \cdot a_1$	$a_3 \cdot a_1$	$a_2 \cdot a_1$	a_1
$a_4 \cdot a_2$	$a_3 \cdot a_2$		a_2	
$a_4 \cdot a_3$	a_3			
a_4				

Рис.2.2 Оптимізована структура обчислень квадрату

Аналіз оптимізованої структури показує, що на бітовому рівні здійснюється виконується 6 логічних операцій і 7 арифметичних операцій додавання. Тобто, кількість бітових операцій зменшується вдвоє. На рівні слів, виконується три операції логічного множення та відповідна кількість операції арифметичного додавання. З рисунка 2.2 чітко видно, що довжина слова, яке використовується на циклах множення скорочується: на першому циклі вона

дорівнює 5-ти, на другому циклі довжина слова становить вже 4, на третьому циклі в операції приймають участь лише два біта, а в четвертому – один.

В певному сенсі можна говорити, що на біт a_1 множиться 5-розрядний код a_4, a_3, a_2, 0, 1. Тоді в другому циклі на біт a_2 множиться код a_4, a_3, 0, 1, який складається з 4-х розрядів. Відповідно, на третьому циклі на наступний біт a_3 множиться код a_4,0, 1,

Якщо, відповідно до формули (2.11) рознести окремо процеси формування "розрідженого" двійкового коду Θ числа А та коду ϑ взаємодії розрядів числа, то останній утворюється як сума $a_1 \cdot (a_4, a_3, a_2) \cdot 2^2 + a_2 \cdot (a_4, a_3) \cdot 2^3 + a_4 \cdot a_3 \cdot 2^4$.

Якщо розрядність слова співпадає з розрядністю процесору, то ефект оптимізації операції піднесення до квадрату не відчувається, в силу того, що скорочення кількості розрядів в слові не призводить до прискорення обробки слова. Проте, такий ефект з'являється як тільки розрядність слова перевищує розрядність процесора. Саме така ситуація має місце в криптографічних застосуваннях для яких довжина слів дорівнює 2048, а розрядність процесора лежить в межах від 16-ти до 64-х.

Виконані дослідження дозволяють обґрунтувати і сформулювати процедуру роздільного піднесення до квадрату у вигляді наступної послідовності дій:

1. Встановити індекс j активного біту в одиницю: j=1.
2. Обнулити j–тий біт числа B; здійснити зсув числа B: B << = 1.
3. Якщо j-тий розряд числа А дорівнює нулю, тобто a_j=0, то перейти на п. 5. Інакше скопіювати число B число C: C= B, після чого j-тий розряд числа C встановити в одиницю c_j = 1.
4. Виконати додавання: R = R + C.
5. Якщо молодший біт проміжного результату R дорівнює одиниці, тобто r_1=1, то здійснити корекцію R шляхом додавання до нього модулю M: M=R+M.

6. Зсунути код проміжного результату R праворуч: $R = R/2$.

7. Виконати інкремент індексу j: $j=j+1$, якщо після цього j не перевищує n, тобто якщо $j \le n$, то перехід на повторне виконання п. 2.

8. Якщо отриманий результат R більший за модуль M або рівний йому, виконати віднімання від нього модуля: $R=R-M$.

Код числа B при значеннях $j>1$ завжди містить нуль в молодшому розряді. Це означає, що додавання числа B до проміжного результату R після першого такту не породжує одиниці в молодшому розряді коду проміжного результату R.

Якщо поточний j-тий розряд a_j числа А дорівнює одиниці, а наступний $(j+1)$-тий розряд a_j числа А дорівнює нулю, то можна виконати додавання до R коду B, корекцію по двох молодших розрядах коду R проміжного результату і зсув відразу на 2 розряди коду R.

Якщо поточний j-тий розряд a_j числа А дорівнює одиниці, а наступний $(j+1)$-тий розряд a_j числа А дорівнює одиниці, то можна виконати додавання до R коду B, корекцію по одному молодшому розряду коду R проміжного результату і зсув на один розряд коду R.

Для реалізації цієї ідеї можна застосувати передобчислення корегуючого коду, який являє собою добуток модуля на число, яке не перевищує h.

Запропонована процедура модулярного піднесення до квадрату $A^2 \bmod m$ включає в себе виконання наступної послідовності дій:

1. В змінну D копіюється значення числа A, що підноситься до квадрату: $D=A$. Індекс j поточного розряду множника встановлюється в одиницю: $j=1$, обнулюється значення R коду поточного результату: $R=0$.

2. Число D зсувається на один розряд ліворуч: $D = D<<1$.

3. В числі D обнуляється $(j+1)$-й розряд: $d_{j+1}=0$.

4. Якщо поточний j-тий розряд a_j числа дорівнює нулю, тобто якщо $a_j=0$,

виконується перехід на п.5, інакше число D копіюється в число C: $C=D$, після чого в j-тий розряд c_j числа копіюється значення відповідного розряду a_j числа A : $c_j = a_j$; сформований таким чином код С додається до поточного результату R: $R=R+C$.

5. Якщо молодший біт коду поточного результату R дорівнює одиниці, тобто $r_1=1$ то до нього додається код модуля m : $R=R+m$.

6. Код R поточного результату зсувається праворуч на один біт: $R= R>>1$.

7. Якщо індекс j поточного біту менший за n, тобто $j<n$, то він збільшується на одиницю, тобто: $j=j+1$, після цього здійснюється повернення на повторне виконання п. 3.

8. Якщо код R поточного результату більший або рівний модуля m, то від коду R віднімається значення модуля m: $R=R-m$.

Подальше прискорення пов’язане з використанням зсуву коду С , тобто з додаванням на два зсуви відразу двох доданків.

Для криптографічних застосувань модуль при обчисленні модулярної експоненти являє собою частину відкритого ключа учасника віддаленої взаємодії. Відповідно, чисельне значення модуля змінюється рідно. Це означає, що для незмінного модуля, розроблений алгоритм може бути спрощений за рахунок використання передобчислень, що залежать тільки від модуля. Зокрема, значення поправки перед зсувом суми часткових добутків залежать тільки від значення молодших розрядів суми часткових добутків і при малих значеннях k можуть бути заздалегідь обчислені для всіх 2^k можливих значень k молодших розрядів у вигляді $T1(r_k)$ із збереженням результатів у табличній пам'яті. При цьому значення $T_1(r_k)$ обчислюється відповідно до виразу:

$$T_1(r_k) = r_k + M \cdot (r_k \cdot M' \bmod 2^k). \qquad (2.12)$$

Обмеження на кількість розрядів, обробка яких суміщується у часі визначаються граничним об’ємом пам'яті для таблиць передобчислень.

2.3 Організація розпаралелювання операції модулярного піднесення до квадрату на багатоядерних обчислювальних платформах

Обчислення операції модульного множення чисел, розрядність яких перевищує розрядність модуля $I \cdot J \bmod K$ включає дві частини: множення компонентів $M=I \cdot J$ та знаходження залишку від результату після ділення добутку $I \cdot J$ на модуль K.

Наступна організація паралельних обчислень має на меті досягнення конкурентного результату знаходження модулярного добутку $I \cdot J \bmod K$ у форматі у незалежних операцій обчислення. Для описаних обчислювальних процесів, відповідно, потрібно у ядер. Це передбачає, що L-бітний множник $I = i_1 + i_1 \cdot 2^2 + i_2 \cdot 2^2 + \cdots + i_L \cdot 2^L,\ \forall d \in \{1,2,\dots,L\}: id \in \{0,1\}$, розкладається на у частинних множників $I_1, I_2, \dots, I_y$ з L бітами. Кожен з множників I_c розрядністю L-біт складається з тих $g=L/y$ цифр множника L, остача відділення на у, у яких рівна с, залишкові цифри часткового множника рівні нулю. Тобто, пояснюючи іншими словами, для кожного d-того частинного множника K_d можна записати таке представлення:

$$I_c = \sum_{d=c}^{(g-1)y} a_d \cdot 2^d. \qquad (2.13)$$

Для кращого розуміння можна навести такий конкретний випадок, нехай глибина бітів L=12, кількість незалежних розрахункових операцій становить у=3, то множник $I = 0011\ 1001\ 1100_2 = 924_{10}$, розбивається на три частинні множники, кожен з який складається з $f = L/y = 4$ значущих двійкових бітів вхідного множника I. Отже, перший частковий множник, а саме I_0 містить в собі кожну четверту цифру, рахуючи від найменш значущої, відповідно: 1-ий, 4-ий та 7-ий біти вхідного множника I: $I_1 = 0010\ 0000\ 1000_2$. Та, відповідно, другий не повний множник I_1 складається з 2-го, 5-го та 8-го знаку вхідного множника I: $I_2 = 0000\ 1001\ 0000_2$ Останній, третій частковий множник –

$I_3 = 0001\ 0100\ 0100_2$.

Звідси докладно зрозуміло, що диз'юнкція усіх окремих множників рівна множнику I операції модулярного множення $I_1 \vee I_2 \vee ... \vee I_y = I$, тоді кон'юнкція часткових добутків рівна нулю $I_1 \wedge I_2 \wedge ... \wedge I_y = 0$. Звідси очевидно, що модулярний добуток *I·J mod K* – це, відповідно, сума модулярних часткових множників та множника J :

$$I \cdot J \bmod K = (\sum_{c=1}^{y} I_c \cdot J \bmod K) \bmod K. \qquad (2.14)$$

Отже, диференціація значимих розрядів операнду *I*, на множники $I_1, I_2, \dots, I_y$ які незалежно виконують операцію знаходження модулярного добутку на множник J і є частковими гарантує підвищення швидкості виконання обчислень модулярного множення за допомого розпаралелювання процесу модульного знаходження добутку. Зважаючи на те, що в часткових множниках, які незалежно множаться на множник *J*, є значна перевага кількості нулів, це створює позитивні умови для ефективного перевикористання обчислень, що вже зроблені в процесі знаходження залишків від ділення за модулем при використанні алгоритму Монтгомері [2].

Алгоритм Монтгомері базується на прнципі заміни обчислення модулярного залишку від ділення на модуль *K* обчисленням залишку від ділення на модуль *E*, який є степенем двійки, з метою зменшення операцій ділення до операцій зсуву.

Важливо пам'ятати, що при розрахунку $I \cdot J \bmod K$ слід брати до уваги також складність перед- та післяобчислень. Якщо зважати на складність обчислення операції модулярного множення з корекцією, що потребує попередньо операції знаходження модульного добутку на обернений до *2L* модуль *K* і розрахунок його добутку із результатом за допомогою алгоритму Монтгомері, то і кінцевому результаті його складність буде *4K·(K+1)*. Це означає, що при виконанні однієї операції модулярного множення алгоритм

Монтгомері не має очевидних переваг над базовим алгоритмом.

Тож, зробивши аналіз особливих властивостей цих алгоритмів, можна сформулювати вимоги до концептуально нової схеми організації часткових обчислень модулярного добутку. Зважаючи на усі загальні принципи, які є наріжним каменем алгоритму Монтгомері (першочергово, зменшення розміру всіх тимчасових результатів шляхом заміни їх на еквівалентні числа, які мають ту саму залишкову величину від ділення на заданий модуль), існує можливість реалізації обчислення разом з їх обов'язковим збереженням, що дозволяє отримати високоефективний прикладний алгоритм паралельного модулярного множення. За збереження всіх умов, отримання результату модулярного знаходження добутку цикл має бути пройдений лише один раз, іншими словами організація обчислень повинна бути однопрохідною. Алгоритм Монтгомері в класичній інтерпритації не виконує цієї умови, оскільки він виконується в обидві сторони. Тому в процесі формування результату, що грунтується на результаті часткових знаходжень добутків потрібно досягати максимального навантаження ядер процесора.

Як вже було згадано раніше, перевагою описаного методу ділення множника I на часткові множники $I_1, I_2, \dots, I_y$ є наявність груп нулів, які кількісно переважають в складі числа і мають загальну кількість не менше y. Тож за таких умов існує можливість вирішити проблему прискорення обрахунку модулярного добутку за допомогою сумування лінійної комбінації та проміжного результату, вибраного так, щоб нижні у цифр суми часткового результату та цієї лінійної комбінації $M+P(K)$ в кінцевому результаті давали нуль, на противагу додаванню модуля до проміжного результату. Згідно алгоритму, одразу після суму лінійної комбінації та проміжного результату зсувають праворуч на у біт, обов'язково без втрати значущих бітів. Таке вирішення проблеми дозволяє прискорити редукцію суми лінійної комбінації та проміжного результату в у разів одночасно. Прикладна реалізація вище

описаної технології прискореного обрахунку остачі від ділення проміжного результату на модуль *K* вимагає в обов'язковому порядку забезпечити заздалегідь для кожного з 2^y можливих варіантів значень найменш значущих у цифр проміжного результату M, що містить в собі значення лінійної комбінації модуля *P(K)*, у найменш значущих цифр якого є алгебраїчним доповненням для у найменш значущих цифр проміжного результату. Результати проведених перед- та післяобчислень зберігаються у вигляді таблиці перед- та/або післяобчислень.Нижче наведено приклад таблиці *T* передобчислень *P(K)* для *y=4* і значення модуля *K= 17·31 = 527* у таблиці.

Таблиця 2.2 Приклад попередніх обчислень лінійних комбінацій P(m) для модуля *K*= 17·31 = 527 та *S*=4

Молодші розряди M	*P*(K)	Молодші розряди P(K)
$m_4m_3m_2m_1$		$p_4p_3p_2p_1$
0 0 0 1	527 = K	1 1 1 1
0 0 1 0	1054 = 2·K	1 1 1 0
0 0 1 1	1581 = 2·K + K	1 1 0 1
0 1 0 0	2108 = 4·K	1 1 0 0
0 1 0 1	2635 = 4·K + *K*	1 0 1 1
0 1 1 0	3162 = 4·K + 2·K	1 0 1 0
0 1 1 1	3689 = 4K + 2K + K	1 0 0 1
1 0 0 0	4216 = 8K	1 0 0 0
1 0 0 1	4743 = 8K + K	0 1 1 1
1 0 1 0	5270 = 8K + 2K	0 1 1 0
1 0 1 1	5797 = 8K + 2K + K	0 1 0 1
1 1 0 0	6324 = 8K + 4K	0 1 0 0
1 1 0 1	6851 = 8K + 4K + K	0 0 1 0
1 1 1 0	7378 = 8K + 4K + 2K	0 0 1 0
1 1 1 1	7905 = 8K + 4K + 2K + K	0 0 0 1

Повна розрядність пам’яті таблиці для зберігання проміжних результатів *P(K)* становить $L \cdot 2^y$ бітів. Таблицю, представлену вище, де зображений спосіб зберігання *P(K)* в одному місці, можна розглядати як окремий випадок розділеної організації результатів проміжних обчислень. За умови використання мультисекційних таблиць можна значно мінімізувати кількість потрібної для зберігання пам’яті. У вище реалізованому прикладі відбувається прискорене множення чисел ємністю 2048 бітів на мікроконтролері з двосекційною пам’яттю та розрядністю 8 біт, затрати пам’яті у 8,57 разів менше, ніж її затребуваність при односекційній організації пам’яті таблиць. Проте, використання мультисекційної організації пам’яті таблиць тягне за собою сповільнення ефективності виконання молярного зменшення.

Метод модулярного знаходження добутку з одночасними паралельними обчисленнями на у ядрах процесора передбачає виконання операцій обчислення часткових добутків паралельно на всіх ядрах, а після цього - їх каскадне модулярне додавання з метою скорочення часу генерації отримання кінцевого результату модулярного множення.

Цей конкретний випадок передбачає виконання процедури обчислення часткового модулярного добутку в послідовному проходженні наступного алгоритму:

1. Умовний лічильник в циклі - *t* обнуляємо, також встановлюємо значення 0 для поточного результату *M: h = 0; Y = 0*.
2. Зсуваємо частковий множник I_g праворуч на *g-1* розряд (двійковий): $I_g = I_g \gg (g-1)$, при цьому більш значущі розряди заповнюються нулями.
3. Якщо найнижчий бінарний розряд часткового добутку M рівний одиниці: $m_1 = 1$, То множник J додається до результуючого коду *M+=J*.

4. Коли значення лічильника циклів *t* стає кратним y, пропустити крок 5, та звернутись до пункту 6.
5. Табличний код *H[n]*, адресований у нижчими розрядами коду результату потрібно просумувати з кодом часткового добутку *M*, у нижчими розрядами коду результату $n = m_1 + 2 \cdot m_2 + \cdots + 2^{y-1} \cdot y_s : M += H[n]$.
6. Результуючий код *M* та множник *J* зсуваються праворуч на y розрядів: *M=M>>Y*. Лічильник циклів t алгоритму збільшується на одиницю: $t{+}{+}$, та виконується звертання до третього пункту алгоритму повторно.
7. До коду часткового добутку M додається табличний код $H[n]$, адресований y-g менш значущими розрядами коду результату

$n = m_1 + 2 \cdot m_2 + \cdots + 2^{y-g-1} \cdot y_{s-r+1} : M += H[n]$.

8. Код результату *M* зсувається праворуч на $y\text{-}g$ розрядів: $M = M >> y\text{-}g$.
9. Кінець.

Після виконання описаної процедури, *M* згенерувала код модулярного добутку, що містить $I \cdot J \cdot E^{-1} \bmod K$, де E^{-1} є мультиплікативним оберненням $E=2^l$ за модулем *K*. Для досягнення правильного результату, отриманий код *M* потрібно модулярно помножити на E: $M' = E \cdot M \bmod K$. Однак при виконанні операції модулярного множення як складової модулярного експоненціювання, виправлене множення кодом *K* виконується лише один раз, після виконання всіх циклів класичного алгоритму модулярного піднесення до степеня.

Ефективність запропонованого методу модулярного множення можна оцінити через досягненне прискорення обчислювальної реалізації цієї операції при використанні у ядер процесора. Числовим виразом для оцінки прискорення може бути коефіцієнт z, який визначається співвідношенням часу h_1 для виконання модулярного множення у вигляді одного процесу з використанням зведення Монтгомері до часу h_s для виконання цієї операції у вигляді у

паралельних процесів з використанням розробленого методу:

$$z = \frac{h_1}{h_s}. \quad (2.15)$$

Кількість часу h_1 знаходження модулярного добутку з використанням одного процесора та послідовності виконання циклів множення та редукції проміжного результату визначається часом виконання l циклів для чисел конкретної довжини. Кожен цикл включає додавання множника J до коду проміжного результату M, в залежності від поточного розряду множника I, а в подальшому і в залежності від значення меншого значущого розряду отриманої суми M, може виконуватись або - ні додавання модуля K до коду попереднього результату обчислення M. Останній етап – зсув коду проміжного результату на один біт праворуч. Отже, за таких умов, середня кількість виконаних операцій з l-розрядними числами у циклі дорівнює двом. Тобто, якщо для часу виконання вибрати позначення h_l то $h_1 = 2 \cdot L \cdot h_l$.

В описаний розроблений алгоритм вкладено реалізацію обчислення добутку множника на проміжний результат з використанням L/y циклів, кожен з який складається з наступних кроків: сумування множника з проміжним результатом, якщо найнижчий двійковий біт часткового множника рівний одиниці, та сумування результату з кодом із таблиці попередніх обчислень, і на сам кінець – зсув фінального результату вправо.

Згідно з дослідженнями, середня кількість операцій дорівнює 3,5 для l-розрядних чисел. Звідси, значення $h_k = 3.5 \cdot L \cdot h_l/y$. Чисельно значення коефіцієнта прискорення обчислюється із наступної формули:

$$z = \frac{h_1}{h_s} = \frac{2 \cdot L \cdot h_L}{3.5 \cdot h_L \cdot \frac{L}{y}} \approx 0.57 \cdot y. \quad (2.16)$$

Експерименти на багатоядерних процесорах за допомогою спеціально розробленої програми показали значення коефіцієнта прискорення, що

дорівнює $0.5 \cdot K$, близьке до передбаченої теоретичної оцінки.

Інший підхід до покращення ефективності обрахунків полягає в одночасному аналізі групи у бітів множника J та групову редукцію Монтгомері. Це рішення має прикладний характер та практично використовується в комп'ютерній арифметиці як один із способів прискорення операції множення на процесорі.

Основоположна ідея полягає в тому, що одночасна обробка у розрядів множника виконується за одну ітерацію циклу. Тобто, множник J розбивається на $a = L/y$ секцій $s_a, s_{a-1}, \dots, s_2, s_1,\ \forall c \in \{1,2,\dots,a\}: 0 \le s_a \le 2^y - 1$. Операція знаходження модульного добутку виконується за допомогою а кількості циклів, в кожному з яких виконується множення I на відповідний фрагмент множника J. Опісля до цього результату додається попередньо обчислене значення, яка обраховується за модулем E цілим числом, таким, щоб менш значущі у розрядів суми були рівні нулю. Наступним кроком застосовується праве зсування на у біт суми проміжних добутків. Кінцеве значення у визначається на основі обсягу пам'яті, яку можна виділити для зберігання таблиць проміжних результатів, а також згідно з кількістю самих операцій знаходження проміжних результатів.

Звідси випливає, що загальна кількість бітів поточного коду результату , залежно від комбінації яких визначається тип операції, що реалізує додавання множників та корекцію групи Монтгомері рівна:

$$B = 2^{3 \cdot y}. \tag{2.17}$$

Оскільки кожен набір операцій, який потрібно виконати залежить від трьох компонент : множника J з у бітами, у менших значущих бітів множника I, та у менших значущих бітів поточного коду результату. Згідно з тим, що кожен рядок таблиці проміжних результатів містить код з l бітів, загальний обсяг таблиці можна знайти з наступної рівності:

$$Q = 2^{3 \cdot y} \cdot l. \tag{2.18}$$

Однак, об’єм пам'яті таблиці для зберігання проміжних обчислень не є чільним критичним обмеженням для збільшення кількості бітів у множника, обробка яких суміщається в часі. Зі збільшенням у кількість попередніх обчислень, які необхідно виконати перед фактичним модульним множенням, швидко зростає. Під час попередніх обчислень необхідно знайти суму двох компонент - певного піднабору зсунутих кодів множника *I* та певного піднабору зсунутих кодів модуля *E*. Очевидно, що кількість варіантів вибору кожного з піднаборів дорівнює $2^y - 1$. Наприклад, під час аналізу двох бітів множника *y=2* може бути додано один незсунутий код *I*, один зсунутий вліво код множника *I* та обидва: зсунуті і незсунуті коди *I*.

Отже, загальна кількість L сум для реалізації передобчислень:

$$L = (2^y - 1)^2. \tag{2.19}$$

Отже, загальна кількість L операцій додавання l-бітових кодів, необхідних для виконання модулярного множення, може бути обчислена за формулою:

$$L_0 = \frac{l}{y} + L = \frac{l}{y} + (2^y - 1)^2. \tag{2.20}$$

Похідна загальної кількості операцій додавання наведена в формулі:

$$\frac{dL_0}{dy} = -\frac{l}{y^2} + 2^{y+1} \cdot (2^y - 1) \cdot ln\, 2. \tag{2.21}$$

Числове значення коефіцієнта прискорення γ, яке досягається за рахунок накладання обробки кількох бітів множника та комбінації зменшення Монтгомері, може бути подано наступною формулою:

$$\gamma = \frac{l}{L_0} = \frac{y}{1+\frac{y}{l}\cdot(2^y-1)^2}. \tag{2.22}$$

Очевидно, що зі збільшенням кількості розрядів у множника, обробка якого перекривається в часі, об'єм попередніх обчислень швидко зростає.

Висновки до розділу 2

В результаті проведених досліджень, направлених на прискорення базової для модулярного експоненціювання операції модулярного піднесення до квадрату отримані наукові та практичні результати, які можуть бути сформульовані наступним чином:

1. Показано, що основними резервами підвищення швидкодії комп'ютерної реалізації базової процедури широкого кола алгоритмів криптографічного захисту інформації є виключення операційної надмірності виконання двох базових операцій – модулярного піднесення до квадрату та модулярного множення на ціле число.

2. Теоретично доведено, що за рахунок виключення операційної надмірності при обчислення модулярного квадрату можна гранично прискорити обчислювальну реалізацію цієї операції в два рази. Доведено, що ефективне використання цього резерву прискорення обчислення модулярної експоненти може бути реалізоване лише за у мови використання спеціального методу прискореного модулярного множення на постійне число.

3. Теоретично обґрунтовано, запропоновано та досліджено метод прискореної обчислювальної реалізації операції модулярного піднесення до квадрату, який відрізняється тим, що довжина множника зменшується на кожному циклі, за рахунок чого досягнуто зменшення кількості операцій процесорного множення і, тим самим забезпечено прискорення виконання цієї операції.

4. Запропонований метод пристосовано для ефективної реалізації модулярної редукції з використанням технології Монтгомері з використанням передобчислень, які залежать тільки від модуля і, відповідно, для криптографічних застосувань можуть проводитися заздалегідь і тільки один раз, оскільки модуль є частиною відкритого ключа.

5. Розроблено і досліджено алгоритм прискореного модулярного

піднесення до квадрату, який базується на запропонованому методі та використанню модулярної редукції Монтгомері і відрізняється тим, що за рахунок одночасної обробки декілька розрядів множника, дозволяє досягти подальшого прискорення обчислювальної реалізації операції модулярного піднесення до квадрату. Показано, що обмеження на кількість розрядів, обробка яких суміщується у часі визначаються граничним об’ємом пам'яті для таблиць передобчислень.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ТЕСТУВАННЯ ТА ДОСЛІДЖЕННЯ МЕТОДУ ПРИСКОРЕНОГО МОДУЛЯРНОГО ПІДНЕСЕННЯ ДО КВАДРАТУ

Важливою компонентою дослідження реальної ефективності запропонованого методу прискореного модулярного піднесення до квадрату виступає експериментальна частина.

Для того, щоб найбільш повно реалізувати потенціал запропонованого методу в плані прискорення виконання модулярного піднесення до квадрату, розробка програми здійснена на мові програмування C++. Ця мова програмування дозволяє ефективно керувати процесорними засобами і пам’яттю, за рахунок чого можна досягти найбільшого виграшу в швидкодії за рахунок виключення дублювання операцій на бітовому рівні.

Основною метою розробки програми є експериментальна перевірка працездатності запропонованого методу прискореного модулярного піднесення до квадрату, а також визначення реальних показників прискорення обчислення модулярного квадрату. Важливим є також отримання залежностей показників швидкодії від розрядності процесора та довжини чисел, які використовуються в рамках криптографічного протоколу. Розроблена програма має забезпечувати гнучко змінювати довжину слів та емулювати різну розрядність процесора, що потрібно для отримання залежностей характеристик ефективності методу від параметрів його реалізації.

Представляється також вагомим визначення експериментальним шляхом доцільності поєднання операції піднесення до квадрату та модулярної редукції. За відсутності такого поєднання операції піднесення до квадрату та модулярної редукції виконуються послідовно і не залежать одна від одної. Перевагою такого варіанту є те, що повною мірою використовуються апаратні засоби прискореного процесорного множення. Негативна сторона полягає в тому, що розрядність суми часткових добутків стає вдвічі більше, що уповільнює процес

модулярної редукції. Тому потрібно експериментально дослідити час виконання модулярного піднесення до квадрату для обох випадків для різних значень розрядності процесора та довжини чисел, які застосовуються в протоколі криптографічного захисту. При цього для обох зазначених вище випадків використовується модулярна редукція Монтгомері. Для проведення вказаних експериментальних досліджень потрібно розробити програму, яка здійснює секційне піднесення до квадрату з використанням процесорної операції множення та подальшою модулярною редукцією отриманого квадрату за методом Монтгомері.

3.1 Організація даних

Організація структур даних, які використовуються в програмі має забезпечити високу ефективність досягнення зазначених вище цілей створення програми, а також простоту їх написання.

В розробленій програмі зберігаються дві функціональні компоненти операції модулярного піднесення до квадрату: число A, що підноситься до квадрату та модуль M. Для зберігання цих чисел в програмі виділяються два масиви по 512 байтів, які можуть з допомогою зміни типу вказівника адресуватися в різних форматах: 16-розрядних слів, 32-розрядних подвійних слів або як байти (при моделюванні роботи 8-розрядного термінального мікроконтролера). Фактично, ці два байтових масива зберігаються в класі NUM. Крім того, в цьому ж класі організовано массив, що складається із 512 байтів для зберігання суми часткових добутків.

Особливістю виконаної розробки є використання змінних форматів множника: реально його довжина скорочується на 2 біти при виконанні кожного кроку модулярного піднесення до квадрату. Саме ця особливість розробки являє собою вирішальний чинник прискорення піднесення до квадрату: при додаванні багаторозрядних чисел до суми часткових добутків, їх довжина постійно скорочується. Відповідно, потрібно здійснювати менше

процесорних операцій додавання зсунутого множника суми часткових добутків. Для врахування довжини коду множника при виконанні операції модулярного піднесення до квадрату, в програмі використовується спеціальний байтовий масив M маски. В кожен поточний момент часу код маски одиницями відмічає ті розряди множимого, які приймають участь в операції додавання до суми часткових добутків.

Крім наведених масивів, в яких зберігаються числа, що приймають участь в виконанні операції модулярного піднесення до квадраті, в розробленій програмі використовуються результати передобчислень для прискорення виконання модулярної редукції отриманого результату. На відміну від існуючих схем передобчислень [17] в розробленій програмі використовуються передобчислення, які адресуються групою молодших розрядів коду суми часткових добутків. Таке рішення зумовлене тим, що, як зазначалося раніше, довжина операндів і розробленому методі прискореного модулярного піднесення до квадрату зменшується з кожним циклом виконання цієї операції.

3.2 Розробка функцій і методів

Для програмної реалізації запропонованого методу модулярного піднесення до квадрату створено ряд програм і програмних модулів.

Модуль формування таблиці передобчислень реалізує формування таблиці, кожним елементом якого є результат множення модуля на число k , яке визначається наступним чином: $k=2^h-1$, де h- загальна кількість розрядів, що оброблюються одночасно. Відповідно, це означає, що одночасно оброблюються $h/2$ розрядів суми часткових добутків та така ж кількість молодших розрядів множника. Кожна з комірок таблиці передобчислень має містити код, який, при його додаванні до суми часткових добутків з урахуванням значень молодших розрядів множника сформує результат, який містить h нулів в молодших розрядах нового значення суми часткових добутків. Для формування таблиці передобчислень, враховуючи її порівняно невеликий об’єм використано

перебір: це означає, що модуль послідовно множиться на 1,2,…: аналізуються молодші розряди утвореного добутку – від числа 2^h віднімається значення молодших h розрядів. Отриманий залишок визначає номер комірки, в яку записується значення добутку.

Модуль обчислення модулярної інверсії використовується в розробленій програмі для форкання поправочного множника, який використовується в методі модулярної редукції Монтгомері для отримання коректного результату.

Модуль модулярного піднесення до квадрату – є основним методом класу. Цей метод передбачає послідовне виконання визначеної кількості операційних циклів. В кожному циклі аналізується значення поточного розряду множника: якщо від дорівнює одиниці, то на множене накладається код маски і вона додається до суми часткових добутків. Вузловим моментом при написанні цієї функції є визначення адрес фрагментів суми часткових добутків, які приймають участь в операції додавання. Як було показано вище, кількість фрагментів поступово зменшується від одного до циклу до наступного. Саме це є дієвим чинником зменшення кількості процесорних операцій. Так, на першому циклі модулярного піднесення до квадрату в операції приймають участь 128 фрагментів (при довжині числа – 2048 і розрядності мікроконтролера 16), а на останньому циклі в операції приймає участь лише один фрагмент суми часткових добутків.

Для зберігання результату модулярного піднесення до квадрату функції зарезервований бітовий набір R. На початку обчислень значення молодшого розряду бітового набору R встановлюється в одиницю. В якості першого кроку, передбачає обчислення таблиць передобчислювання T і W, описаних у запропонованому алгоритмі створюється структура даних S типу RA розміром $n \cdot 2d-1$ елементів. Потім виконується заповнення елементів структури S, шляхом організації циклу від 0 до $2d-1$ та організації обчислень значень елементів структури всередині цього циклу.

Результатом роботи функції є значення яке містить бітовий набір R, який

являє собою поточне значення суми часткових добутків.

Для виконання порівняльного аналізу розроблено методу та відомим методом молярного піднесення до квадрату, в рамках програми розроблено модуль роздільного обчислення квадрату та редукції отриманого результату. В цій функції використовується секційне множення: число розбивається на секцій, довжина яких дорівнює розрядності мікроконтролеру. В цій функції виключена притаманна піднесенню до квадрату операційна надмірність. Отриманий результат – масив довжиною $2 \cdot n$ бітів формується у явному вигляді і зберігається в описаному вище байтовому масиві. Для формування квадрату використовується операція процесорного множення. яка виконується над кодами, довжина яких дорівнює розрядності процесора. Це забезпечує високу швидкість формування квадрату, оскільки повною мірою використовується потужність процесора. Модулярна редукція здійснюється побітово за методом Монтгомері. Тобто ця операція виконується за $2 \cdot n$ циклів і операція виконується на довгими числами відповідної розрядності.

3.3 Експериментальне дослідження запропонованого методу прискореного модулярного піднесення до квадрату з використанням розробленої програми

Основним чинником ефективності запропонованого методу модулярного піднесення до квадрату є прискорення виконання цієї важливої для криптографічних застосувань операції. Відповідно, задачею експериментального дослідження ефективності запропонованого методу є визначення часових характеристик реалізації запропонованого методу в різних режимах, а також порівняння отриманих часових характеристик з базовим варіантом. В якості базового варіанта обрано розподілену реалізацію операцій піднесення до квадрату та модулярної редукції за методом Монтгомері. При цьому в базовому варіанті передбачається, що операція піднесення до квадрату виконується шляхом додавання часткових множників процесорних секцій

числа. Зрозуміло, що розрядність секцій дорівнює розрядності процесора. В результаті виконання такої операції утворюється 2-n розрядний код квадрату числа. Модулярна редукція отриманого добутку в базовому варіанті здійснюється за методом Монтгомері. При цьому в кожному циклі обробки один раз відбувається редукція з розрядом, відповідно загальна кількість циклів редукції Монтгомері складає *n* циклів.

З огляду на те, що реально застосовуються мікроконтролери, які мають від 8 до 32 розряди, є значний інтерес дослідити залежність часу обчислення модулярного квадрату для різних значень розрядності (8, 16, 32). У представленому дослідженні аналізуються три можливі варіанти реалізації модулярного піднесення до квадрату:

- Найпростіший варіант - суміщена обробка, при якій множення і модулярна редукція виконуються порозрядно, тобто загальна кількість операцій становить *n*.
- Варіант суміщеної обробки двох розрядів множника одночасно. У цьому варіанті редукуються одночасно два розряди суми часткових добутків. Для коректної редукції потрібно використовувати 15 лінійних комбінацій модуля, які вибираються в залежності від двох молодших розрядів коду суми часткових добутків та двох молодших розрядів множника. В цьому варіанті кількість циклів дорівнює *n*/2.
- Варіант суміщеної обробки чотирьох розрядів множника одночасно. У цьому варіанті редукуються одночасно чотири розряди суми часткових добутків. Для коректної редукції потрібно використовувати 255 лінійних комбінацій модуля, які вибираються в залежності від чотирьох молодших розрядів коду суми часткових добутків та чотирьох молодших розрядів множника. В цьому варіанті кількість циклів дорівнює n/4.

Результати проведених експериментальних досліджень для довжини чисел рівних 1024 представлено в графічному вигляді на рис. 3.1

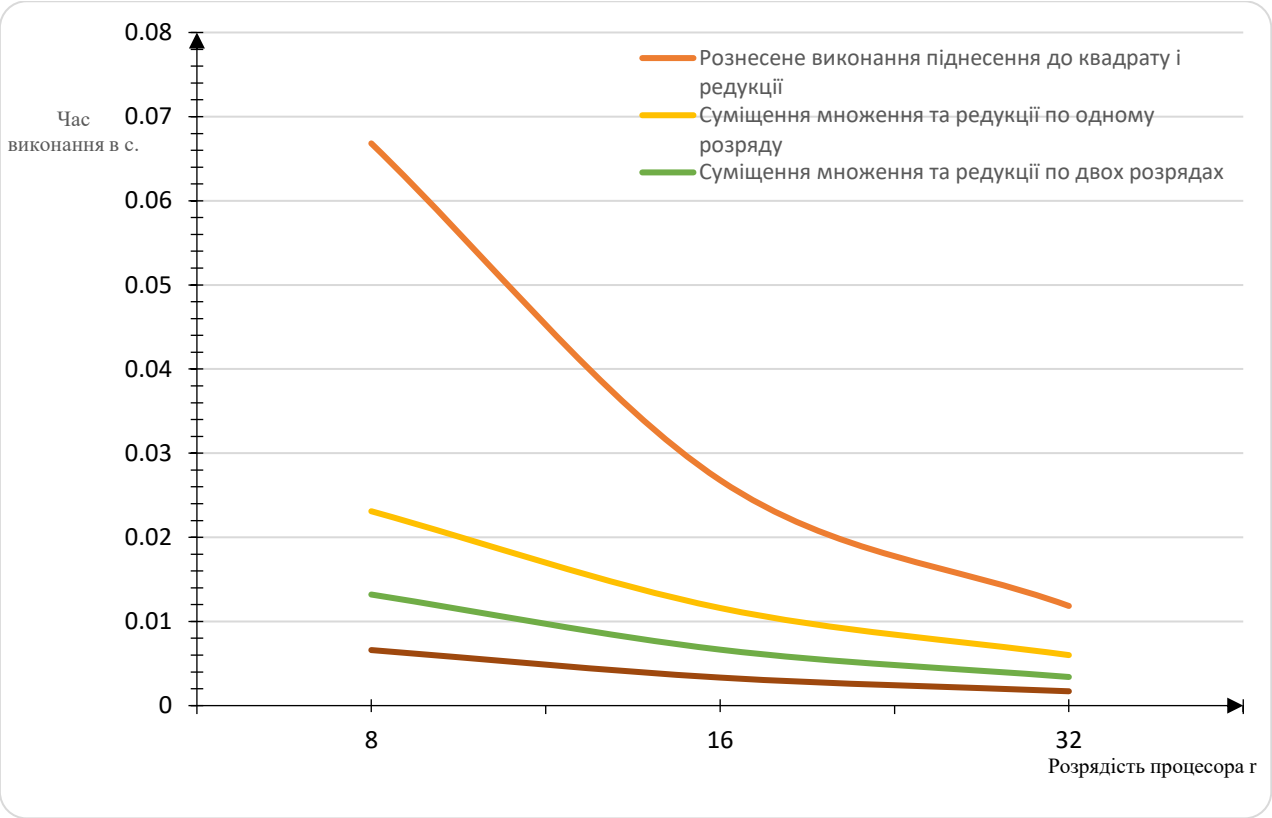


Рис 3.1 Залежність часу виконання модулярного піднесення до квадрату від розрядності мікроконтролера для 1024-розрядних чисел.

На графіку представлено 4 залежності часу виконання програми від розрядності процесора для базового варіанту і трьох можливих реалізацій запропонованого методу. З графіків видно, що для всіх розрядностей процесора використання запропонованого методу модулярного піднесення до квадрату дозволяє підвищити швидкодію виконання цієї важливої для криптографічних застосувань операції. Основними чинниками досягнутого прискорення в порівнянні з базовим варіантом є:

1. Виключення операційної надлишковості при виконанні операції модулярної редукції.
2. Виконання операції над значно меншою кількістю розрядів, що зумовлено тим, що формування суми часткових добутків і редукція організовані одночасно. До цього слід додати, що кількість розрядів, які задіяні в операції поступово зменшується.

На рис.3.2. представлено графік залежностей коефіцієнту прискорення виконання операції модулярного піднесення до квадрату в порівнянні з рознесеним множенням та редукцією від розрядності мікроконтролера для чисел довжиною 1024.

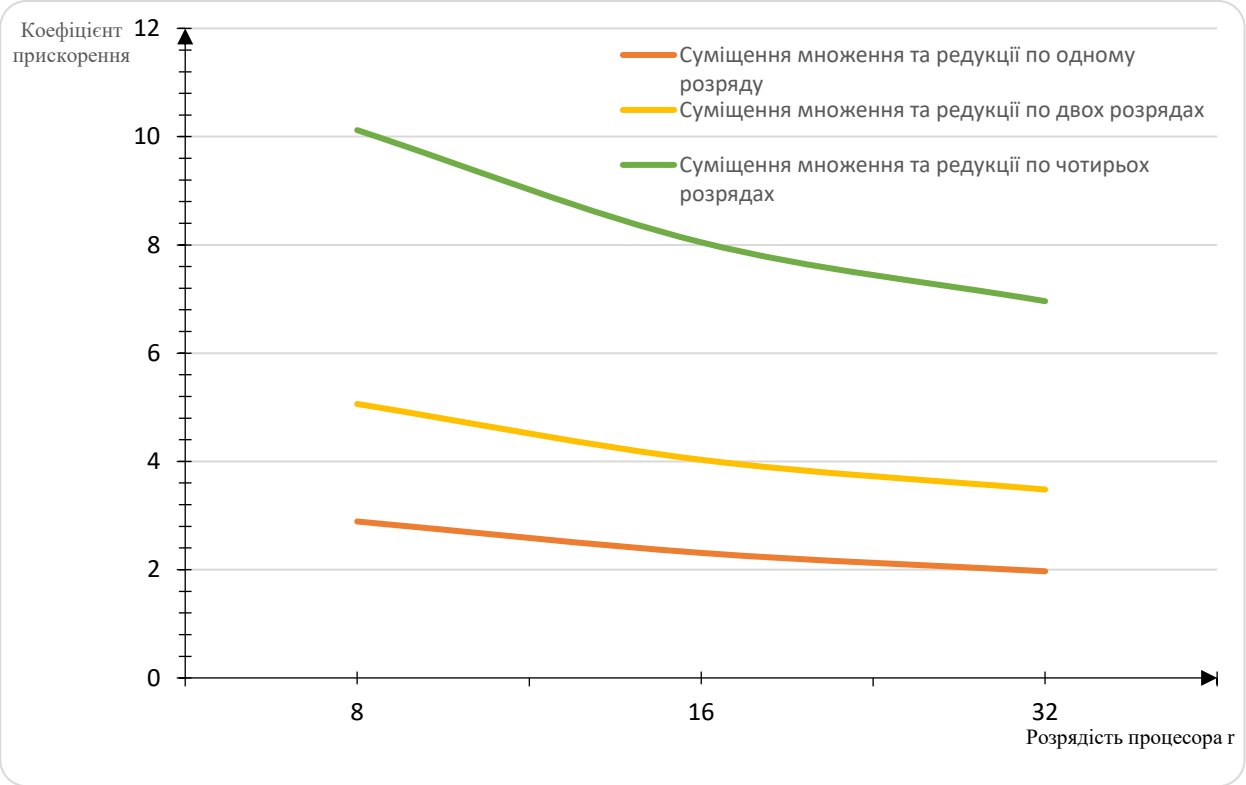


Рис 3.2 Залежність коефіцієнту прискорення модулярного піднесення до квадрату в порівнянні з рознесеним множенням та редукцією від розрядності мікроконтролера для чисел довжиною 1024.

Аналіз отриманих залежності показує, що прискорення виконання модулярного піднесення до квадрату досягає найбільших значень при малих значеннях розрядностей мікроконтролера. Зі збільшення розрядності процесора, ефективність запропонованого методу з точки зору прискорення обчислень зменшується.

З графіків, представлених на рис.3.2 видно, що навіть в напростішому варіанті, тобто без використання передобчислень, суміщення множення та модулярної редукції забезпечує прискорення виконання модулярного

піднесення до квадрату практично в два рази. Такий результат співпадає з виконаними в другому розділі теоретичними оцінками.

Важливим аспектом оцінки ефективності запропонованого методу в плані підвищення швидкодії є дослідження впливу розрядності чисел.

Результати проведених експериментальних досліджень для довжини чисел рівних 2048 представлено в графічному вигляді на рис. 3.3.

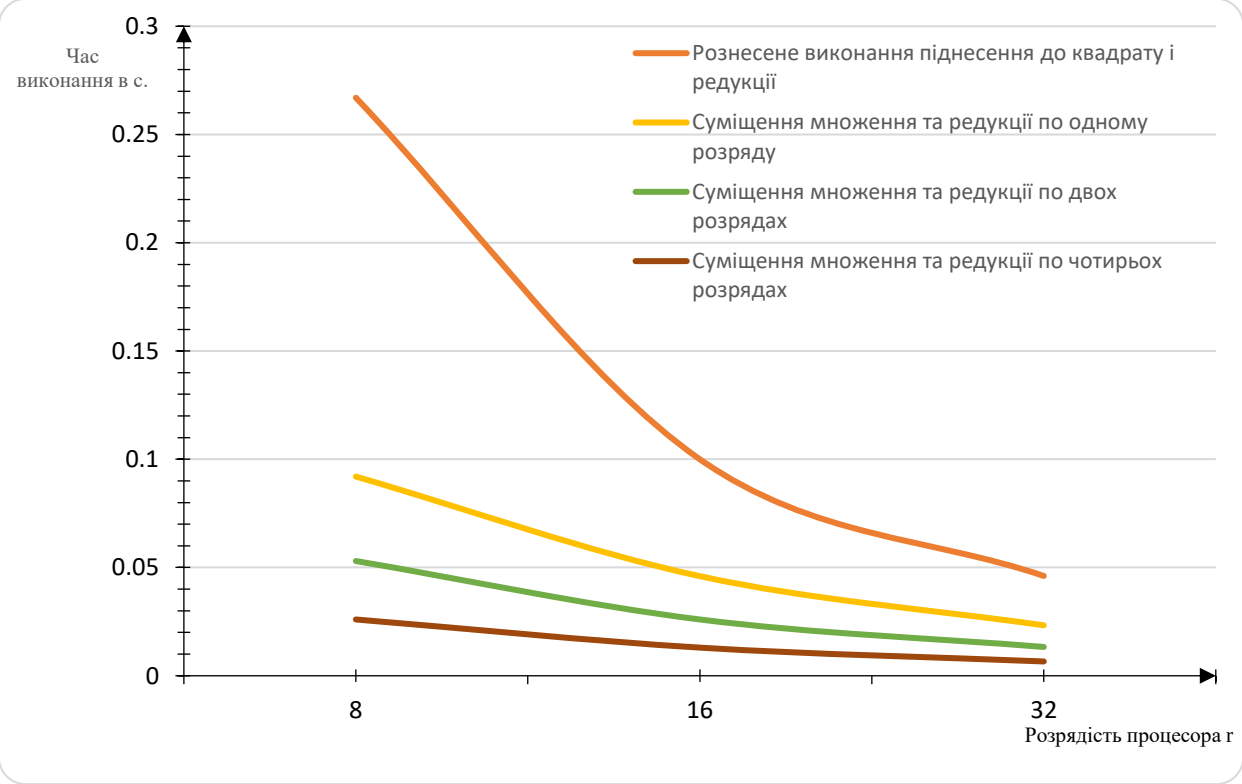


Рис 3.3 Залежність часу виконання модулярного піднесення до квадрату від розрядності мікроконтролера для 2048-розрядних чисел.

На графіку на рис.3.3 представлено чотири залежності часу виконання програми від розрядності процесора для базового варіанту і трьох можливих реалізацій запропонованого методу. З графіків видно, що для всіх розрядностей процесора використання запропонованого методу модулярного піднесення до квадрату дозволяє підвищити швидкодію виконання цієї важливої для криптографічних застосувань операції.

При порівнянні графіків на рис.3.1 і рис.3.3 можна зробити висновок про

те, що при збільшенні розрядності чисел, над якими здійснюється операція модулярного експоненціювання вдвоє, час виконання обчислень модулярного квадрату зростає в 3.57 раз. Це свідчить про те, що запропонований методо модулярного піднесення до квадрату дозволяє в 1.12 раз зменшити залежність часу виконання операції від розрядності чисел, над котрими виконується операція.

Для порівняльної оцінки ефективності запропонований методо модулярного піднесення до квадрату побудовані графіки залежності коефіцієнту прискорення модулярного піднесення до квадрату в порівнянні з рознесеним множенням та редукцією від розрядності мікроконтролера для чисел довжиною 2048. Ці графіки представлені на рис.3.4.

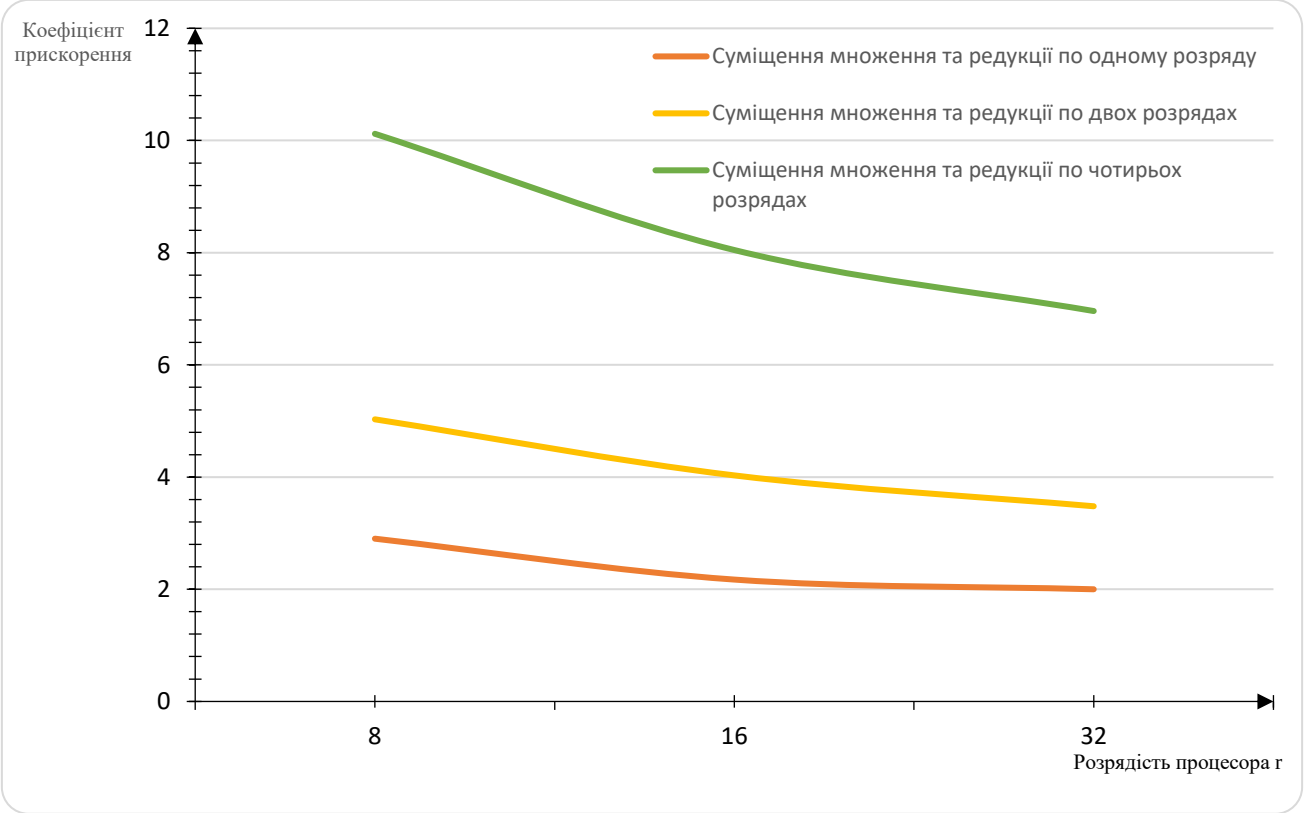


Рис 3.4 Залежність коефіцієнту прискорення модулярного піднесення до квадрату в порівнянні з рознесеним множенням та редукцією від розрядності мікроконтролера для чисел довжиною 2048.

Аналіз представлених на рис.3.4. графіків свідчить про те, що коефіцієнт

прискорення обчислення модулярного квадрату по запропонованому методу в порівняні з базовим варіантом практично не залежить від розрядності чисел, над якими виконується операція модулярного експоненціювання, частиною якої є модулярне піднесення до квадрату.

Для більш повною оцінки впливу на ефективність запропонованого методу розрядності чисел, які застосовуються в механічних криптографічних захисту, досліджені залежності часу виконання модулярного піднесення до квадрату від розрядності мікроконтролера для 4096-розрядних чисел, які графічно представлені на рис. 3.5.

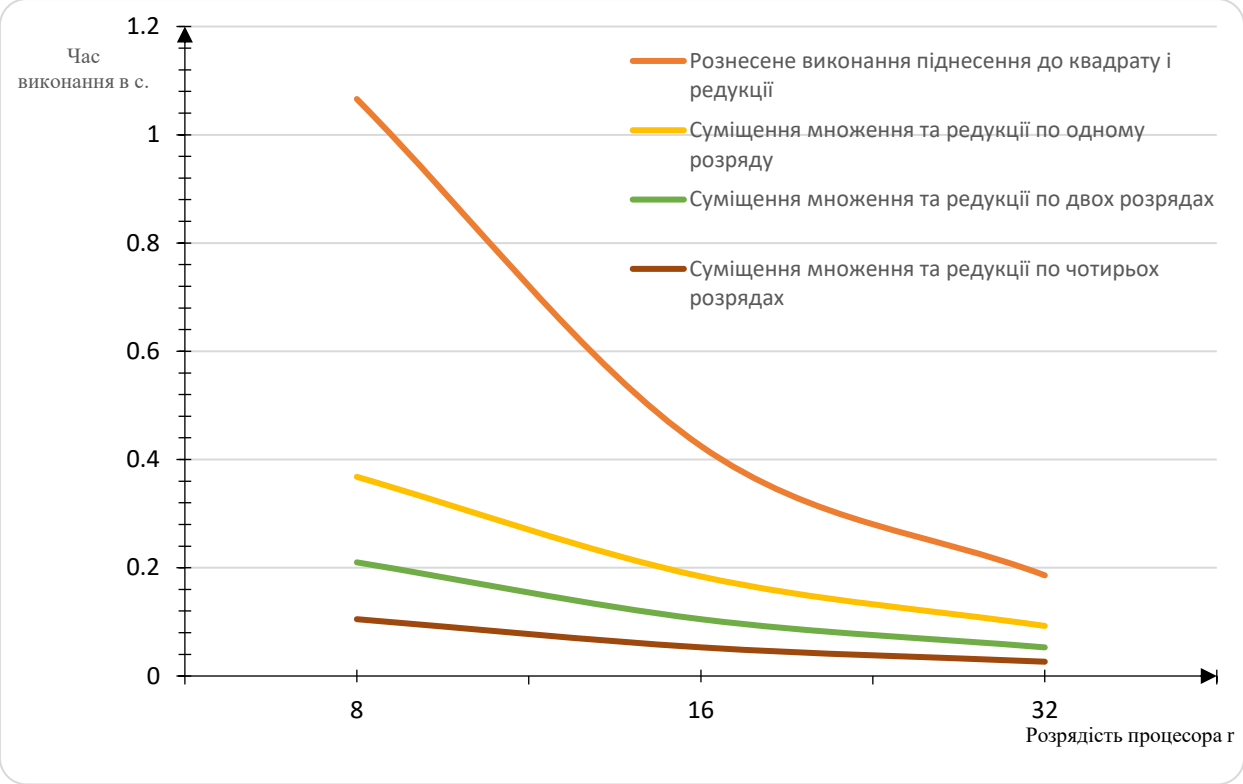


Рис 3.5 Залежність часу виконання модулярного піднесення до квадрату від розрядності мікроконтролера для 4096-розрядних чисел.

Порівняльний аналіз графіків часу виконання операції модулярного піднесення до квадрату для різних довжин чисел, що використовуються в криптографічних протоколах свідчить про те, що запропонований метод виконання цієї операції дозволяє змінити характер залежності часу обчислень

від розрядності. Цей ефект досягається за рахунок того, що в запропонованому методі довжина реальних оператор не постійна, а зменшується від циклу до циклу. В принципі цей ефекти має важливе значення в плану того, що запропонований метод дозволяє мінш чутливо для продуктивності збільшувати розрядність чисел задля підвищення рівня захищеності криптографічних механізмів.

На рис.3.6 представлено залежність коефіцієнту прискорення модулярного піднесення до квадрату в порівнянні з рознесеним множенням та редукцією від розрядності мікроконтролера для чисел довжиною 4096.

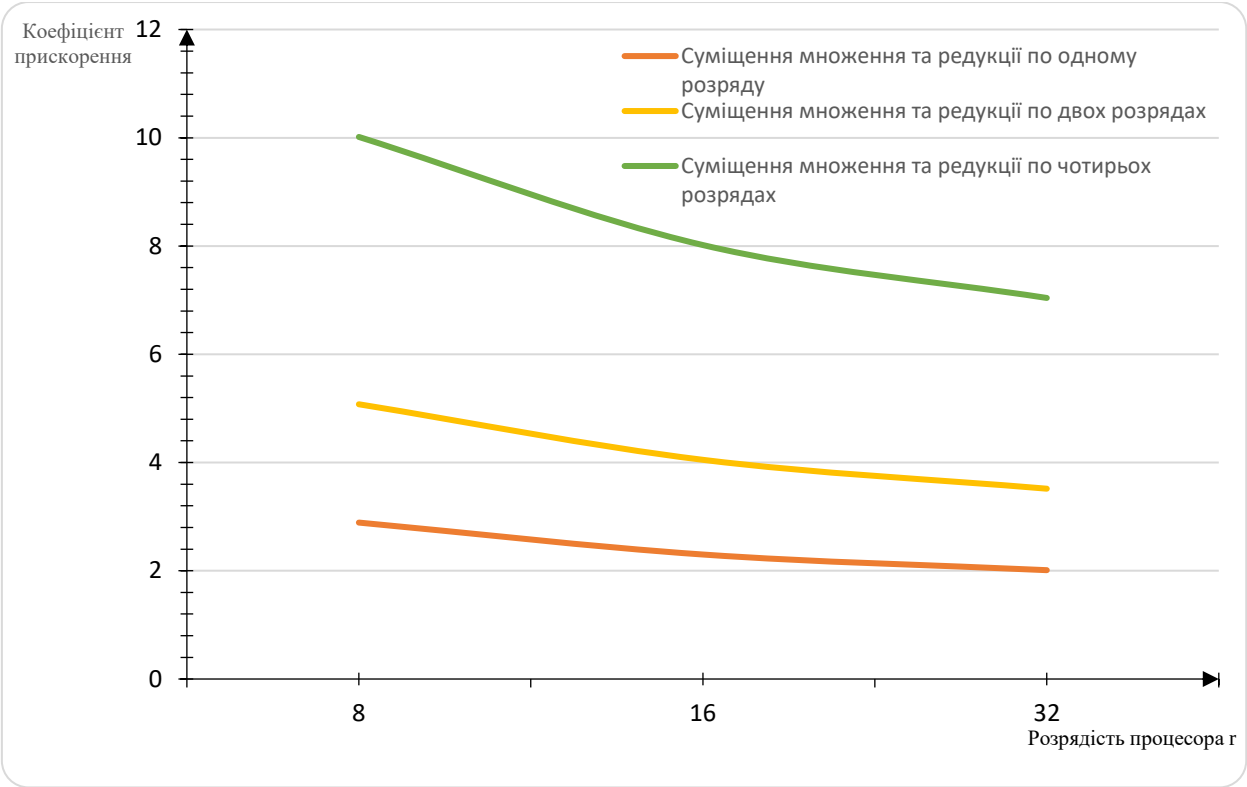


Рис 3.6 Залежність коефіцієнту прискорення модулярного піднесення до квадрату в порівнянні з рознесеним множенням та редукцією від розрядності мікроконтролера для чисел довжиною 4096.

Аналіз представлених на рис.3.6. графіків свідчить про те, що коефіцієнт прискорення обчислення модулярного квадрату по запропонованому методу в порівняні з базовим варіантом практично не залежить від розрядності чисел,

над якими виконується операція модулярного експоненціювання, частиною якої є модулярне піднесення до квадрату.

Висновки до розділу 3

В результаті виконання розробок та досліджень, які складають третій розділ проєкту можна зробити наступні висновки:

1. Визначено архітектуру та виконана розробка програмних засобів для тестування працездатності в різних режимах запропонованого в попередньому розділі методу прискореного модулярного піднесення до квадрату, яке здійснюється на числами, довжина яких на порядок перевищує розрядність процесора. Розробка програмних засобів виконана на мові програмування C++, що дозволило найбільш повно реалізувати потенціал методу в плані прискорення виконання модулярного піднесення до квадрату. Розроблена програма дозволяє гнучко змінювати довжину слів та емалювати різну розрядність процесора, що потрібно для отримання залежностей характеристик ефективності методу від параметрів його реалізації.

2. Для проведення порівняльного аналізу часових характеристик двох варіантів реалізації прискорення за рахунок виключення притаманної піднесенню до квадрату операційної надмірності програму, яка здійснює секційне піднесення до квадрату з використанням процесорної операції множення та подальшою модулярною редукцією отриманого квадрату за методом Монтгомері.

3. З використанням розробленої програми виконано перевірку працездатності запропонованого методу прискорення модулярного піднесення до квадрату за рахунок зменшення довжини операндів і, відповідно, кількості процесорних операцій. Показано, що змінний формат чисел, застосований в розробці являє собою вирішальний чинник прискорення піднесення до квадрату: при додаванні багаторозряодних чисел до суми часткових добутків, їх довжина постійно скорочується. Відповідно, потрібно здійснювати менше процесорних операцій додавання зсунутого множника суми часткових

добутків.

4. З використанням розробленої програми досліджено залежність часу виконання операції модулярного піднесення до квадрату від розрядності мікроконтролеру. Аналіз отриманих залежності показує, що прискорення виконання модулярного піднесення до квадрату досягає найбільших значень при малих значеннях розрядностей мікроконтролера. Зі збільшення розрядності процесора, ефективність запропонованого методу з точки зору прискорення обчислень зменшується.

5. Порівняльний аналіз графіків часу виконання операції модулярного піднесення до квадрату для різних довжин чисел, що використовуються в криптографічних протоколах свідчить про те, що запропонований метод виконання цієї операції дозволяє змінити характер залежності часу обчислень від розрядності. Цей ефект досягається за рахунок того, що в запропонованому методі довжина реальних оператор не постійна, а зменшується від циклу до циклу.

ВИСНОВКИ

Результати бакалаврського проекту, спрямованого на теоретичне обґрунтування, розробку та експериментальне дослідження методу прискореного модулярного піднесення до квадрату, можуть бути сформульовані у вигляді наступних висновків:

1. Операція модулярного піднесення до квадрату є критичною з позиції програмної реалізації в реальному часі на термінальних мікроконтролерах Інтернету речей. Велика розрядність чисел, що використовується в сучасних протоколах, разом з обмеженою потужністю та компактністю термінальних процесорів мікроконтролерів, становить серйозну перешкоду для реалізації криптографічних алгоритмів в реальному часі.
2. Проведений аналіз показав, що найбільш перспективним шляхом прискорення програмної реалізації базової операції криптографічних алгоритмів з відкритим ключем - модулярного піднесення до квадрату є пошук шляхів прискорення виконання саме операції піднесення до квадрату, яка становить дві третини обчислень модулярного піднесення до степеня.
3. В результаті детального вивчення операції модулярного піднесення до квадрату було виявлено, що швидкодію можна підвищити приблизно вдвоє шляхом виключення зайвих операцій. Іншим резервом прискорення операції модулярного піднесення до квадрату є поєднання в часі операцій піднесення до квадрату та модулярної редукції.
4. Порівняльний аналіз редукції Баррета та Монтгомері показав, що застосування останньої забезпечує суттєво більші можливості для прискорення виконання операції модулярного піднесення до квадрату.
5. Детальний аналіз показав, що важливим чинником прискорення операції модулярного піднесення до квадрату є організація одночасного множення

та редукції отриманого часткового результату з обробкою декількох розрядів множника та використання таблиць передобчислень, які залежать лише від значення модуля.

6. Показано, що основними резервами підвищення швидкодії комп'ютерної реалізації базової процедури широкого кола алгоритмів криптографічного захисту інформації є виключення операційної надмірності виконання двох базових операцій – модулярного піднесення до квадрату та модулярного множення на ціле число.

7. Теоретично доведено, що за рахунок виключення операційної надмірності при обчислення модулярного квадрату можна гранично прискорити обчислювальну реалізацію цієї операції в два рази. Доведено, що ефективне використання цього резерву прискорення обчислення модулярної експоненти може бути реалізоване лише за у мови використання спеціального методу прискореного модулярного множення на постійне число.

8. Теоретично обґрунтовано, запропоновано та досліджено метод прискореної обчислювальної реалізації операції модулярного піднесення до квадрату, який відрізняється тим, що довжина множника зменшується на кожному циклі, за рахунок чого досягнуто зменшення кількості операцій процесорного множення і, тим самим забезпечено прискорення виконання цієї операції.
9. Запропонований метод пристосовано для ефективної реалізації модулярної редукції з використанням технології Монтгомері з використанням передобчислень, які залежать тільки від модуля і, відповідно, для криптографічних застосувань можуть проводитися заздалегідь і тільки один раз, оскільки модуль є частиною відкритого ключа.

10. Розроблено і досліджено алгоритм прискореного модулярного піднесення до квадрату, який базується на запропонованому методі та використанню модулярної редукції Монтгомері і відрізняється тим, що за рахунок одночасної обробки декілька розрядів множника, дозволяє досягти подальшого прискорення обчислювальної реалізації операції модулярного піднесення до квадрату. Показано, що обмеження на кількість розрядів, обробка яких суміщується у часі визначаються граничним об’ємом пам'яті для таблиць передобчислень.

11. Визначено архітектуру та виконана розробка програмних засобів для тестування працездатності в різних режимах запропонованого в попередньому розділі методу прискореного модулярного піднесення до квадрату, яке здійснюється на числами, довжина яких на порядок перевищує розрядність процесора. Розробка програмних засобів виконана на мові програмування С++, що дозволило найбільш повно реалізувати потенціал методу в плані прискорення виконання модулярного піднесення до квадрату. Розроблена програма дозволяє гнучко змінювати довжину слів та емалювати різну розрядність процесора, що потрібно для отримання залежностей характеристик ефективності методу від параметрів його реалізації.

12. Для проведення порівняльного аналізу часових характеристик двох варіантів реалізації прискорення за рахунок виключення притаманної піднесенню до квадрату операційної надмірності програму, яка здійснює секційне піднесення до квадрату з використанням процесорної операції множення та подальшою модулярною редукцією отриманого квадрату за методом Монтгомері.

13. З використанням розробленої програми виконано перевірку

працездатності запропонованого методу прискорення модулярного піднесення до квадрату за рахунок зменшення довжини операндів і, відповідно, кількості процесорних операцій. Показано, що змінний формат чисел, застосований в розробці являє собою вирішальний чинник прискорення піднесення до квадрату: при додаванні багаторозрозних чисел до суми часткових добутків, їх довжина постійно скорочується. Відповідно, потрібно здійснювати менше процесорних операцій додавання зсунутого множника суми часткових добутків.

14. З використанням розробленої програми досліджено залежність часу виконання операції модулярного піднесення до квадрату від розрядності мікроконтролеру. Аналіз отриманих залежності показує, що прискорення виконання модулярного піднесення до квадрату досягає найбільших значень при малих значеннях розрядностей мікроконтролера. Зі збільшення розрядності процесора, ефективність запропонованого методу з точки зору прискорення обчислень зменшується.

15. Порівняльний аналіз графіків часу виконання операції модулярного піднесення до квадрату для різних довжин чисел, що використовуються в криптографічних протоколах свідчить про те, що запропонований метод виконання цієї операції дозволяє змінити характер залежності часу обчислень від розрядності. Цей ефект досягається за рахунок того, що в запропонованому методі довжина реальних оператор не постійна, а зменшується від циклу до циклу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ostrovska. B Method of accelerated modular multiplication with Montgomeri group reduction / I. Boyarshin, O. Markovskyi, B. Ostrovska // Proceeding of International Conference Security, Fault Tolerance, Intelligence. ICSFTI-2022,- Kyiv.- P.40-45.
2. Ostrovska. B Organization of parallel execution of modular multiplication to speed up the computational implementation of public-key cryptography / I. Boyarshin, O. Markovskyi, B. Ostrovska //Information, Computing and Intelligent systems. – 2022. – № 3. – P. 77.
3. Кабір Л.А. Метод прискорення модулярного множення за технологією Монтгомері / Л.А. Кабір, О.В. Русанова, І.О. Гуменюк //Альманах науки № 1(52).- 2022.- С.44-46.
4. Трибунська К.Є. Метод прискорення модулярного множення з використанням групової редукції /К.Є. Трибунська // Актуальні питання розвитку науки та освіти: матеріали М Міжнародної науково-практичної конференції м.Львів, 30-31 березня 2022 року.-Львів: Львівський науковий форум, 2022.- С.38- 46.
5. Thangaval M. Improved secure rsacryptosystem (ISRSAC) for data confidentiality in cloud / M. Thangaval, P.Varalakshmi // International Xournal of Information Systems and Change Managerment,- 2017.- Vol.9,- No.4, - P.46-53.
6. Osadchyy V. The Order of Edwards and Montgomery Curves «, / V.Osadchyy // WSEAS Transactions on Mathematics, - 2020.- Vol. 19.- № 25, - P. 253-264.
7. Haches G. Montgomery multiplication with no final subtraction./ G. Haches, X.X. Quisquater // Cryptographic Hardware and Embedded System-CHES'2000. LNCS-1965, Springer-Verlag. — 2000.- P. 293-301.

8. Elfard S. Xustification of Montgomery Modular Reductions / S. Elfard //Advanced Computing.- 2012. - № 11. – P.41-45.
9. Che Wun Chion. Parallel modular multiplication with table look-up. / Che Wun Chion, Ted C.Yang // International Xournal of Computer Mathematics.-1998.- Vol.69.-Issue 1-2.- P.22-23.
10. Zuras D., More on squaring and multiplying larges integers, IEEE Transactions on Computers,-1994.- Vol. 43, - № 8,- P. 899–908.
11. Giorgi P. Parallel modular multiplication on multi-core processors. / Giorgi P., Imbert L., Izard T. // IEEE Symposium on Computer Arithmetic, Apr 2013, Austin, TX, United States. - P.135-142.
12. Buhrow B. Parallel modular multiplication using 512-bit advanced vector instructions: RSA fault-inxection countermeasure via interleaved parallel multiplication / Buhrow B., Gilbert B., Haider C. // Xournal of Cryptographic Engineering.-2021.- № 2.- P.46-53.
13. Hong S-M. New Modular Multiplication algorithms for fast modular exponeniation / Hong S-M., Oh S-Y., Yoon H // Advances in Cryptology-Proceedings of Eurocrypt ’96. –Springer-Verlag. -1996. – P.166-177.
14. Laszlo Hars. Long Modular multiplication for Cryptographic Applications.//Cryptographic Hardware and Embedded System- CHES’2004. LNCS-3156, Springer-Verlag, - 2004. - P.45-61.
15. Giorgi P. Parallel modular multiplication on multi-core processors. / Giorgi P., Imbert L., Izard T. // IEEE Symposium on Computer Arithmetic, Apr 2013, Austin, TX, United States. - P.135-142.
16. Buhrow B. Parallel modular multiplication using 512-bit advanced vector instructions: RSA fault-inxection countermeasure via interleaved parallel multiplication / Buhrow B., Gilbert B., Haider C. // Xournal of Cryptographic Engineering.-2021.- № 2.- P.46-53.
17. Марковський О.П. Метод прискорення експоненціювання з використанням передобчислень / О.П. Марковський, О.В. Русанова, А.А. Олієвський,

В.М.Черевик // Телекомунікаційні та інформаційні технології. - 2018.- № 1(58). – С.31-39.

18. Elfard S. Xustification of Montgomery Modular Reductions / S. Elfard //Advanced Computing.- 2012. - № 11. – P.41-45.
19. Анисимов А.В. Швидке пряме обчислення модулярної редукції // Кібернетика та системний аналіз.-1999.-№ 4.-С.3-12.
20. Kawamura S. A fast modular exponentiation algorithm / S. Kawamura, K. Takabayashi, A. Shimbo // IEEE Transaction on Information Theory. – Vol. 94. – № 6. – 2015. – P.2136-2142.
21. Самофалов К.Г. Ефективна реалізація мультиплікативних операцій модулярної арифметики в системах захисту інформації / К.Г. Самофалов, Г.М.Луцкий, А.П. Марковский // Proceeding of International scientific conference UNITECH-09. Gabrovo 20-21 November 2009.- Technical University of Gabrovo. - 2009.— V.1.— P.435-437.
22. Анісімов А.В. Алгоритмічна теорія великих чисел / А.В. Анісімов // К.: Академперіодика. —2001. — С.153.
23. Blum T., Paar C. Montgomary Modular Exponentiontiation on Reconfigurable Hardware.//Proc.14-th IEEE Symp.on Comput. Arithmetic, Adelaide,14-16 April 1999,-IEEE Press,-1999- P.70-77.

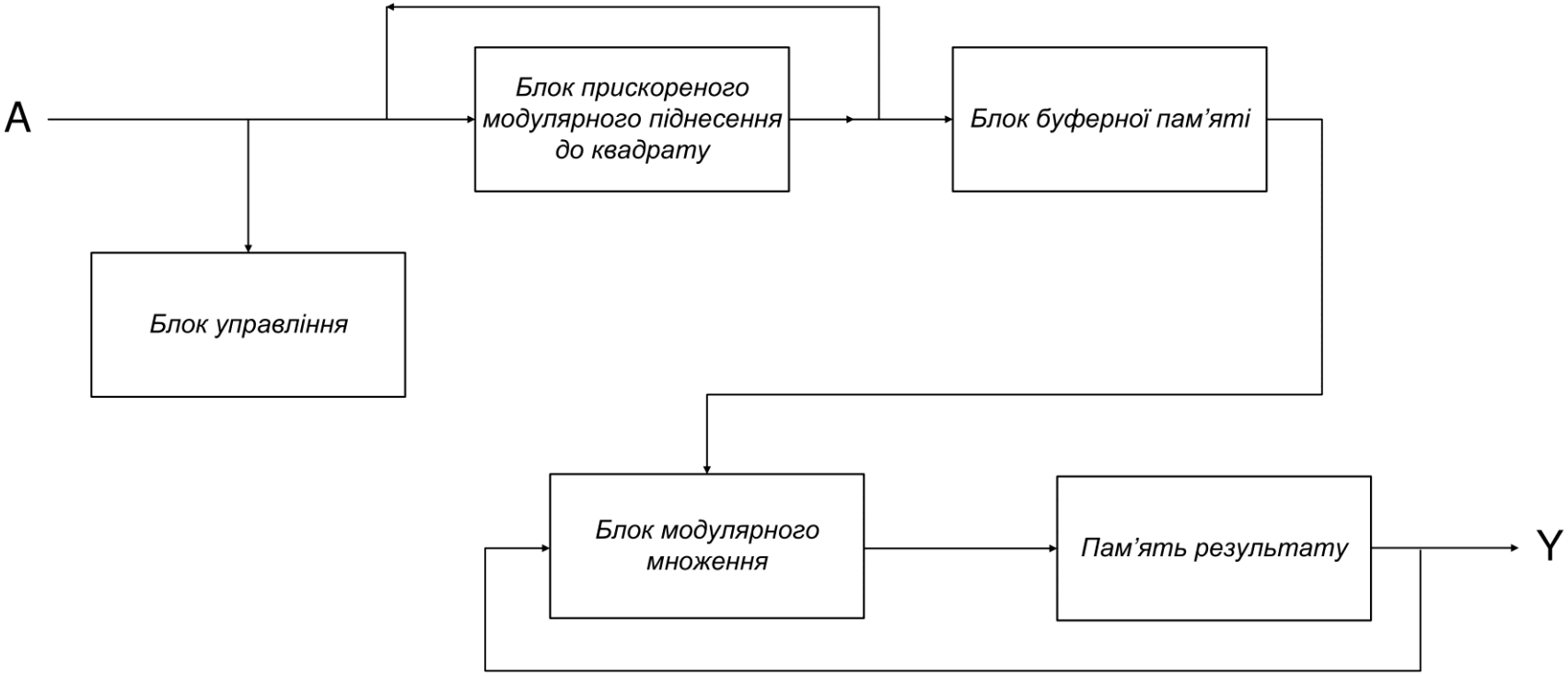
ДОДАТОК 1

Метод прискореного модулярного піднесення до квадрату

Електрична схема двопотокового обчислювача модулярної експоненти (структурна схема)

Аркушів 1

Київ – 2023 р.



Зм	Лист	Чедонум.	Підп.	Дата	**ІАЛЦ.468243.004 Е1**
Розроб.		Островська Б. Й.			**Двопотоковий обчислювач модулярної експоненти Схема електрична структурна**
перевірив		Марковский О. П.			
Н.контр.		Виноградов			**КПІ ім. Ігоря Сікорського», ФІОТ, ІВ-93**
Затв.					

Літ.	Маса	Масштаб
Аркуш	Аркушів 1	

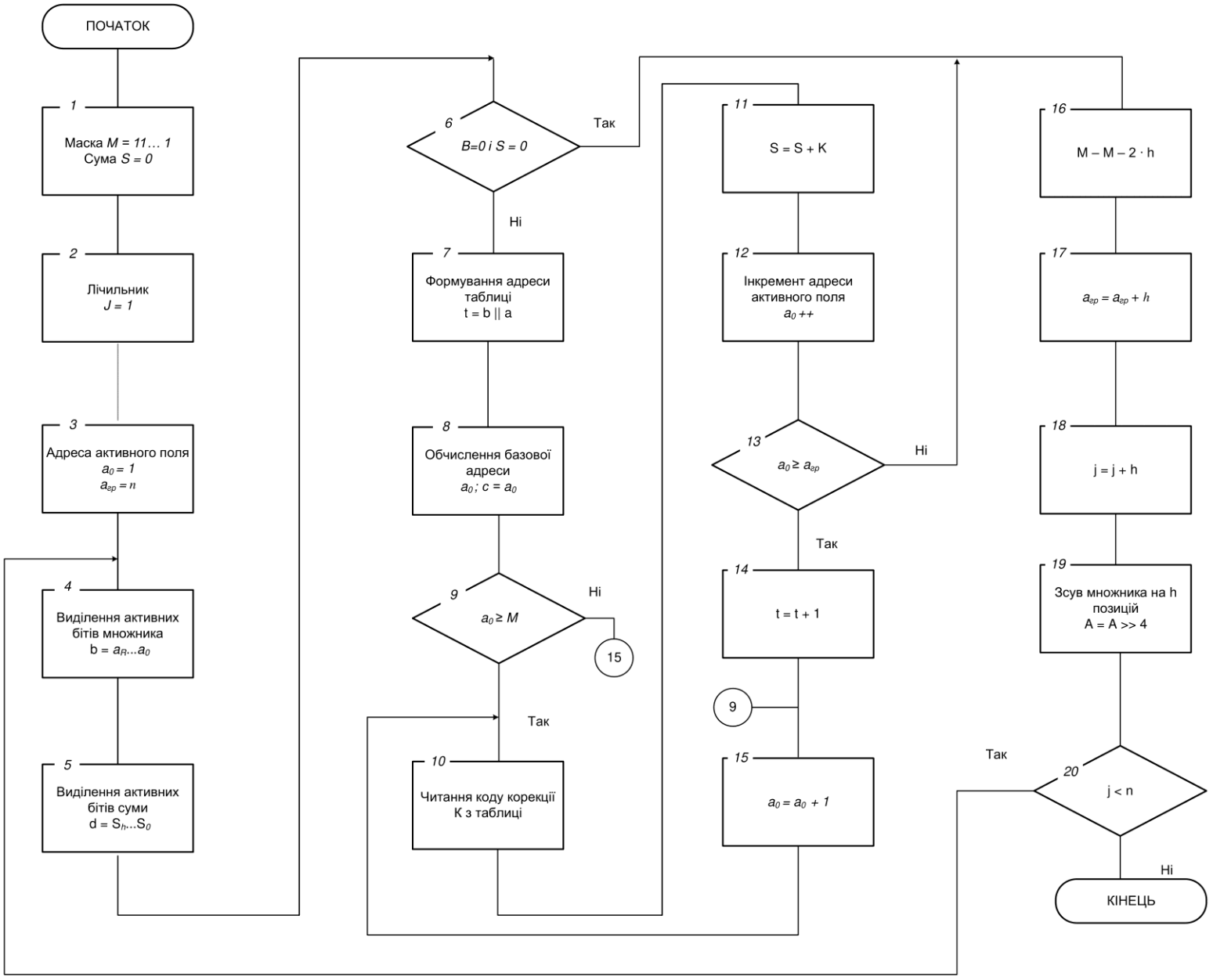
ДОДАТОК 2

Метод прискореного модулярного піднесення до квадрату

Блок-схема алгоритму прискореного модулярного піднесення до квадрату (принципова схема)

Аркушів 1

Київ – 2023



Зм.	Лист	№докум.	Підп.	Дата	**ІАЛЦ.468243.005 Д1**		
Розроб.		Островська Б. В.			*Блок-схема алгоритму прискореного модулярного піднесення до квадрату (принципова схема)*	Літ.	Маса	Масштаб
Перевір.		Марковский О. П.						
Н.контр.		Виноградов				Аркуш	Аркушів 1	
Затв.						**КПІ ім. Ігоря Сікорського», ФІОТ, ІВ-93**		

ДОДАТОК З

Метод прискореного модулярного піднесення до квадрату

Блок модулярного піднесення до квадрату(функціональна схема)

Аркушів 1

Київ – 2023 р.

Зм.	Лист	№докум.	Подп.	Дата			
Разроб.		Островська Б.В					
Перевірив		Марковский О. П.					
Н.контр.		Виноградов Ю.М					
Затв.							

ІАЛЦ.468243.006 Е2

Блоку модулярного піднесення до квадрату (функціональна схема)

Лит.	Маса	Масштаб
Аркуш	Аркушів 1	

КПІ ім. Ігоря Сікорського», ФІОТ, ІВ-93

ДОДАТОК 4

Метод прискореного модулярного піднесення до квадрату

Текст програмного коду

Аркушів 8

Київ – 2023 р.

```
1.  #include <iostream>
2.  #include <math.h>
3.  #include <stdlib.h>
4.  #include <stdio.h>
5.
6.  unsigned long MON_REC(unsigned long,int);
7.  int ORTA(int *, int );
8.    int POH(int*,int,int);
9.   void DIFF(int*,int*);
10. long CENA(int*);
11.
12.int main()
13.{
14.         long i,j,k,n,m,s,f,r,s2,s3,s4,d;
15.         float a,t1,t2,t3,t4, ma,d1,d2,d3,d4;
16.    n=0;
17.         f=64 ; m=413;
18.         std::cout<<"\n  153-9 ="<<MON_REC(153,9);
19.         for(j=2;j<m;j++)
20.         {
21.           if((f*j)%m==1) std::cout<<"\n Inverse "<<f<<"  ="<<j;
22.
23.
24.           n=5+j;
25.           s=n*7;
26.           if((s&1)==1) s+=413; s>>=1;
27.           if((s&1)==1) s+=413; s>>=1;
28.           if((s&1)==1) s+=413; s>>=1;
29.           if(s==259) std::cout<<"\n  J="<<j;
30.         }
31.         r=0;
32.          s=285; d=13; m=413;
33.          for(j=0;j<4;j++     )
34.          {
35.            if((d&1)==1) r+=s;
36.            if((r&1)==1) r+=m;
37.            r>>=1;
38.            d>>=1;
39.          }
40.          if(r>=413) r-=413;
41.         std::cout<<"\n R = "<<r;
42.          //
43.          r=317;
```

```
          for(j=0;j<6;j++)
          {
            if((r&1)==1) r+=m;
            r>>=1;
                 std::cout<<"\n    "<<j<<"  R_j ="<<r;
          }


        for(i=0;i<413;i++)
        {
          r=i;
          s=26; d=47; m=413;
          for(j=0;j<6;j++    )
          {
            if((d&1)==1) r+=s;
            if((r&1)==1) r+=m;
            r>>=1;
            d>>=1;
          }
          if(r==4) std::cout<<"\n Y= "<<i;
   }

        if((j*64)%413 ==1) std::cout<<"\n 8 ="<<j;

   d=0; s=0; s2=0;s3=0;s4=0; k=0;
        for(i=0;i<32;i++)
   for(j=i;j<32;j++)
   {
     d++;
          n=2048-i*64-j*64;
          if((i==30)&&(j==30)) std::cout<<"\n 30 30 n="<<n;
          if(i!=j) n--;
          if(n>0) { s+=n; s2++; }
          else if(n<0) { s3 -=n; s4++; }   else k++;
        }
   std::cout<<"\n Multiplications ="<<d;
   std::cout<<"\n with reduction "<<s2<<" Avarege "<<(s+0.0)/s2;
   std::cout<<"\n with antireduction "<<s4<<" Avarege "<<(0.0+s3)/s4;
   std::cout<<"\n without reduction "<<k;

        d=(416+3+226+362+163+60)%m;
        d=258;
        std::cout<<"\n  D="<<d;
```

```
87.         r=8;
88.    for(j=3;j<18;j++)
89.    {
90.      if(((r*d)%m)==62) std::cout<<"\n  OK j="<<j;
91.      r*=2;
92.         }
93.             //  m=16  d=1
94.    float A[13]={0,2,3.87,5.62,7.21,8.69,9.99,11.21,12.28,13.2,14,14.67,15.2};
95.    s=d;
96.    for(j=0;j<8;j++)
97.    {
98.       if(d<13) a=A[d]; else a=m;
99.                  dd=(m-a)*(1-P);
100.                       d=dd; if(dd-d>0.5) d++;
101.                       s+=dd;
102.                       std::cout<<"\n j="<<j<<"  "<<d;
103.           }
104.           std::cout<<"\n s="<<s;
105.
106.           s=0;
107.           for(l=0;l<10000;l++)
108.                {
109.                  d=z; b=0;
110.                  for(u=0;u<8;u++)
111.                  {
112.                        k=0; i=0;
113.                        do
114.                     {
115.                    j=rand()%m;
116.                        q=1<<j;
117.                        if((k&q)==0) { k|=q; i++; }
118.                     } while (i<d);
119.                 i=1<<(m-1);
120.                      if((k&i)!=0) r=((k-i)<<1)+1; else r=k<<1;
121.                      r|=k; j=0; while(r>0) { if((r&1)==1) j++; r>>=1; }
122.                      dd=(m-j)*(1-P);
123.
124.
125.                      d=dd; if((dd-d)>0.5) d++;
126.                           b+=dd;
127.              }
128.              s+=b;
129.           }
```

```
130.          if(f) cout<<"\n  All triples are ortogonal ";
131.          else cout<<"\n Triple NO";
132.           // Testing for 4;
133.           f=1;
134.           for(v=0;v<(n-3);v++)
135.          {
136.            for(i=0;i<m;i++) T[i][0]=A[i][v];
137.            for(l=v+1;l<(n-2); l++)
138.            {
139.              for(i=0;i<m;i++) T[i][1]=A[i][l];
140.              for(t=l+1; t<(n-1); t++)
141.              {
142.               for(i=0;i<m;i++) T[i][2]=A[i][t];
143.                for(q=t+1; q<n; q++)
144.                {
145.                     for(i=0;i<m;i++) T[i][3]=A[i][q];
146.                     r=0;
147.                      // Select 4 rows
148.                for(r1=0;r1<(n-3);r1++)
149.                   {
150.                     for(i=0;i<4;i++) G[0][i]=T[r1][i];
151.                     for(r2=r1+1;r2<(n-2);r2++)
152.                     {
153.                      for(i=0;i<4;i++) G[1][i]=T[r2][i];
154.                      for(r3=r2+1;r3<(n-1); r3++)
155.                      {
156.                        for(i=0;i<4;i++) G[2][i]=T[r3][i];
157.                        for(r4=r3+1;r4<n; r4++)
158.                        {
159.                              for(i=0;i<4;i++) G[3][i]=T[r4][i];
160.                              if(ORTA(&G[0][0],4)) r=1;
161.                        }
162.                     }
163.                   }
164.                }
165.                if(r==0) f=0;
166.              }
167.            }
168.          }
169.         }
170.         if(f) cout<<"\n  All fours are ortogonal "; else cout<<"\n Four NO";
171.               s/=10000;
172.               std::cout<<"\n d="<<z<<"  s="<<s;
```

```
173.          std::cout<<"\n  ";
174.          m=8 ;
175.          b=m*m; a=m*b;
176.          s=(6*b-43*m+56)/(b-3*m+2);
177.          std::cout<<"\n C= "<<s;
178.          //
179.          z=0; q=0; u=0; j=0;
180.          int V[16]={0};
181.               for(i=7;i<65536;i++)
182.          {
183.            k=i; d=0; for(l=0;l<16;l++) { V[l]=k&1; d+=k&1; k>>=1; }
184.            if(d==3)
185.                 {
186.                   g=0;
187.                        for(l=0;l<16;l++)
    if((V[l]==1)||((V[l]==0)&&(V[(l+1)%16]==1))) g++;
188.                        if(g==6) { q++; std::cout<<"\n  5 q="<<q<<"    ";
    for(l=0;l<16;l++) std::cout<<" "<<V[l]; }
189.                        z++;
190.                 }
191.               }
192.                   s=(16*4+192*5+352*6+0.0)/z;
193.                        std::cout<<"\n Total "<<z<<"  s= "<<s;"";
194.                   int A[7];
195.              int B[7][7];
196.              int R[13]={0};
197.              k=43; q=k;
198.              for(i=1;i<7;i++) { A[i]=q&1; q>>=1; }
199.              for(i=1;i<7;i++)
200.              for(j=1;j<7;j++) B[i][j]= A[i]&A[j];
201.              R[1]=1;
202.              R[2]= A[2]^A[2]; p= A[2];
203.              s= A[3]+A[3]+A[2]+p;
204.              R[3]= s&1; p = s>>1;
205.              s= A[4]+A[4]+B[2][3]+B[2][3]+p;
206.              R[4]= s&1; p= s>>1;
207.              s= A[5]+A[5]+ A[3]+ 2*B[2][4]+ p;
208.              R[5]= s&1; p = s>>1;
209.              s= 2*A[6]+ 2*B[2][5] + 2*B[3][4] + p;
210.              R[6]= s&1; p=s>>1;
211.              s= 2*B[2][6]+ +2*B[3][5]+A[4]+p;
212.              R[7]= s&1; p=s>>1;
213.              s= 2*B[3][6] + 2*B[4][5] + p;
```

```
214.            R[8]= s&1; p=s>>1;
215.            s= A[5]+ 2*B[4][6]+ p;
216.            R[9]= s&1; p=s>>1;
217.            s= 2*B[5][6] + p;
218.            R[10]= s&1; p=s>>1;
219.            s = A[6]+p;
220.            R[11] = s&1; p= s>>1;
221.            R[12] = p ;
222.            q=0;
223.            for(j=12; j>0; j--) { q<<=1; q+=R[j]; }
224.            p=k*k;
225.                 t1=0;t2=0;t3=0; c1=0; c2=10;
226.              for(j=0;j<8;j++) { t1+=X[0][j]; t2+=X[1][j]; t3+=Y[j];
    if(Y[j]>c1) c1=Y[j]; if(Y[j]<c2) c2=Y[j];}
227.                 t1/=8; t2/=8; t3/=8;
228.                 d1=(1.9-0.2)/2.8;  d2=(2.8-0.1)/2.8; d3=(c1-c2)/2.8;
229.                 d1=d1*d1; d2=d2*d2; d3=d3*d3; t4=d1*d2;
230.                 c1=0; c2=0;
231.                 for(j=0;j<8;j++) { c1+=(X[0][j]-t1)*(Y[j]-t3); c2+=(X[1][j]-
    t2)*(Y[j]-t3);}
232.                 c1/=8; c2/=8;
233.                 a=c1*d2/t4; b=c2*d1/t4; ma= t3-a*t1-b*t2;
234.            std::cout<<"\n k="<<k<<"  Real ="<<p<<"  Calculeted = "<<q;
235.            std::cout<<"\n   ";
236.            for(j=12;j>0;j--) std::cout<<" "<<R[j];
237.
238.              return 0;
239.       }
240.
241.       unsigned long MON_REC(unsigned long k, int m)
242.        {
243.          unsigned long M=413; int j;
244.               for(j=1;j<=m;j++)
245.               {
246.                        if((k&1)==1) k+=M;
247.                        k>>=1;
248.               }
249.               return k;
250.        }
251.       int ORTA(int *h, int n)
252.       {
253.         int i,j,k,f,z,cj,d;
254.         int A[100]={0};
```

```
255.        for(i=0;i<n;i++)
256.        {
257.          A[i]=0; k=1;
258.          for(j=0;j<n;j++) { A[i]+=*(h+n*i +(n-1-j))*k; k*=2; }
259.        }
260.        for(i=0;i<n;i++) if(A[i]==0) return 0;
261.        z=1;
262.        for(j=0;j<n;j++)
263.        {
264.         d=0;
265.         for(i=0;i<n;i++) d+= A[i]&z;
266.         if(d==0) return 0;
267.         z<<=1;
268.        }
269.        f=1;
270.        for(j=1;j<k;j++)
271.        {
272.          z=0; cj=j;
273.          for(i=0;i<n;i++)
274.          {
275.            if(cj&1) z^=A[i];
276.            cj>>=1;
277.          }
278.          if(z==0) return 0;
279.        }
280.        return 1;
281.      }
282.
283.        long CENA(int* v) //
284.        {
285.                long r=0; int i,j;
286.                for(i=0;i<n;i++)
287.                for(j=0;j<n;j++)
288.                r+=(*(v+n*i+j)-0.5*N)*(*(v+n*i+j)-0.5*N);
289.                return r;
290.        }
291.
292.        void DIFF(int* v,int* h) //
293.        {
294.                int i,j;
295.          for(i=0;i<n;i++)
296.          for(j=0;j<n;j++)
297.          *(v+n*i+j)=POH(h,i+1,j+1);
```

```
 }


int POH(int *h,int k, int m)

{
   int i,j,q,d;
        long a=0;
        m=1<<(m-1); k=1<<(k-1); //
        for(i=0;i<16;i++) //
        {
          q=*(h+i); //
          q&=k;
          j=i^m;
          d=*(h+j);
          d&=k;
          if(d!=q) a++;
        }
        return a;
}

int NL(int *h,int k)

{
  int i,j,t,q,d,s;
  q=N;
  for(j=1;j<16;j++) //  j
  {
    t=0;
        for(i=0;i<N;i++)  //
        {
          d=0; s=i&j;
          while(s>0) { d^= s&1; s>>=1; }
          s=(*(h+i)>>(k-1))&1;
               if(d!=s) t++;

        }
        if(t<q) q=t;
        // t=N-t;
        if(t<q) q=t;
  }
  return q;
}
```