**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
“__” _____2022 р.
(підпис)

Дипломний проєкт

**на здобуття ступеня бакалавра
за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”**

на тему: *Метод прискореного модулярного множення для систем криптографічного захисту даних*

Виконав: студент 4 курсу, групи ІП-84
(шифр групи)

Кабір Лабіб Ахмед _____
(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) д.т.н., проф. Сімоненко В.П. _____
(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 – 2022 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

" Інженерія програмного забезпечення комп'ютерних систем"

спеціальності 121 " Інженерія програмного забезпечення "

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

"__" _____2022 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Кабір Лабіба Ахмед

1. Тема проєкту *Метод прискореного модулярного множення для систем криптографічного захисту даних*
керівник проєкту Марковський Олександр Петрович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 10 червня 2022 року № 1033-с.
2. Термін здачі студентом закінченого проєкту *11.06.2022*
3. Вихідні дані до проєкту *див. технічне завдання*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються) *Огляд існуючих методів і технологій виконаннямультиплікативних операцій модулярної арифметики над числами великої розрядності. Розробка методу прискореного модулярного множення з суміщенням редукції Монтгомері. Розробка програмних та апаратних засобів прискореного модулярного множення.*
5. Перелік графічного матеріалу (назви креслень з точним позначенням обов'язкових креслень) *блок-схема процедури формування таблиць передобчислень, схема електрична принципова співпроцесору прискореного модулярного множення, структурна схема співпроцесору прискореного модулярного множення*

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В.П., проф., д.т.н.		

7. Дата видачі завдання *26 вересня 2022*

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми* проєкту	*26.09.2021*	
2.	*Вивчення та аналіз завдання*	*26.09.2021-27.01.2022*	
4.	*Розробка алгоритму організації пошуку*	*27.01.2022-20.04.2022*	
5.	*Програмна реалізація системи*	*20.04.2022-20.05.2022*	
6.	*Оформлення пояснювальної записки*	*20.05.2022-04.06.2022*	
7.	*Захист програмного продукту*	*27.05.2022*	
8.	*Передзахист*	*11.06.2022*	
9.	*Захист*	*24.06.2022*	

Студент-дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

Анотація

Проєкт присвячено проблемі підвищення ефективності криптографічних алгоритмів, в основі яких лежить операція модулярного множення. На сучасному етапі розвитку інформаційних технологій, в умовах потреби підвищення ефективності криптографічних механізмів захисту даних, задача прискорення операцій модулярної арифметики є високопріоірітетною.

В роботі пропонується оригінальний спосіб прискорення модулярного множення, що базується на технології Монтгомері і передбачає собою використання передобчислень.

Доведено, що запропонований метод в 1.3 рази ефективніший за існуючі, з точки зору швидкодії.

Ключові слова: модулярна арифметика, модулярне множення, технологія Монтгомері, системи захисту інформації з відкритим ключем, асиметрична криптографія.

Abstract

The project is devoted to the problem of increasing the efficiency of cryptographic algorithms that are based on the operation of modular multiplication. The most effective of the currently existing methods of modular multiplication is Montgomery's method. However, at the current stage of the development of information technologies, in the conditions of the need to increase the efficiency of cryptographic mechanisms of data protection, the task of acceleration of modular multiplication is high priority.

The paper proposes an original method of accelerating modular exposure, based on Montgomery technology, and involves the use of recalculations. It is proved that the proposed method is 1.3 times more effective than the existing, in terms of speed.

Key words: modular arithmetic, modular multiplication, Montgomery's technology, public key information security systems, asymmetric cryptography.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екзем-пляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.468243.002 ТЗ***	Метод прискореного модулярного	5		
			множення для систем			
			криптографічного захисту даних			
			Технічне завдання			
	А4	***ІАЛЦ.468243.003 ПЗ***	Метод прискореного модулярного	69		
			множення для систем			
			криптографічного захисту даних			
			Пояснювальна записка			
	А1	***ІАЛЦ.468243.004 Д1***	Метод прискореного модулярного	1		
			множення для систем			
			криптографічного захисту даних			
			Блок схема алгоритму формування			
			таблиць передобчислень			
	А1	***ІАЛЦ. 468243.005 Д2***	Метод прискореного модулярного	1		
			множення для систем			
			криптографічного захисту даних			
			Принципова схема			
	А1	***ІАЛЦ.468243.006 Д3***	Метод прискореного модулярного	1		
			множення для систем			
			криптографічного захисту даних			
			Структурна схема			
	А4	***ІАЛЦ.468243.007 Д4***	Метод прискореного модулярного	11		
			множення для систем			
			криптографічного захисту даних			
			Текст програмного коду			

Зм	***Лист***	***№ докум.***	***Підп***	***Дата***	***ІАЛЦ.468243.001 ОА***		
Розроб		Кабір Л. А.			*Метод прискореного модулярного множення для систем криптографічного захисту даних* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Марковський О. П.					
Реценз.						***КПІ ім. Ігоря Сікорського», ФІОТ, ІП-84***	
Н. контр.		Сімоненко В. П					
Затв.							

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ
на тему: «Метод прискореного модулярного множення для систем криптографічного захисту даних»

Київ – 2022 р.

ЗМІСТ

Стр.

					ІАЛЦ.468243.002 ТЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Кабір Л. А.			*Метод прискореного модулярного множення для систем криптографічного захисту даних* Технічне завдання	Літ.	Аркуш 1 / Аркушів 5
Перевірив		Марковський О.П					
Реценз.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-84	
Н. Контр.		Сімоненко В. П.					
Затвердив							

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання поширюється на дослідження і розробку методу прискореного модулярного множення для систем криптографічного захисту даних.

Область застосування: захист інформації в комп'ютерних системах та мережах.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проекту по освітньо-професійній програмі "Інженерія програмного забезпечення комп'ютерних систем" спеціальності 121 "Інженерія програмного забезпечення", затверджене кафедрою Обчислювальної техніки Національного технічного Університету України "Київський Політехнічний інститут імені Ігоря Сікорського".

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Мета проєкту полягає в підвищенні швидкості обчислюваної реалізації базової операції широкого кола криптографічних алгоритмів захисту інформації – модулярного множення за рахунок суміщення в часі виконання множення та модулярної редукції Монтгомері з застосуванням передобчислень.

4. ДЖЕРЕЛА РОЗРОБКИ

4.1 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.

4.2 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.

4.3 Osadchyy V. The Order of Edwards and Montgomery Curves «, / V.Osadchyy // WSEAS Transactions on Mathematics, - 2020.- Vol. 19.- № 25, - P. 253-264.

4.4 Haches G. Montgomery multiplication with no final subtraction./ G. Haches, J.J. Quisquater // Cryptographic Hardware and Embedded System- CHES'2000. LNCS-1965, Springer-Verlag. — 2000.- P. 293-301.

4.5 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.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до продукту, що розробляється

5.1.1 Розроблюваний метод прискореного модулярного множення формує залишок від ділення добутку двох заданих чисел на фіксований модуль. Розрядність чисел і модуля обмежена значенням 4096.

Розрядність процесора може бути змінною і задається параметром, значення якого лежать в межах від 8-ми до 64-х.

5.1.2 Метод прискореного модулярного множення має реалізувати на програмному рівні суміщене в часі виконання множення та редукції Монтгомері шляхом використання передобчислень, які виконуються перед циклом множення.

5.1.3 Інший варіант модулярного множення має реалізувати на програмному рівні суміщення в часі виконання множення з обробкою змінної кількості розрядів множника та групову редукцію Монтгомері з використанням передобчислень.

5.1.4 Потрібно створити модифікації програми без вказаних вище методів прискорення модулярного множення і з ними для проведення експериментальної порівняльної оцінки швидкодії двох вказаних вище методів прискорення модулярного множення.

5.2Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Visual Studio 2017
- C++11

5.3Вимоги 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	26.09.2021-20.12.2021
Створення та узгодження технічного завдання	20.12.2021-15.01.2021
Вивчення літературних джерел	15.01.2021-27.01.2022
Розробка організації пошуку	27.01.2022-20.04.2022
Розробка програмної моделі	20.04.2022-10.05.2022
Відлагодження програми та виправлення помилок	10.05.2022-20.05.2022
Оформлення документації дипломного проєкту	20.05.2022-04.06.2022

Пояснювальна записка

до дипломного проєкту

на тему: «Метод прискореного модулярного множення для систем криптографічного захисту даних»

Київ – 2022 р.

ЗМІСТ

Стр.

ВСТУП

Розвиток хмарних технологій кардинальним чином змінив організацію В сучасних умовах розширення та поглиблення інформаційної інтеграції в усіх сферах людської діяльності, потребує удосконалення технологій захисту даних та контролю прав доступу до них. Необхідною передумовою ефективності інформаційної інтеграції є забезпечення надійного захисту інформації в інтегрованих системах обробки інформації та комп'ютерних мережах.

Зростання пропускної спроможності засобів передачі інформації визначає необхідність відповідного зростання продуктивності засобів захисту інформації в комп'ютерних мережах. Одним із найбільш поширених засобів захисту інформації, яка розповсюджується комп'ютерними мережами, є використання криптографічних алгоритмів та протоколів. Водночас, слід відзначити, що найбільш поширеними в комп'ютерних мережах є протоколи, що базуються на криптографічних алгоритмах з відкритим ключем.

Чільне місце в цих технологіях займають криптографічні алгоритми, які базуються на важко-вирішуваних задачах теорії чисел.

Базовою обчислювальною операцією алгоритмів цього класу виступає модулярне множення на числах, розрядність яких значно більша чим розрядність процесора. Час виконання таких операцій кубічно залежить від розрядності чисел з якими працюють алгоритми. Використання чисел розрядністю 2048 у більшості сучасних протоколів інформаційної безпеки, дозволяє досягти компромісу між швидкістю реалізації та рівня захищеності даних. Базовими обчислювальними операціями при їх реалізації є мультиплікативні операції модулярної арифметики. З огляду на забезпечення необхідного рівня надійності захисту, розрядність таких чисел становить декілька тисяч с перспективою подальшого збільшення. Відповідно, виконання операцій модулярного множення та експоненціювання над такими числами вимагає чималих обчислювальних ресурсів. Подальше збільшення розрядності

чисел, що використовуються в мережевих протоколах до 4096 буде мати наслідком багатократне підвищення обчислювальної складності, оскільки остання для операції експоненціювання пропорційна кубу довжини операндів. Зрозуміло, що таке зростання обчислювальної складності реалізації протоколів захисту інформації не може бути повною мірою компенсоване збільшенням продуктивності елементної бази обчислювальних засобів. Особливо гостро проблема продуктивності реалізації мережевих протоколів захисту інформації стоїть для малорозрядних мікроконтролерів, які становлять практично половину термінальних пристроїв сучасних мереж.

Останнє десятиліття процесу інформатизації суспільства знаменувалося появою та широким розповсюдженням хмарних технологій. Ці технології надають можливості необмежених обчислювальних потужностей. Оборотною стороною хмарних технологій полягає використання цих можливостей для порушення механізмів криптографічного захисту даних. Ця обставина призводить до об'єктивного зниження рівня захищеності широкого класу криптографічних алгоритмів в тому числі тих, які базуються на важко-розв'язуваних задачах теорії чисел. Єдина можливість збільшення рівня захищеності цих алгоритмів полягає в збільшенні розрядності чисел, з якими вони працюють.

З іншого боку це збільшує час виконання операцій модулярного експоненціювання, що суттєво зменшує швидкість реалізації алгоритмів, що базуються на цих операціях. В багатьох практично важливих застосуваннях протоколи криптографії з відкритим ключем реалізуються малопотужними термінальними пристроями. Для цього класу обчислювальних платформ потрібно розробити спеціальні методи прискореної реалізації мультиплікативних операцій модулярної арифметики, які б дозволяли прискорити реалізацію існуючих протоколів мережевої безпеки.

Мережеві технології все частіше застосовуються для задач телеконтролю і телекерування, що зумовлює жорсткі вимоги щодо продуктивності реалізації

протоколів роботи в мережах. Для згаданих застосувань, в якості термінальних пристроїв мереж переважно використовуються мікроконтролери, які мають порівняно низьку продуктивність.

З іншого боку, переважна більшість протоколів захисту інформації в мережах орієнтована на використання криптографічних алгоритмів на основі операцій модульної арифметики на числами, розрядність яких вимірюється нині тисячами. Реалізація таких складних обчислень на малорозрядних мікроконтролерах потребує значних витрат часу, що негативно впливає на ефективність мережевих технологій обробки інформації.

Таким чином, задача створення нових підходів до прискорення обчислювальної реалізації мультиплікативних операцій модулярної арифметики є актуальною та практично важливою для сучасного етапу розвитку комп’ютерних технологій.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ МЕТОДІВ І ТЕХНОЛОГІЙ ВИКОНАННЯ МУЛЬТИПЛІКАТИВНИХ ОПЕРАЦІЙ МОДУЛЯРНОЇ АРИФМЕТИКИ НАД ЧИСЛАМИ ВЕЛИКОЇ РОЗРЯДНОСТІ

Значення проблеми оперативності захисту інформації в локальних та глобальних комп'ютерних мережах визначається ростом інформаційної інтеграції, як основного чинника підвищення ефективності інформаційних технологій. Розширення сфер застосування швидкісних комп'ютерних мереж і збільшення їх пропускної спроможності потребує адекватного зменшення витрат часу на реалізацію процедур додаткової обробки даних при передачі в мережі. Це обумовлює необхідність підвищення продуктивності однієї з найбільш ресурсоємних складових обробки даних в мережах – протоколів забезпечення конфіденційності й контролю цілісності інформації. Для вирішення цієї нагальної задачі потрібно виявити специфічні особливості використання мультиплікативних операцій модулярної арифметики в системах криптографічного захисту інформації, виконати огляд існуючих методів та технологій виконання цих операцій на різних обчислювальних платформах, виявити резерви та можливості подальшого зростання продуктивності комп'ютерної реалізації цього класу операцій.

1.1 Аналіз особливостей використання мультиплікативних операцій модулярної арифметики в сучасних системах криптографічного захисту інформації

Для аналізу особливостей використання мультиплікативних операцій модулярної арифметики в сучасних системах криптографічного захисту інформації та оцінки різних алгоритмів захисту даних вузловим моментом є

визначення основних критеріїв їх якості. Значимість окремих критеріїв залежить від особливостей використання алгоритму в рамках того чи іншого криптографічного протоколу, а також від конкретних умов практичного застосування алгоритму. Більше того, для різних класів алгоритмів криптографічного захисту даних об’єктивно існують свої особливі критерії якості, які зкмовлені їх математичними характеристиками. Так, наприклад, для алгоритмів несиметричної криптографії, або алгоритмів з "з відкритим ключем" суттєвим критерієм якості виступає час формування закритих і відкритих ключів. Цей критерій не використовується для алгоритмів інших класів, зокрема хеш-алгоритмів та алгоритмів симетричного шифрування даних. В загальному вигляді, для оцінки якості всіх алгоритмів криптографічного захисту інформації можуть застосовуватися два базових критерія:

- Рівень захищеності інформації, який оцінюється назагал об’ємом ресурсів (пам'яті, даних, обчислювальних), потрібних для порушення створеного алгоритмом захисту;
- Час обчислювальної реалізації алгоритму програмними засобами чи шляхом використання спеціалізованих апаратних засобів;
- Стійкість захисту до впливу навмисних і ненавмисних завад в каналах передачі даних і при зберіганні зашифрованої інформації на магнітних, твердотільних та оптичних носіях.

Революційною подією в розвитку сучасної криптографії стала поява в 1978 році першого несиметричного алгоритму шифрування RSA [1], і якому для шифрування та дешифрування використовуються різні ключі. Поява такого проривного алгоритму як RSA надала могутній поштовх розвитку ефективних протоколів криптографічного захисту інформації, породив алгоритми цифрового підпису та схеми ідентифікації, в основі яких лежить концепція “нульових знань”. Базовою операцією алгоритму RSA, як і всіх його наступників є операція модулярного піднесення до степеню.

Математичною основою алгоритму RSA є мала теорема Ферма та узагальнення Ейлера малої теореми Ферма. Мала теорема Ферма постулює наступне: якщо m – просте число, то для будь-якого A< m виконується:

$$A^{m-1} \bmod m = 1 \quad \text{або} \quad A^m \bmod m = A. \qquad (1.1)$$

Наприклад: при m = 11 і A= 6: 6^{10} mod 11 = 60466176 mod 11 = 1. Іншими словами, мала теорема Ферма встановлю, що операція піднесення до степені по модулю m утворює цикл, який повторюється через m-1 кроків. Можна розглядати ситуацію , як ходіння по циклу довжиною m-1 з кроком довжиною E, який дорівнює першому ключу. Підбір другого ключа D здійснюється таким чином, щоб зробити по циклу певну кількість кроків довжиною E і пройти при цьому цілу кількість кіл і іще один крок.

Це дозволяє вибрати два ключі таким чином, щоб їх добуток являв собою мультиплікативну інверсію по модулю m. Наприклад, при модулі 11 в якості закриваючого ключа можна вибрати E = 3. Відповідно, для формування другого ключа Q потрібно віднайти розв’язок такого рівняння: $3 \cdot Q = 10 \cdot k + 1$, де k- ціле число. Це рівняння має розв'язок при k=2: Q = 7. Таким чином ключі описаної схеми несиметричного шифрування дорівнюють 5 і 7. Тобто справедливими є наступні тотожності: A^E mod 11 = B та B^Q mod 11 = A. Наприклад, якщо A=5, то B=5^3 mod 11 = 4, тобто шифротекстом є число B=4. Дешифрування відбувається з застосуванням відкриваючого ключа Q = 7: 4^7 mod 11 = 5.

Недолік розглянутої схеми несиметричного шифрування на основі малої теореми Ферма полягає в тому, що зловмисник знає модуль і задача отримання секретного ключа для нього повністю аналогічна задачі підбору секретного ключа тим, хто створює пару ключів.

В основі цього недоліку лежить те, що зловмисник, знаючи модуль m знає довжину циклу m-1 повторення. Для того, щоб він не міг швидко підібрати другий із пари ключів, він не має знати довжину циклу.

Рішення цієї проблеми надає використання узагальнення Ейлера малої теореми Ферма: Якщо p і q – прості числа, то для $A< q \cdot p$ виконується:

$$A^{(p-1)\cdot(q-1)} \bmod p \cdot q = 1. \quad (1.2)$$

Тобто, в цій схемі довжина циклу $L = (p-1)\cdot(q-1) \neq m = p \cdot q$. Наприклад: p=11; q=13; $L = (p-1)\cdot(q-1) = 120 \quad m = p \cdot q = 143$

Оскільки зловмисник знає модуль m, то класичний метод зламу RSA полягає в розкладанні числа (модуля) на два простих множника – задача факторизації.

При однаковому рівні захищеності алгоритм RSA на 3-4 порядки повільніший в порівнянні з алгоритмами симетричного шифрування при програмній реалізації. При апаратній реалізації – повільніший на 4-5 порядків.

Крім несиметричного шифрування, операція модулярного експоненціювання широко використовується в системах криптографічно строгої ідентифікації та автентифікації віддалених абонентів. такі системи, в математичному плані, будуються на незворотних перетворення теорії чисел.

Найбільшого поширення набули схеми криптографічно строгої ідентифікації, в основі яких лежать незворотні багатозначні перетворення класичної теорії чисел. Історично першою схемою ідентифікації такого типу стала FFSIS (Feige Fiat Shamir Identification Scheme) [2]. Цикл в цій схемі також базується на розширенні Ейлера малої теореми Ферма, тобто генерація ключів також здійснюється за рахунок того, абонент вибирає два секретні прості числа p і q, добуток яких утворює модуль M=$p \cdot q$. Саме тн, що абонент знає секретні коди обраних простих чисел p і q, а, отже, і довжину цикла повторення кодів при модулярному експоненціюванні: $(p-1)\cdot(q-1)$ дозволяє йому генерувати коректні сеансові паролі. Недолік цієї схеми полягає в тому, що передбачені нею обчислювальні процедури мають значну обчислювальну складність, що зумовлює великий час ідентифікації, пов'язаний з тим, що процес ідентифікації складається з великої кількості циклів акредитації, у кожному з яких

виконуються обміни даними та операції модулярного множення довгих чисел розрядністю *n* багато більшої розрядності процесора.

Інші, широко відомі схеми криптографічно строгої ідентифікації на основі незворотні багатозначні перетвореннь теорії чисел труднорозв'язних, такі як схема Guillou-Quisquater [3] та Schnorr [4], в яких передбачено використання меншого числа пересилок ідентифікаційної інформаціє, але процес ідентифікації передбачає застосування в системі модулярного експоненціювання, тобто обчислення $A^E \bmod M$. Усі компоненти цієї операції, включаючи показник E та модуль M, мають велику розрядність *n* (реально *n*=2048). Це визначає суттєву обчислювальну складність операцій ідентифікації і, відповідно, великі витрати часу.

Автентичність та цілісність документів забезпечується криптографічними механізмами цифрового підпису DSS [5] та ГОСТ Р34.10-94. В основі цих механізмів лежать операції модулярного експоненціювання, а основною операцією є модулярного експоненціювання, тобто обчислення $A^E \bmod M$.

Зокрема, стандартизований до використання на Україні алгоритм цифрового підпису DSS (Digital Signature Standard) [6] передбачає процедуру перевірки автентичності та цілісності електронних документів з використанням в якості незворотних перетворень відповідні задачі класичної теорії чисел. Аналіз математичних принципів, покладених в основу DSS показує, що в ньому використовується два простих числа *p* і *q*, для яких існує *g* таке, що:

процес ідентифікації складається з великої кількості циклів акредитації, у кожному з яких виконуються обміни даними та операції модулярного множення довгих чисел розрядністю *n* багато більшої розрядності процесора.

Інші, широко відомі схеми криптографічно строгої ідентифікації на основі незворотні багатозначні перетворень теорії чисел труднорозв'язних, такі як схема Guillou-Quisquater [7] та Schnorr [8], в яких передбачено використання меншого числа пересилок ідентифікаційної інформаціє, але процес

ідентифікації передбачає застосування в системі модулярного експоненціювання, тобто обчислення A^E mod M. Усі компоненти цієї операції, включаючи показник E та модуль M, мають велику розрядність *n* (реально *n*=2048). Це визначає суттєву обчислювальну складність операцій ідентифікації і, відповідно, великі витрати часу.

Автентичність та цілісність документів забезпечується криптографічними механізмами цифрового підпису DSS [9] та ГОСТ Р34.10-94. В основі цих механізмів лежать операції модулярного експоненціювання, а основною операцією є модулярного експоненціювання, тобто обчислення A^E mod M .

Зокрема, стандартизований до використання на Україні алгоритм цифрового підпису DSS (Digital Signature Standard) [10] передбачає процедуру перевірки автентичності та цілісності електронних документів з використанням в якості незворотних перетворень відповідні задачі класичної теорії чисел. Аналіз математичних принципів, покладених в основу DSS показує, що в ньому використовується два простих числа p і q, для яких існує g таке, що:

$$g^p \ mod \ q = 1. \qquad (1.3)$$

Тоді для будь-яких цілих x і m виконується $(g^{x+m\cdot q} \text{mod} q) \text{mod}\ p = (G^x \text{mod}\ q) \text{mod}\ p$.

Одержувач документу m' перевіряє цифровий підпис таким порядком.

1) Обчислюється хеш-сигнатура $H(m')$ отриманого документа m' з використанням відповідного стандартизованого хеш-алгоритма. Якщо в процесі передачі одержувачу документ не змінений, то $m = m'$ і, відповідно, $H(m) = H(m')$.
2) Обчислюється мультиплікативна інверсія s^{-1}.
3) Обчислюється значення $u_1 = (H(m') \cdot s^{-1}) \ mod \ q$.
4) Обчислюється значення $u_2 = (r \cdot s^{-1}) \ mod \ q$.
5) Обчислюється значення $v = ((g^{u_1} \cdot y^{u_2}) \ mod \ p) \ mod \ q$.

6) Якщо $v = r$, то вважається, що документ є оригінальним і він підписаний саме тим, хто знає закритий ключ x.

Побіжний аналіз протоколу DSS показує, що при реалізації п.3 і 4 потрібно здійснювати операцію модулярного множення, а виконання п. 5 наведеної вище процедури передбачає обчислення модулярної експоненти.

Таким чином, проведений аналіз показав, що значна частина сучасного арсеналу криптографічних механізмів захисту інформації використовує в якості базових операцій модулярне множення та модулярне експоненціювання над числами, довжина яких на порядок перевищує розрядність процесора. Найбільшою проблемою ефективності цих механізмів є повільність комп'ютерної реалізації, зумовленою значною обчислювальною складністю вказаних вище мультиплікативних операцій модулярної арифметики. Відповідно, ефективність цих криптографічних механізмів може бути підвищена за рахунок прискорення виконання мультиплікативних операцій модулярної арифметики.

Проведений аналіз особливостей практичного використання протоколів, основаних на криптографічних алгоритмах несиметричної криптографії, зокрема, алгоритмів шифрування RSA, El-Gamal, алгоритму цифрового підпису –DSS показує, а також алгоритмів криптографічно строгої ідентифікації FFSIS, Гіллоу-Квіскватера та Шнорра свідчить, що їх ключі змінюються відносно рідко, оскільки їх відкрита частина ідентифікує абонентів комп’ютерної мережі. Відповідно, модуль M, що є частиною відкритого ключа абонентів, можна розгляди як постійне число і зменшити, за рахунок цього, обчислювальну складність мультиплікативних операцій модулярної арифметики. Незмінність M дозволяє спростити, зокрема, виконання найбільш ресурсоємкої операції модулярної редукції у процесі множення в формі використання результатів передобчислень. Такі обчислення залежать тільки від значення модуля M, а тому виконуються одноразово при зміні модуля. Результати передобчислень

можуть зберігатися в табличній пам'яті і використовуватися при кожному виконанні операції модулярного множення.

1.2 Огляд методів прискореного множення

Проведений аналіз показав, що в більшості реальних алгоритмів криптографічного захисту, що базуються на використанні незворотних задач теорії чисел, незворотні перетворення реалізуються в формі модулярного експоненціювання, тобто обчислення $A^E \bmod m$. Виконання цієї операції зводиться до n циклів, де n- розрядність модуля m. В залежності від порядку, в якому аналізуються розряди коду експоненти E виділяють два базових алгоритми модулярного експоненціювання [11]:

При проходженні розрядів коду експоненти починаючи зі старших її розрядів використовується одна змінна R, в якій формується код результату. До виконання циклів значення R встановлюється в одиницю, а індекс j поточного розряду коду експоненти встановлюється в n: $j=n$. В кожному з послідовних циклів експоненціювання виконується модулярне піднесення до квадрату поточного значення R: $R = R \cdot R \bmod m$ після чого здійснюється аналіз значення поточного біту e_j коду експонента: якщо вказаний біт дорівнює одиниці, тобто $e_j=1$, то виконується модулярне множення поточного результату R на число A, яке підноситься до ступеня: $R= R \cdot A \bmod m$.

Цілком очевидно, що середня кількість операцій модулярного множення становить $1.5 \cdot n$. Оскількі всі ці операції лежать на критичному шляху алгоритму, то з цього випливає, що алгоритм модулярного експоненціюванні зі старших розрядів принципово не може бути розпаралелений на рівні операцій модулярного множення.

Другий класичний алгоритм модулярного експоненціювання, який передбачає проходженні розрядів коду експоненти від молодшого до старшого,

використовує дві довгих змінних: Q, в якій формуються двійкові ступені числа A та R, в якій формується код результату. До виконання циклів значення Q і R встановлюється в одиницю, а номер j поточного розряду експоненти E встановлюється в одиницю: j=1. В кожному з послідовних циклів експоненціювання виконується модулярне піднесення до квадрату поточного значення Q: $Q = Q \cdot Q \bmod m$ після чого здійснюється аналіз значення поточного біту e_j експоненти: якщо вказаний біт дорівнює одиниці, тобто e_j=1, то виконується модулярне множення поточного результату R на число Q : $R = R \cdot Q \bmod m$.

Цілком очевидно, що середня кількість операцій модулярного множення для цього різновиду алгоритму модулярного експонеціювання становить $1.5 \cdot n$. Проте, на відміну від попереднього алгоритму, операції модулярного піднесення до квадрату і модулярного множення можуть бути суміщені в часі в рамках одного циклу. Тобто, теоретично, алгоритм може за рахунок розпаралелювання, виконуватися вдвічі швидше в порівнянні з першим алгоритмом.

З проведеного аналізу цілком очевидним є той факт, що кардинальне прискорення модулярного експоненціювання може бути здійснене лише на рівні операції модулярного множення.

Вважаючи на практичну важливість ефективної реалізації механізмів захисту інформації, що мають за основу операції модулярної арифметики, до теперішнього часу запропоновано ряд алгоритмів прискореного модулярного множення.

При виконанні модулярного множення чисел довжиною n на k-розрядному процесорі, число A представляється у вигляді s фрагментів, розрядність яких дорівнює розрядності процесору, тобто k, відповідно, чисельне значення $s=n/k$: $A=\{a_{s-1},\ldots,a_1,a_0\}$, $0 \le a_j < 2^k$: $A = \sum_{j=0}^{s-1} a_j \cdot 2^{j \cdot k}$. Базовою операцією модулярної арифметики в протоколах захисту інформації є

модулярне множення: $R=A \cdot B \bmod m$, причому $2^{n-1} \le m < 2^n$, $A<m$, $B<m$. Кожне з чисел A, B і m що приймають участь в операції складаються з s фрагментів: $A = \sum_{j=0}^{s-1} a_j \cdot 2^{j \cdot k}, B = \sum_{j=0}^{s-1} b_j \cdot 2^{j \cdot k}, m = \sum_{j=0}^{s-1} m_j \cdot 2^{j \cdot k}$, де a_j, b_j, m_j – k-розрядні фрагменти, $j \in \{0,\ldots,s-1\}$.

Об'єм обчислень процедур модулярного множення при їх реалізації на різних обчислювальних платформах обчислюється числом використаних в них операцій процесорного множення .

Операція модулярного множення складається з двох частин: множення, тобто обчислення добутку B·A та модулярної редукції – віднаходження залишку B·A по модулю m: B·A mod m. Ці дві складові можуть виконуватися окремо, а можуть поєднуватися у часі.

Обчислення модулярного добутку B·A зазвичай виконується по схемі секційного множення: тобто кожен із фрагментів числа B перемножається на всі фрагменти числа A. При цьому кількість операцій процесорного множення для обчислення добутку чисел, що складаються з s фрагментів становить s^2.

Основною перевагою роздільного виконання множення є те, що при цьому можна застосовувати існуючі способи прискореного множення, такі як метод Карацуби [12], метод Фюрера [13], метод Toom-Cook [14,15] та FFT-базовані методи [7]. Ці методи секційного множення, в яких кількість процесорних множень менша за s^2.

Метод Карацуби дозволяє при організації множення чисел, які складаються з k окремих секцій зменшити кількість процесорних множень з s^2 до. $s^{\log_2 3} = s^{1.585}$.

Приклад, що ілюструє використання технології швидкого множення Карацуби. Припустимо, що обчислюється добуток A·B. Кожне із чисел A і B складаються з трьох секцій s=3, довжина яких дорівнює розрядності r процесора. . Тобто $A=a_3 \cdot 2^{2 \cdot r} + a_2 \cdot 2^r + a_1$ $B=b_3 \cdot 2^{2 \cdot r} + b_2 \cdot 2^r + b_1$.

За технологією Карацуби обчислюються добутки:

$P_1=(a_3+a_2+a_1)\cdot(b_3+b_2+b_1) = a_1\cdot b_1 + a_1\cdot b_2 + a_1\cdot b_3 + a_2\cdot b_1 + a_2\cdot b_2 + a_2\cdot b_3 + a_3\cdot b_1 + a_3\cdot b_2 + + a_3\cdot b_3$,

$P_2=(a_3+a_2)\cdot(b_3+b_2) = a_3\cdot b_3 + a_3\cdot b_2 + a_2\cdot b_3 + a_2\cdot b_2$,

$P_3=(a_2+a_1)\cdot(b_2+b_1) = a_2\cdot b_2 + a_2\cdot b_1 + a_1\cdot b_2 + a_1\cdot b_1$.

Після цього результат обчислюється у наступному вигляді:

$A\cdot B = a_3\cdot b_3\cdot 2^{4\cdot r} + (P_2 - a_3\cdot b_3 - a_2\cdot b_2)\cdot 2^{3\cdot r} + (P_1 - P_2 - P_3 + 2\cdot a_2\cdot b_2)\cdot 2^{2\cdot r} + +(P_3 - a_1\cdot b_1 + +a_2\cdot b_2)\cdot 2^r + a_1\cdot b_1$.

Цілком очевидно, що при такій організації обчислення добутку необхідно виконати лише 6 операцій процесорного множення, що співпадає з теоретичною оцінкою кількості процесорних множень для операції при використанні технології Карацуби: $s^{1.565} = 3^{1.565} = 5.7$.

Організація паралельного виконання алгоритмів цього класу найчастіше реалізується на рівні секцій, довжина яких дорівнює розрядності k процесора. Це дозволяє розпаралелити виконання множення на рівні секцій під час використання мультипроцесорних комп'ютерних систем. Важливою перевагою множення по секціях є можливість практично вдвічі скоротити кількість секційних множень при реалізації модулярного зведення квадрат. Саме ця операція становить 2/3 обсягу обчислень, що здійснюються при модулярному експоненціюванні [16].

Таким чином, здійснений аналіз процедур секційного множення чисел великої розрядності, що складаються з s фрагментів показав, що кількість процесорних множень за рахунок використання спеціальних методів може бути зменшена з s^2 до рівня s^u, де u стале число, що залежить від методу.

1.3 Аналіз технологій модулярної редукції

Операція модулярної редукції, тобто віднаходження залишку B·A по модулю *m*: B·A mod *m* виконується значно повільніше в порівнянні з операцією множення довгих чисел. Це зумовлено тим, отримання залишку B·A по модулю *m* вимагає істотно більших обчислювальних ресурсів, оскільки неможливо використовувати для цієї мети вбудовану операцію процесорного ділення. Це означає, що операцію редукції потрібно виконувати побітово. А при цьому кількість процесорних операцій не може бути меншою за *n*.

При використанні класичних алгоритмів ділення з відновленням або без відновлення залишку, модулярна редукції виконується з використанням операції цілочисельного ділення *2·k*-розрядного діленого на *k*–розрядний подільник з одержанням части та залишку. Приймаючи до уваги те, що операція ділення *n*–розрядних чисел на *k*–розрядному процесорі (*n>>k*) виконується доволі неефективно, реалізація редукції в класичному алгоритмі потребує *(s+2.5)* операцій процесорного множення і *s* операцій процесорного ділення.

Тому основні зусилля досліджень останніх років спрямовані на прискорення редукції.

Тут можна назвати два напрями: використання передобчислень і заміна неефективної для редукції операції процесорного ділення на процесорні множення. Перший напрямок базується на тому, що у реальних криптографічних системах модуль *m* є частиною відкритого ключа та змінюється відносно рідко. Тому в рамках цього напряму підвищення швидкості виконання редукції досягається за рахунок якнайбільшого використання результатів передобчислень, що залежать тільки від модуля [17] и можуть виконуватися один раз.

Число P =A·B може бути представлено у вигляді суми двох компонент: (n-1)-розрядного числа L, яке співпадає з n-1 молодшими розрядами A·B і n-розрядного числа H, яке складається з n старших розрядів A·B:

$$P = \sum_{j=0}^{2\cdot n} p_j \cdot 2^j = L + H,\ L' = \sum_{j=n}^{n\cdot 2} p_j \cdot 2^j, L = \sum_{i=0}^{n-1} p_i \cdot 2^i. \quad (1.4)$$

Згідно з властивістю конгруентності для модулярної редукції, залишок P mod m може бути представлений у вигляді модулярної редукції суми залишків L та H:

$$P\ mod\ m = (\sum_{j=0}^{2\cdot n} p_j \cdot 2^j)\ mod\ m = (L + H)\ mod\ m =$$

$$= (H\ mod\ m + L\ mod\ m)\ mod\ m. \quad (1.5)$$

В силу того, що старший, n-й розряд модуля m дорівнює одиниці, а L являє собою (n-1)-розрядне число, то $L<m$ і, відповідно, L mod m = L. Число H має тільки k значущих розрядів (інші n молодших розрядів H дорівнюють нулю). Таким чином, H і відповідно H mod m може приймати тільки 2^n різних значень. Всі можливі n-розрядні значення $H\ mod\ m$ для відповідних значень H можуть бути попередньо обчислені та збережені в пам’яті у вигляді таблиці. Проте очевидно, що об’єм пам'яті для зберігання такої таблиці доволі великий і для багатьох обчислювальних платформ не може бути реалізований з технічних причин.

Тому на практиці частіше попередньо обчислюються і зберігаються значення 2^{n+1} mod m, 2^{n+2} mod m,… 2^{2n-1} mod m. Загальна кількість таких значень дорівнює n, тобто не перевищує декількох тисяч, що легко реалізується з використанням вбудованої постійної памяті навіть для малопотужних мікроконтролерів.

Обчислення редукції добутку P в цьому варіанті здійснюється за наступною формулою:

$$P\ mod\ m = L + \sum_{i=n}^{2\cdot n} p_i \cdot T[2^i\ mod\ m]. \quad (1.6)$$

Обчислювальна складність реалізації модулярної редукції при цьому визначається в середньому $n/2$ операціями додавання n–розрядних чисел. Виходячи з цього, час реалізації процедури модулярної редукції не перевищує $0.5 \cdot n \cdot t_a$, Об'єм пам'яті, потрібний для зберігання попередньо обчислених всіх можливих значень $T(Z)$ складає n^2 бітів.

Описаний підхід особливо ефективний при реалізації модулярного множення на малорозрядних мікропроцесорах, мікроконтролерах та смарт-картах. Для вказаних застосувань, об'єм пам'яті для зберігання результатів передобчислень у вигляді спеціальних таблиць при постійному модулі виявляється доволі прийнятним з точку зору реалізації для більшості використань. Наприклад, для реалізації прискореного модулярного множення 2048-розрядних чисел на 8-розрядному мікроконтролері об'єм пам'яті становить близько 512 Кілобайтів.

Використання багатосекційних таблиць передобчислень дозволяє суттєво зменшити об'єм пам'яті для їх зберігання. Нижче, наведено приклад виконання редукції з використанням передобчислень. Якщо модуль M = 13. Число P = $10010110_2 = 150_{10}$, то правильний результат P mod M = 150 mod 13 = 7 . Оскільки модуль, що є частиною відкритого ключа, практично незмінний, то попередньо обчислюється:

10000_2 mod M = 16 mod 13 = 3 P = 1 0 0 1 0110_2

100000_2 mod M = 32 mod 13 = 6

1000000_2 mod M = 64 mod 13 = 12 P mod M = (11 + 3 + 6) =

10000000_2 mod M = 128 mod 13 = 11 = 10100 = 3 + 4 = 7

Обчислення суми добре розпаралелюється і може бути використане для прискорення обчислень. Іншим шляхом прискорення є збільшення пам'яті таблиць передобчислень [18].

Інші напрями досліджень полягають в переході від операції ділення до множення та зсуву. Щоб уникнути операції ділення багаторозрядних чисел при

реалізації модулярної редукції запропоновано ряд специфічних алгоритмів, найбільш відомими з яких є алгоритми Баретта та Монгомері.

Технологія модулярної редукції Барретта також передбачає здійснення певних передобчислень. Наприклад, є добуток P=A·B, модуль m. Потрібно обчислити ξ = P mod m. За методом Барретта шукається найменше q, для якого: $P - q \cdot m < m$ у вигляді добутку:

$$q = \frac{P}{m} = \frac{P}{2^n} \cdot \frac{\omega}{2^n} \quad \text{де } \omega = \frac{2^{2 \cdot n}}{m}. \qquad (1.7)$$

Цілком очевидно, що чисельне значення ω залежить лише від модуля і може бути обчислений один раз.

Наприклад: P = 150, модуль m = 13 . Тоді правильний результат ξ = P mod M = 150 mod 13 = 7. Значення n = 4

$$\omega = \frac{2^{2 \cdot n}}{M} = \frac{2^8}{13} = \frac{256}{13} = 19 \qquad \frac{\omega}{2^n} = \frac{19}{16} = 1.1875$$

Ділення на ступінь 2 здійснюється зсувом:

$\frac{P}{2^4}$ = 1 0 0 1 . 0 1 1 0 = 9.375

9.375 · 1.1875 = 11.1328125 $\rightarrow q$ = 11

ξ = P - q·M = 150 - 11·13 = 7

Висновок: редукція Баррета реалізується двома операціями множення:

$\frac{P}{2^n} \cdot \frac{\omega}{2^n}$ та $q \cdot m$

Інша технологія виконання редукції добутку $P = A \cdot B$ за допомогою операцій множення та зсувів запропонована П. Монтгомері [19,20]. Ця технологія передбачає віднайдення найменшого цілого числа g такого, що сума $P+g \cdot m$ є кратною $2 \cdot n$.

Технологія Монтгомері може застосовуватися як до попередньо обчисленого добутку $P = A \cdot B$, так і реалізуватися в рамках суміщення множення та редукції. На практиці модулярне множення з використанням редукції

Монтгомері ефективне лише в якості базової операції модулярного експоненціювання.

Застосування модулярного множення з використанням технології Монтгомері вимагає додаткових операцій, що виконуються до і після множення. При обчисленні $A^E \bmod m$, множення виконується тисячі разів, проте додаткові операції виконуються лише до та після модулярного експоненціювання і, відповідно не мають суттєвого впливу на обчислювальну складність модулярного експоненціювання.

Класичний алгоритм модулярного множення за технологією Монтгомері фактично суміщає в часі виконання власне множення та редукції. Це дозволяє виключити необхідність роботи з числами, розрядність яких вдвічі перевищує n. Перевагою технології Монтгомері є і те, що одна операція операція зсуву виконується для власне множення і редукції.

Вихідними даними для модулярного множення з застосуванням редукції Монтгомері є: модуль m, співмножники A і B, а також попередньо обчислена модулярна інверсія Q' числа 2^n за модулем 2^n: $Q \cdot 2^n \bmod m = 1$. В процесі виконання алгоритму ітераційно формується n-розрядний код результату $R = A \cdot B \cdot Q \underline{\bmod} m$, при обробці розрядів множника, починаючи з молодших розрядів. Проміжний результат при цьому зсувається праворуч. Відповідно, алгоритм модулярного множення з використанням технології Монтгомері можна представити з використанням нотацій C++ в наступному вигляді:

1. $R = 0$.

2. *for* (j=1; j <= n; j++)

```
 {
    if (a_j = =1 )
      R += B;
    if (r_0 = = 1 )
      R += m ;
```

R>>=1;

}

3. *if* ($R \geq m$) R - = m.

Щоб отримати правильний результат $A \cdot B \ mod \ M$ потрібно виконати додаткові обчислення над R : помножити одержаний результат на 2^n .

Проведений аналіз алгоритму модулярного множення з використанням редукції Монтгомері показав, що його обчислювальна складність, що виконується за цим алгоритмом, без урахування додаткових перетворень близька до теоретичного мінімуму. Аналіз чисельних публікацій, присвячений розвитку ідей алгоритму Монтгомері значною мірою підтверджують цю оцінку.

Висновки до розділу 1

В результаті проведених досліджень, які складають перший розділ роботи, і які направлені на аналіз використання мультиплікативних операцій модулярної арифметики в сучасних засобах криптографічного захисту інформації, критичний аналіз існуючих алгоритмів їх обчислювальної реалізації, а також виявлення можливостей її прискорення, зроблені наступні висновки:

1. Проведений аналіз показав, що значна частина сучасного арсеналу криптографічних механізмів захисту інформації використовує в якості базових операцій модулярне множення та модулярне експоненціювання над числами, довжина яких на порядок перевищує розрядність процесора. Найбільшою проблемою ефективності цих механізмів є повільність комп'ютерної реалізації, зумовленою значною обчислювальною складністю вказаних вище мультиплікативних операцій модулярної арифметики. Відповідно, ефективність цих криптографічних механізмів може бути підвищена за рахунок прискорення виконання мультиплікативних операцій модулярної арифметики.
2. Проведений аналіз особливостей практичного використання протоколів, основаних на крипто¬графічних алгоритмах несиметричної криптографії, зокрема, алгоритмів шифрування RSA, El-Gamal, алго¬рит¬му цифрового підпису –DSS показує, а також алгоритмів криптографічно строгої ідентифікації FFSIS, Гіллоу-Квіскватера та Шнорра свідчить, що їх ключі змінюються відносно рідко, оскільки їх відкрита частина ідентифікує абонентів комп’ютерної мережі. Відповідно, модуль M, що є частиною відкритого ключа абонентів, можна розгляди як постійне число і зменшити, за рахунок цього, обчислювальну складність мульти-плікативних операцій модулярної арифметики.

3. Проведений аналіз існуючих алгоритмів обчислення базової операції механізмів захисту інформації з відкритим ключем – модулярного експоненціювання показав, що вони мають строго послідовний характер і не можуть бути розпаралелені: кардинальне прискорення модулярного експоненціювання може бути здійснене лише на рівні операції модулярного множення.
4. Аналіз процедур секційного множення чисел великої розрядності, що складаються з s фрагментів показав, що кількість процесорних множень за рахунок використання спеціальних методів може бути зменшена з s^2 до рівня s^u, де u стале число, що залежить від методу.
5. Аналітичний огляд існуючих технологій модулярної редукції показав, що найбільший потенціал для подальшого прискорення модулярного множення надає технологія Монтгомері, яка дозволяє сумістити в часі виконання множення та віднаходження залишку.

РОЗДІЛ 2
РОЗРОБКА МЕТОДУ ПРИСКОРЕНОГО МОДУЛЯРНОГО МНОЖЕННЯ З СУМІЩЕННЯМ РЕДУКЦІЇ МОНТГОМЕРІ

Широкий клас важливих для практики криптографічних механізмів, таких, зокрема як алгоритми шифрування з відкритим ключем, протоколи цифрового підпису, обміну ключами та інші мають в якості математичної бази мультиплікативні операції модулярної арифметики, які виконуються над числами, розрядність яких значно перевищує розрядність процесора. Реально сьогодні в більшості протоколів передбачена розрядність 2048 чи 4096. Класичний метод зламу цих алгоритмів полягає в факторизації, тобто розкладенні відкритого модуля на два простих множника. Складність цієї задачі повністю визначається розрядністю модуля. Це означає, що єдиним способом підвищення рівня захисту для цих алгоритмів є збільшення розрядності. Широке застосування хмарних технологій потенційно надає зловмисникам доступ до значних обчислювальних потужностей, які можуть бути використані для факторизації модуля. Це диктує необхідність підвищення рівня захищеності, тобто розрядності чисел. Це різко уповільнює виконання мультиплікативних операцій модулярної арифметики. Добре відомо, що для модулярного експоненціювання збільшення вдвоє розрядності має наслідком збільшення об’єму обчислень у вісім разів. Тобто маємо кубічну залежність.

Алгоритм модулярного експоненціювання має послідовний характер, тобто його виконання не може бути організовано паралельно. Єдиним виходом зостається пошук шляхів прискорення основної складової цього алгоритму – модулярного множення.

Модулярне множення складається з власне операції множення та модулярної редукції, тобто віднаходженні залишку від ділення добутку на модуль. Реально редукція здійснюється з використанім технологій Баррета або Монтгомері.

Доведено, що більше ефективними є алгоритми модулярного множення з редукцією Монтгомері, в яких ці дві операції суміщуються в часі, тобто не відбувається накопичення повного добутку.

2.1 Розробка методу модулярного множення з суміщенням додавання множника та модуля для редукції Монтгомері

Існує два варіанти цього алгоритму. Різниця між цими варіантами полягає в тому, що у першому варіанті прохід відбувається від старшого до молодшого розряду коду експоненти, а в другому навпаки – від молодшого до старшого.

Перший варіант цього алгоритму зводиться до такої послідовності дій:

1) В допоміжну змінну R записується 1. В лічильник j записується 0;
2) Обчислення медулярного піднесення до квадрату змінної R ( )
3) Якщо молодший розряд E дорівнює 1, то обчислюється модулярне множення R на A ( );
4) Якщо j<=n , то до лічильника додається 1 (j = j + 1) та виконується повернення до 2 пункту.

Алгоритм модулярного множення ($C * D \ mod \ M$) за методом Монгомері зводиться до виконання таких дій:

1) В допоміжну змінну R записується та лічильник j записується 0 (R=0, j=0);
2) Якщо j розряд D дорівнює 1, то до R додається C ($R=R+C$);
3) Якщо R – непарне число, то до R додається M ($R=R+M$);
4) Виконується зсув на один розряд вправо числа R ($R = R/2$);
5) Запропонований алгоритм прискореного модулярного множення Монгомері зводиться до такої послідовності дій:
6) Якщо j<=n, то лічильник збільшуємо на 1 (j = j + 1) та виконується повернення до повторного виконання 2 пункту.

Запропонований алгоритм прискореного модулярного множення

Монгомері зводиться до такої послідовності дій:

1) В допоміжну змінну R записується та лічильник j записується 0 (R=0, j=0). В допоміжну змінну Q обраховується сума чисел C та M ($Q = C + M$)
2) Якщо поточний розряд числа D (d_j) та молодший розряд числа R (r_0) дорівнюють 0, то виконується перехід до 6 пункту;
3) Якщо d_j= 0 та r_0 = 1, то обчислюється сума R та M ($R = R + M$) та виконується перехід до 6 пункту;
4) Якщо d_j= 1, r_0 та молодший розряд числа C(c_0) обидва дорівнюють 0 або 1 , то обчислюється сума R та C ($R = R + C$) та виконується перехід до 6 пункту;
5) Якщо d_j= 1та справджується одна із умов:
   - r_0 = 0 та c_0 = 1;
   - r_0 = 1 та c_0 = 0,

то обчислюється сума R та Q ($R = R + Q$) та виконується перехід до 6 пункту;

6) Виконується зсув на один розряд вправо числа R ($R = R/2$);
7) Якщо j<=n, то j збільшуємо на 1 ($j = j + 1$) та виконується повернення до повторного виконання 2 пункту.

Аналіз наведеного алгоритму засвідчує, що операція модулярного експоненціювання складається з модулярного піднесення числа до квадрату й модулярного множення. Якщо зважити на те, що для комп'ютера виконання цих операцій рівнозначне за часом й обчислювальною складністю, то значна частина часу витрачається саме на здійснення модулярного множення. Його прискорення значно скоротить загальний час обчислення операції модулярного експоненціювання.

Операція модулярного множення може виконуватися двома способами. Перший з них полягає в послідовному виконанні множення двох чисел і редукції результату. Другий спосіб виконує ці операції одночасно в кілька ітерацій.

Одним з найбільш швидкодіючих методів модулярного множення є метод Монтгомері. Цей метод працює за другим способом і полягає в суміщенні операцій модулярного множення й редукції результату. Алгоритм модулярного множення X·Y mod M за методом Монтгомері зводиться до наступної послідовності дій:

1) Початкові значення результату *R* та лічильника *j* рівні нулю: R=0,j=0.
2) Якщо Y_j, *j*-ий розряд *Y*, дорівнює одиниці, то в змінну *R* обчислюється сума *R* та *X*: R=R+X;
3) Якщо R непарне, то в змінну *R* обчислюється сума *R* та *Z*: R=R+Z.
4) Виконується цілочисельне ділення R на 2, що на практиці є рівнозначним операції зсуву на один розряд вправо: R=R/2.
5) Якщо $j < n - 1$, то *j* інкрементується j=j+1 та перехід до пункту 2.
6) Якщо число R більше за M, то обчислюється різниця R та M: R=R-M.

Технологія Монтгомері значною мірою прискорює модулярне множення, однак, в сучасних умовах розвитку інформаційних технологій, існує необхідність підвищення ефективності криптографічних алгоритмів, в основі яких лежать операції модулярної арифметики.

На рис.2.1 показано класичний алгоритм модулярного множення з редукцією Монтгомері. Як видно з алгоритму, при обробці одного розряду множника можуть виконуватися дві операції додавання: це додавання множимого та додавання модуля.

Процедура множення Монтгомері складається за наступної послідовності дій:

1. Y = 0, лічильник j циклів (номер поточного двійкового розряду множника) встановлюється в нуль: j=0.
2. Обчислюється сума $Y = Y + b_j \cdot A$

3. Якщо значення поточного результату непарне, тобто $y_0 = 1$, то до поточного результату додається модуль Y = Y + M
4. Здійснюється зсув праворуч поточного результату: Y = Y / 2
5. Якщо j < *n*-1, то виконується інкремент *j*: *j*=*j*+1 і повернення на повторне виконання п. 2.
6. Якщо Y ≥ M, то Y = Y-M.

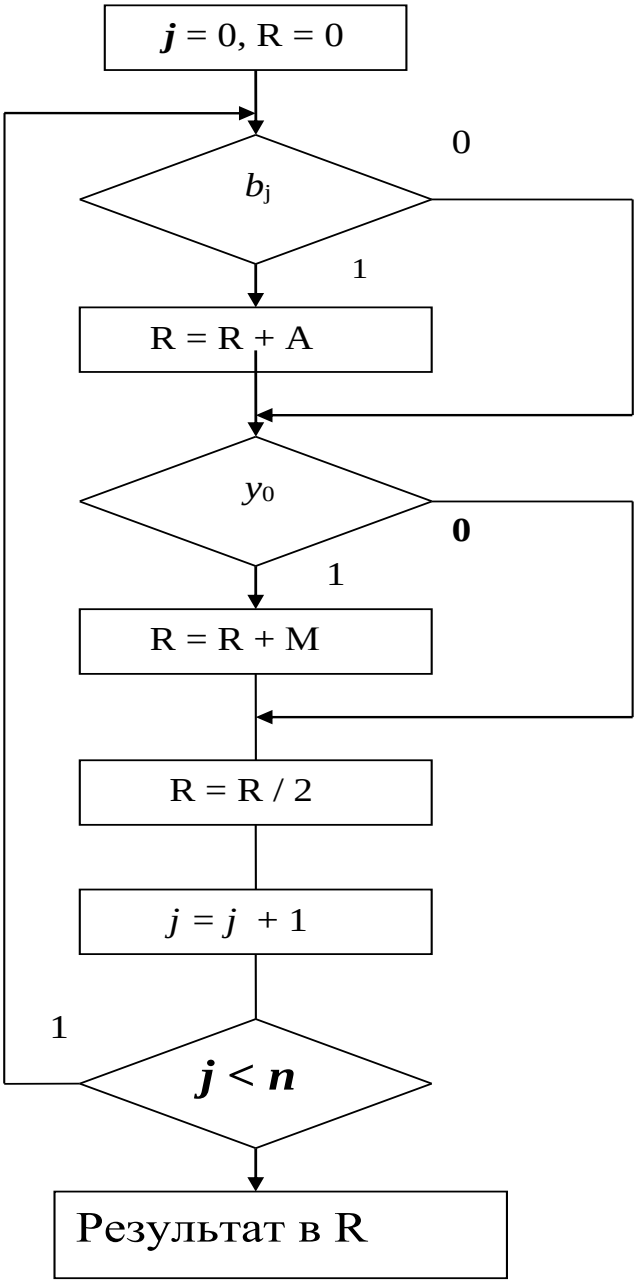


Рис.2.1 Класичний алгоритм модулярного множення з суміщеною редукцією Монтгомері

Функціонування описаної процедури може бути ілюстровано наступним прикладом. Нехай модуль M = 10011_2 = 19_{10}, відповідно, n=5; допоміжна величина R=2^5 = 32, а її мультиплікативна інверсія по модулю M дорівнює R^{-1} = 3. Виконується множення числа A=10001_2 = 17_{10} на число B= 01011_2 = 11_{10} .

Таблиця 2.1

Динаміка покрокових змін величин, що задіяні в процедурі множення Монтгомері чисел A=17 і B=12

j	b_j	Поточний результат Y				
		На початку цикла	Після додавання множимого	y_0	Після додавання модуля	Після зсуву
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	1	0	0+17 = 17	1	17 + 19 = 36	36/2 = 18
3	1	18	18+17= 35	1	35 + 19 = 54	54/2 = 27
4	0	27	27	1	27 + 19 = 46	46/2 = 23

Так як Y > M (23 > 19), то згідно п.6: Y=Y-M = 23-19 = 4

Отриманий результат Y=4 відповідає значенню A·B·R^{-1} mod M = 17·12·3 mod 19 = 4. Для отримання істинного значення A·B mod M результат Y можна помножити на R по модулю M: A·B mod M = Y· R mod M = 4·32 mod 19 = 14.

Мета досліджень полягає в підвищенні швидкодії криптографічних систем шляхом прискорення операції модулярного множення за рахунок суміщення додавання множника та корекції Монтгомері.

Для досягнення поставленої задачі пропонується удосконалити метод модулярного множення за технологією Монтгомері.

Вдосконалення зводиться до наступної послідовності дій:

1) Початкові значення результату R та лічильника j рівні 0: R=0; j=0. Результат передобчислення суми A та M зберігається в змінну Q: Q =A+M.
2) Якщо b_j, j-й розряд множника B та r_0 - молодший розряд результату дорівнюють 0: $y_j = r_0 = 0$, то перехід до пункту 6.
3) Якщо $y_j = 0$ та $r_0 = 1$, то R=R+A. Перехід до 6 пункту;
4) Якщо $y_j = 1$ та $a_0 = r_0$, то R=R+Q. Перехід до пункту 6.

5) Якщо $y_j = 1$ та розряд r не дорівнює розряду c_0, то R=R+Q.
6) Виконується цілочисельне ділення R на 2, що на практиці є рівнозначним операції зсуву на один розряд вправо: R=R/2.
7) Якщо $j < n$, то j інкрементується j=j+1 та виконується перехід до пункту 2.
8) Якщо $R > M$, то R=R-M.

Для спрощення всі дані можуть бути попередньо занесені в таблицю передобчислень, яка адресується значеннями трьох бітів. Тобі алгоритм прискореного модулярного множення матиме вигляд, показаний на рис.2.2. Таблиця результатів передобчислень наведена в таблиці 2.2.

Таблиці 2.2

Таблиця результатів передобчислень

Бітові значення			Результат операції
Поточний біт множника $\boldsymbol{b_j}$	Молодший біт проміжного результату $\boldsymbol{y_0}$	Молодший біт множимого $\boldsymbol{a_0}$	
0	0	0	0
0	0	1	0
0	1	0	M
0	1	1	M
1	0	0	A + M
1	0	1	A + M
1	1	0	A
1	1	1	A

Ефективність запропонованого методу, пропонується оцінювати коефіцієнтом прискорення β, що визначається відношенням часу T виконання модулярного множення за методом Монтгомері до часу T_m здійснення обчислень запропонованим методом:

$$\beta = \frac{T}{T_m}. \quad (2.1)$$

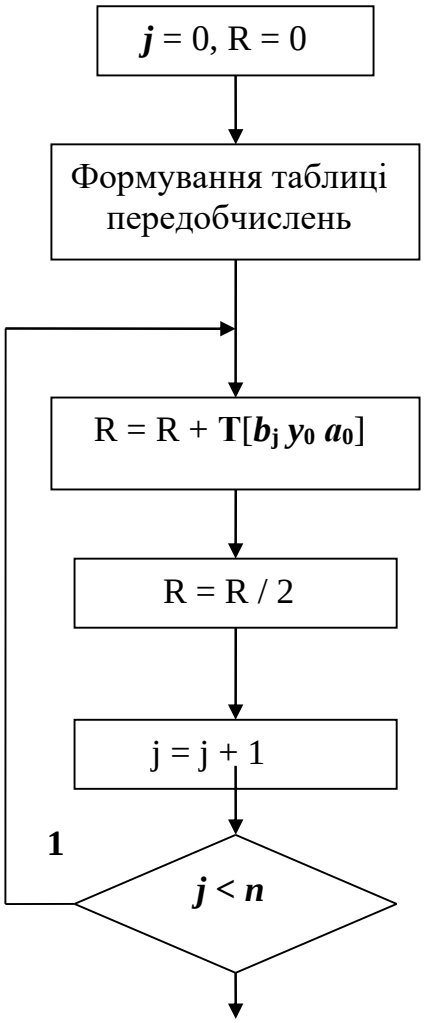


Рис. 2.2 Модифікований алгоритм модулярного множення з суміщеною редукцією Монтгомері

Час *T* виконання модулярного множення за методом Монтгомері, тобто згідно алгоритму, показаному на рис.2.1 можна оцінювати кількістю операцій додавання *n*-розрядних чисел. В класичному методі в середньому на одному циклі виконання алгоритму виконується одна операція додавання, відповідно, в середньому, реалізація класичного алгоритму потребує *n* операцій додавання.

В розробленому методі одночасно здійснюється додавання модуля й множеного до поточного результату при одиничному значенні поточного розряду множника і якщо молодший розряд результату й множеного рівні між собою. Відповідно, операція додавання здійснюється в 6-ти варіантах змінних із 8-ми можливих, тобто з ймовірністю 6/8. Середня кількість операцій

додавання n-розрядних чисел в запропонованому методі становить, таким чином $6 \cdot n/8$. Таким чином маємо $\beta = \frac{n \cdot 8}{6 \cdot n} = 1.33$

Отже, запропонований метод з використанням одночасного додавання до поточного результату суми модуля й множника прискорює операцію модулярного множення на 30%.

В результаті проведених студій був досліджений, розроблений і обґрунтований оригінальний метод прискорення модулярного множення орієнтований на застосування системами захисту інформації. В основу цього методу покладено технологію Монтгомері.

Суть запропонованого методу полягає в проведенні аналізу множників і поточного результату. Розряди цих чисел аналізуються таким чином, щоб виявити ситуацію, коли можна об'єднувати операції додавання модуля й множеного. Це дозволяє скоротити кількість операцій необхідних для виконання модулярного множення.

Було теоретично та експериментально доведено що розроблений метод пришвидшує операцію модулярного множення в 1.3 рази.

2.2 Розробка методу модулярного множення з суміщенням захищеної реалізації дискретного перетворення обробки декількох розрядів множника і групової редукції Монтгомері

Подальше прискорення обчислювальної реалізації операції модулярного множення багаторозрядних чисел може бути досягнуто за рахунок організації одночасного аналізу не одного, а групи з k бітів множника B. Таке рішення доволі часто застосовується в традиційній комп’ютерній арифметиці, як один із способів прискорення процесорної операції множення [21].

Основна ідея полягає в тому, що організується одночасна обробка відразу k розрядів множника. Відповідно, множник B розділяється на $d=n/k$ фрагментів

$b_q, b_{q-1}, \ldots, b_2, b_1$, $\forall i \in \{1,2,\ldots,d\}$: $0 \le b_i \le 2^k$-1. Модулярне множення організується у вигляді d циклів, в кожному з яких множиме А множиться на відповідний фрагмент множника В. До отриманого результату додається передобчислене значення модуля М на ціле число таке, щоб k молодших розрядів суми дорівнювали нулю. Після цього здійснюється зсув праворуч отриманої суми на k розрядів. Значення k визначається об'ємом пам'яті, що може бути виділений для зберігання таблиць передобчислень, а також кількістю операцій передобчислення.

Описана ідея може бути ілюстрована таким прикладом. Потрібно обчислити модулярний добуток 12·53 mod 51 = 24 з використанням редукції Монтгомері. Параметри модулярного множення: множине А=12, модуль М = 51_{10} = 110011_2, розрядність модуля n=6, параметр Монтгомері Q =2^n mod M = 2^6 mod 51 = 5. Множник В= 53_{10} = 110101_2. Розрядність фрагменту k=2, кількість фрагментів $d=n/k=6/2=3$. Фрагменти множника: $b_3=11_2=3$: $b_2 = 01_2=1$; $b_1=01_2=1$. Таблиця передобчислень: T[0]=0; T[1]= M= 51, T[2]= 2·M=102, T[3]=3·M =153.

Таблиця 2.3

Динаміка покрокової трансформації змінних модулярного множення

i	b_i	Множення	Корекція	Зсув
1	1	R=0 + 12·1 = 1100_2	R=R+T[0] = 12=1100	R = 11_2=3
2	1	R=3+12 = 15_{10} = 1111_2	R=R+T[3]=165_{10}=11000001_2	R=110000=48
3	3	R=48+ 3·12=84=1010100_2	R=R+T[0]=84=1010100_2	R=01111_2=15

Для отримання правильного результату потрібно множити отримане значення на Q = 5: R′=15·5 mod 51 = 24.

Оскільки кожний набір операцій, що має бути виконаний для реалізації запропонованої ідеї залежить від трьох компонентів: k -розрядної групи бітів множника В, k молодших розрядів множимого А і k молодших розрядів коду

поточного результату, то можна говорити, що загальна кількість бітів, комбінація яких визначає тип операції, яка реалізує додавання множимих і групову корекцію Монтгомері дорівнює $3 \cdot k$. Відповідно, кількість рядків V таблиці передобчислень може бути обчислений за наступною формулою:

$$V = 2^{3 \cdot k}. \quad (2.2)$$

В силу того, що в кожному рядку таблиці передобчислень зберігається *n*-розрядний код, то загальний об'єм W таблиці передобчислень обчислюється за формулою:

$$W = 2^{3 \cdot k} \cdot n. \quad (2.3)$$

Проте об'єм табличної пам'яті для зберігання таблиці передобчислення не є найбільш критичним обмеженням до збільшення кількості k [22] розрядів множника, обробка яких суміщується в часі. Зі збільшенням k швидко зростає кількість операцій передобчислення, які потрібно здійснити перед початком власне модулярного множення. В ході передобчислень потрібно обчислити суму двох компонентів – певної підмножини зсунутих кодів множимого A і певної підмножити зсунутіх кодів модуля M. Цілком очевидно, що кількість варіантів вибору кожної із підмножин дорівнює 2^k-1. Наприклад, при аналізі двох розрядів множника k=2 може додаватися один незсунутий код A, один зсунутий ліворуч код множимого A і відразу обидва: зсунутий і незсунутий коди A.

Таким чином, сумарна кількість N_p додавання для реалізації передобчислення становить:

$$N_p = (2^k - 1)^2. \quad (2.4)$$

Відповідно, сумарна кількість N_0 операцій додавання *n*–розрядних кодів, потрібних для реалізації модулярного множення складає:

$$N_0 = \frac{n}{k} + N_p = \frac{n}{k} + (2^k - 1)^2. \quad (2.5)$$

Похідна від сумарної кількості операцій додавання:

$$\frac{dN_0}{dk} = -\frac{n}{k^2} + 2^{k+1} \cdot (2^k - 1) \cdot ln\, 2. \quad (2.6)$$

Чисельне значення коефіцієнту β прискорення, яке досягається за рахунок одночасної обробки декількох розрядів множника і суміщення групової редукції Монтгомері може бути відображене формулою:

$$\beta = \frac{n}{N_0} = \frac{k}{1+\frac{k}{n}\cdot(2^k-1)^2} \qquad (2.7)$$

Очевидно, що зі зростанням кількості розрядів множника, обробка яких суміщується швидко зростає об'єм передобчислень. На рис.2.3 представлено графічна залежність коефіцієнту прискорення модулярного множення від розрядності чисел та кількості розрядів множника, обробка яких суміщується. З цих графіків видно, що оптимальним є варіант суміщеної обробки 3-х розрядів множника для розрядності 2048 і чотирьох розряді множника для розрядності 4096.

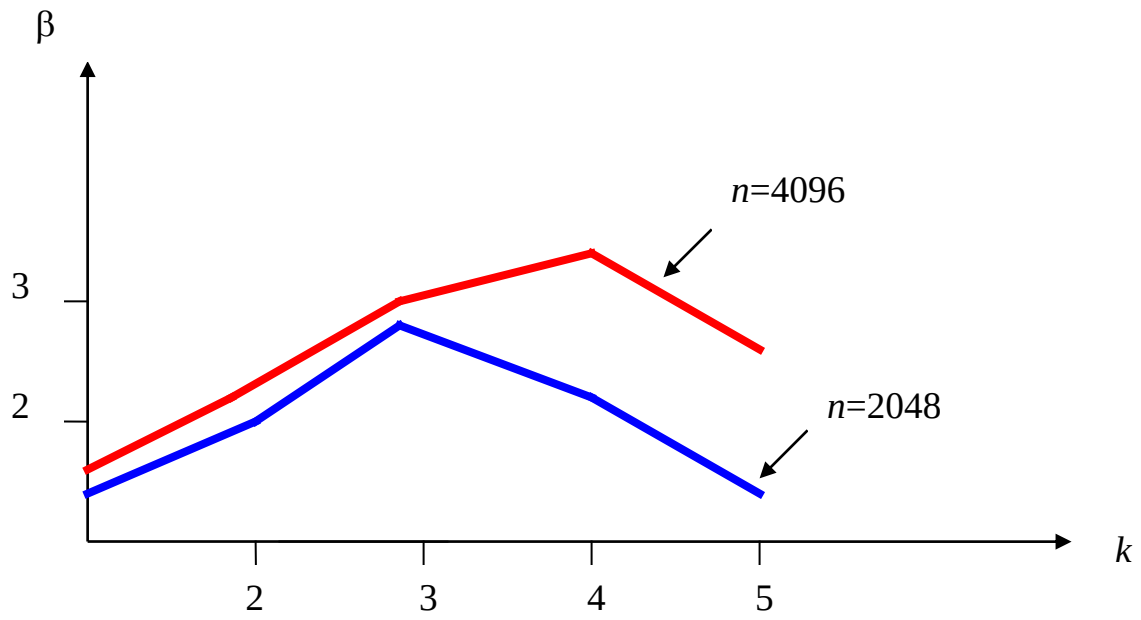


Рис.2.3 Графіки залежності коефіцієнта β прискорення модулярного множення від розрядності *n* чисел і кількості *k* розрядів множника, обробка яких суміщується

Розроблено та досліджено метод прискореного модулярного множення, який відрізняється тим, що операції додавання множимого і маски суміщуються з використанням передобчислень, що дозволяє прискорити виконання важливої

для криптографічних застосувань операції в 2-3 рази в залежності від кількості розрядів множника, які одночасно аналізуються.

2.3 Дослідження можливості виконання модулярного множення з редукцією Монтгомері на багатоядерних процесорах

Подальше збільшення швидкодії обчислювальної реалізації модулярного множення може бути досягнуто за рахунок розділення множника на секції, причому обробка кожної із секцій множника організується на окремому ядрі процесора.

Іншим варіантом організації розпаралелювання при виконанні модулярного множення є адитивний розклад множимого A [23]. Проте цей варіант не забезпечує помітного прискорення процедури модулярного множення.

Найбільшого ефекту в плані прискорення виконання операції модулярного множення може надати суміщення секційного множення та групової редукції. Використання секційного множення надасть можливість ефективно використовувати потужності процесорного ядра за рахунок застосування команди процесорного множення. переважна більшість сучасних процесорів мають спеціальні вбудовані засоби прискореного множення[24].

Головним резервом прискорення операції модулярного множення є розпаралелювання обчислювального процесу, тобто виконання операції на декількох ядрах.

Для прикладу розглянемо модулярне множення числа A = 1101012 = 5310 та числа B = 1010112 = 4310 при значенні модуля M=59 : A□B mod M = 43□53 mod 59 = 37.

Якщо множення виконується на 3-х ядрах, то множник B ділиться на три фрагменти однакової розрядності: $B_1=10_2$, $B_2=10_2$, $B_1=11_2$. Множення A на перший фрагмент B_1 зводиться до 5-ти тактів додавання нулів і одного, 6-го

такту, в якому до проміжного результату Y додається A: Y=Y+A = 0+53 = 53. Оскільки значення Y=53 непарне, то до нього додається модуль: Y=Y+M = 53 + 59 = 112. Далі, здійснюється зсув проміжного результату Y: Y/2 = 56. Таким чином, результат множення на перший фрагмент множника Y_1 = 56.

Множення числа A на другий фрагмент $B_2 = 10_2$ множника складається із 3-х тактів суми з нулем до коду Y_2 проміжного результату, 4-го такту, в якому до нульового значення Y_2 додається A: $Y_2 = Y_2 + A = 0+53 = 53$. Оскільки значення $Y_2 = 53$ непарне, то до нього додається модуль: $Y_2 = Y_2 + M = 53 + 59 = 112$. Далі, здійснюється зсув проміжного результату Y_2: $Y_2/2 = 56$. В 5-тому такті реалізується зсув проміжного результату Y_2 без попередньої корекції, в силу того, що Y_2 парне : $Y_2 = 56/2 = 28$. Аналогічні дії виконуються і в останньому, 6-му такті: $Y_2 = 28/2 = 14$. Таким чином, результат множення A на другий фрагмент множника Y_2 = 14. Наведені обчислення ведемо в таблицю 2.4.

Множення числа A на третій фрагмент $B_3 = 11_2$ множника виконується наступним чином. В першому такті до нульового значення Y_3 додається значення A: $Y_3 = Y_3 + A = 0+61 = 61$. Оскільки значення $Y_3 = 53$ непарне, то до нього додається модуль: $Y_3 = Y_3 + Z = 61 + 51 = 112$. Далі, здійснюється зсув проміжного результату Y_3: $Y_3/2 = 56$. В наступному, другому такті також відбувається додавання A до значення Y_3: $Y_3 = Y_3 + A = 52+61 = 113$. Оскільки значення $Y_3 = 113$ непарне, то до нього додається модуль: $Y_3 = Y_3 + Z = 113 + 51 = 164$. Далі, здійснюється зсув проміжного результату Y_3: $Y_3/2 = 164/2=82$. В 3-му такті реалізується зсув проміжного результату Y_3 без корекції в силу того, що молодший розряд Y_3 – парний: $Y_3 = Y_3/2 = 82/2=41$. Аналогічні дії виконуються і на 4-му такті: $Y_3 = Y_3/2 = 41/2 = 20$. В 5-му такті часткового множення здійснюється зсув проміжного результату Y_3 з його попередньою корекцією для того, щоб молодший біт дорівнював нулю: $Y_3 = Y_3 + Z = 21 + 51 = 72$. Далі, здійснюється зсув проміжного результату Y_3: $Y_3/2 = 72/2=36$. В останньому, 6-му такті здійснюється зсув проміжного результату Y_3 без корекції тому, що молодший розряд Y_3 – парний: $Y_3 = Y_3/2 = 36/2=18$. Таким

чином, результат множення A на третій фрагмент множника Y_3 = 18. Складається із 3-х тактів додавання нуля до коду Y_2 проміжного результату, 4-го такту, в якому до нульового значення Y_2 додається A: $Y_2 = Y_2 + A = 0+53 = 53$. Оскільки значення Y_2 =53 непарне, то до нього додається модуль: $Y_2 = Y_2+M = 53 + 59 = 112$. Далі, здійснюється зсув проміжного результату Y_2: $Y_2/2 = 56$. В 5-тому такті реалізується зсув проміжного результату Y_2 без попередньої корекції, в силу того, що Y_2 парне : $Y_2= 56/2 = 28$.

Таблиця 2.4

Динаміка змін часткових множників при їх обробці на 4-х ядрах

i	Операція	Значення змінних часткових модулярних множень											
		0			1			2			3		
		B_0	P_0	υ_0	B_1	P_1	υ_1	B_2	P_2	υ_2	B_3	P_3	υ_3
Вих.стан		257	0	0	257	0	0	17	0	0	272	0	0
0	P=P+A	257	2825	9	257	2825	9	17	2825	9	272	0	0
	P=P+T[υ]	257	27024	0	257	27024	0	17	27024	0	272	0	0
	B,P >>=4	16	1689	9	16	1689	9	1	1689	9	17	0	0
1	P=P+A	16	1689	9	16	1689	9	1	4514	2	17	2825	9
	P=P+T[ρ]	16	25888	0	16	25888	0	1	52912	0	17	27024	0
	B,P >>=4	1	1618	2	1	1618	2	0	3307	11	1	1689	9
2	P=P+A	1	4443	411	1	4443	33	0	3307	23	1	4514	10
	$P_0=P_0+T[1]$	1	21728	0									
	P1=P1+T[3]				1	49384	8						
	P2=P2+T[3]							0	48248	8			
	P3=P3+T[0]										1	4514	2
	P_0>>=4	1	1358	14									
	P_1>>=3				1	6173	13						
	P_2>>=2							0	12062	14			
	P_3>>=2										1	2257	1

Аналогічні дії виконуються і в останньому, 6-му такті: Y_2= 28/2 = 14. Таким чином, результат множення A на другий фрагмент множника Y_2 = 14.

Інший варіант полягає в тому, що множення множимого A здійснюється на фрагмент множника B, довжина якого дорівнює розрядності процесора *r*. Редукція за методом Монтгомері виконується після кожного циклу множення по фрагментам розрядністю *k*.

Цей варіант може бути ілюстрований наступним прикладом. Потрібно обчислити модулярний добуток 25·77 mod 173 = 22 з використанням редукції Монтгомері. Параметри модулярного множення: множене A=25, модуль M = 173_{10} = 10101101_2, розрядність модуля *n*=8, параметр Монтгомері Q =2^n mod M = 2^8 mod 173 = 83. Множник B= 77_{10} = 1001101_2. Розрядність r процесора дорівнює 3: *r* =3. Розрядність фрагменту *k*=2, Фрагменти множника: b_2=100_2=4: b_1 =1101_2=13. Таблиця передобчислень: T[0]=0; T[1]= 3·M = 519 =1000000111_2; T[2]= 2·M=346, T[3]=·M =173.

Алгоритм часткового модулярного множення числа A на секцію множника B, яка починається з *s*-го біту множника і містить у собі *d* бітів, зводиться до виконання наступної послідовності дій:

1. Лічильник *l* номеру поточного біту множника встановлюється в значення s: j=s; код проміжного значення результату Y обнуляється: Y=0.
2. Якщо значення *l*–го біту b_j множника B дорівнює одиниці, то до коду Y проміжного результату додається код множимого A: Y=Y+A, інакше, перехід на п. 3.
3. Якщо молодший розряд y_1 коду Y проміжного результату дорівнює одиниці: y_1=1, то до коду Y проміжного результату додається код можимого M: Y=Y+M, інакше перехід на п. 4.
4. Лічильник j номеру поточного біту множника збільшується на одиницю: l=l+1.

5. Якщо l<s+d, то повернення на повторне виконання п.2, інакше перехід на п.6.
6. Якщо l >n, то перехід на п.9.
7. Якщо молодший розряд y_1 коду Y проміжного результату дорівнює одиниці: y_1=1, то до коду Y проміжного результату додається код можимого M: Y=Y+M, інакше перехід на п. 8.
8. Лічильник l номеру поточного біту множника збільшується на одиницю: l=l+1. Перехід на повторне виконання п. 6.
9. Кінець алгоритму. Результат в Y

При використанні для обчислення модулярного добутку k процесорів, кількість бітів множника, які оброблюються на першому процесорі – d_1, на другому процесорі - d_2, на останньому k–тому процесорі - d_k При цьому, без втрати загалу, можна вважати, що молодші d_1 розряди множника оброблюються на першому процесорі, наступні d_3 оброблюються на другому процесорі, і так далі. Відповідно, старші d_k розряди множника оброблюються на k-тому процесорі. Зрозуміло, що $d_1+d_2+\ldots+d_k = m$.

Окремо може бути розроблена процедура модулярного піднесення до квадрату. При виконанні цієї операції існує можливість зменшення об'єму обчислень за рахунок виключення дублювання процесорних множень симетричних секцій множимого та множника.

При цьому кількість операцій процесорного множення зменшується з s^2 до $(s^2-s)/2$ [18].

Потрібно обчислити модулярний квадрат 37·87 mod 173 = 105 з використанням редукції Монтгомері. Параметри модулярного множення: множене A=25, модуль M = 173_{10} = 10101101_2, розрядність модуля n=8, параметр Монтгомері Q =2^n mod M = 2^8 mod 173 = 83. Множник B= 77_{10} = 1001101_2. Розрядність r процесора дорівнює 3: r =3. Розрядність фрагменту k=2,

Фрагменти множника: b_2=100_2=4: b_1 =1101_2=13. Таблиця передобчислень: T[0]=0; T[1]= 3·M = 519 =1000000111_2; T[2]= 2·M=346, T[3]=·M =173.

Максимальний ефект прискорення модулярного множення при його реалізації на k процесорах досягається при повному завантаженості кожного із процесорів, тобто за умови, що час виконання операцій на кожному із процесорів приблизно однаковий. Час t_j виконання операцій на j–тому процесорі, $j \in \{1,2,\ldots,k\}$ складається з двох частин: h_j раз виконується цикл множення з редукцією та ($d_{j+1}+d_{j+2}+\ldots+d_k$) на j-тий фрагмент множника та редукції результату і визначається за наступною формулою:

$$t_j = d_j \cdot t_{cm} + (n - \sum_{i=1}^{j} d_i) \cdot t_{cr}, \qquad (2.8)$$

де t_{cm} –середній час виконання циклу множення і редукції, t_{cr} – середній час виконання циклу редукції.

Ідею запропонованого методу ілюстровано наступним прикладом:

$$A = 1010\ 1110\ 1011_2 = 2795_{10}$$

$$B = 1010\ 0110\ 0111_2 = 2663_{10}$$

Значення істиного результату дорівнює:

$$R = A \cdot B\ mod\ M = 2795_{10} \cdot 2663_{10}\ mod\ 3117_{10} = 2806.$$

Множник B розділяється на три часткових множника B_1, B_2, B_3 розрядністю $n_1 = 2, n_2 = 4, n_3 = 6$:

$$B_1 = 0000\ 0000\ 0011_2 = 3_{10}$$

$$B_2 = 0000\ 0010\ 0100_2 = 36_{10}$$

$$B_3 = 1010\ 0100\ 0000_2 = 2624_{10}$$

Згідно з викладеним вище вираховуються результати часткових множень множеного A на виділені часткові множники:

$$R_1 = A \cdot B_1 mod\ M = 2795 \cdot 3\ mod\ 3117 = 2151$$

$$R_2 = A \cdot B_2 mod\ M = 2795 \cdot 36\ mod\ 3117 = 876$$

$$R_3 = A \cdot B_3 mod\ M = 2795 \cdot 2624\ mod\ 3117 = 2896$$

Повний результат R отримується шляхом додавання часткових добутків:

$$R = (R_1 + R_2 + R_3) mod\ M = (2151 + 876 + 2896)\ mod\ 3117 = 2806$$

Далі до R_3 додається значення T[7] = 3431 з таблиці передобчислень: R_3 = R_3 + T[9] 30879 = 33524_{10}= 1000 0010 1111 0100_2. У відповідності з п.8 здійснюється зсув на ω=2 розряди праворуч коду R_3= R_3>>2=33524/4=8381. Значення с збільшується на ω: с=с+ω = 7+2=9. Оскільки с= n-d_j виконується перехід на п.9 , в рамках якого отримане значення R_3 нормалізується, тобто віднього віднімається 2·M: 8381 – 2·M = 8381 – 6862 = 1519.

За п.7 в ρ заносяться значення двох (ω=2) молодших розрядів коду проміжного результату R_3 = 2645_{10}=1010 0101 01**01**$_2$ після чого ρ=1. Далі до R_3 додається значення T[2] [1] = 3431 з таблиці передобчислень: R_3 = R_3 + T[ω][ρ] = 2645 + 3431 = 6076_{10}= 0001 0111 1011 1100_2. У відповідності з п.8 здійснюється зсув на ω=2 розряди праворуч коду R_3= R_3>>2=6076/4=1519. Значення с збільшується на ω: с=с+ω = 7+2=9. Оскільки с= n-d_j виконується перехід на п.9 , в рамках якого отримане значення R_3 порівнюється з модулем: оскульки R_3=1519<M=3431, то третім частковим модулярним добутком є число 1519.

З огляду на доступні ресурси пам’яті для зберігання таблиці передобчислень кількість v циклів фази залишкової редукції, виконання яких суміщується у часі, може бути 16 і більше. За цих умов, кількість значащих розрядів множника B, що оброблюються на кожному із m процесорних ядер. Організація мультіпроцесорної обробки потоку з об’єднанням дозволяє реалізувати фазу залишкової редукції з группуванням, тобто шляхом об’єднання декілька суміжних циклів модулярної редукції Монтгомері в одному циклі. Запропонована технологія відрізняється тим. що розряди множника розносяться по частковим множникам розділяються на групи різного розміру. При цьому кількість розрядів в кожній із частин має тенденцію до збільшення: тобто в частковому множнику, що співвідноситься з молодшими розрядами основного множника оброблюється найменше число значащих розрядів. Це число збільшується при переході до наступного часткового

множника. І, нарешті, в складі часткового множника, в якому розміщено старші розряди множника B міститься найбільша їх кількість. Наприклад, при n=2048 і v=4, μ=3 оптимальна з точки зору однакової обчислювальної завантаженості процесів кількість розрядів множника становить: n_4 = 2048·0.305 = 680, n_3 = 553 n_2= 449 n_1=365. В загальному вигляді при секційному множенні s секцій множимого A на s секцій множника B розподілення по кількості числа потрібних циклів редукції підпорядковано біноміальному закону. Це означає, що один секційний добуток потребує максимальної кількості редукцій, s секційних добутків потребують s-1 циклів редукції, s·(s-1)/2 секційних добутків потребують s-2 циклів редукції.

Значення кількості значащих розрядів в старшому частковому множнику може бути визначення з міркувань того, що при обчисленні відповідного часткового добутку редукція зовсім не виконується. Тоді справедлива наступна формула:

$$n_m \cdot \frac{1-\gamma^m}{1-\gamma} = n. \qquad (2.9)$$

Зазначена в цій формулі сума являє собою сумою кінцевої геометричної прогресії. Відповідно, чисельні значення $n_1,n_2,\ldots,n_m$ визначаються на наступними формулами:

$$n_m = n \cdot \frac{1-\gamma}{1-\gamma^m}, \forall j \in \{1,,2,..,m\}: n_j = n \cdot \gamma^{m-j}. \qquad (2.10)$$

При груповій редукції 4-х розрядів будемо вважати, що R=M-1. Максимальне значення корегуючої добавки ξ·M дорівнює 15·M. Відповідно сума $(R+\xi\cdot M)_{max}$=16·M-1. Реально це число кратне 16 виходячи з базового критерію вибору ξ тому воно дорівнює 16·M-16. При зсуві на 4 розряди, тобто при діленні на 16 завжди гарантовано отримується число менше за модуль M.

Таким чином доведено, що запропонована організація розпаралелювання виконання модулярного множення багаторозрядних числах дозволяє прискорити виконання цієї важливої для криптографічних застосувань операції.

При цьому максмальна ефективність досягається при рівномірному навантаженні на ядра процесору. Разом з тим показано, що існує верхня границя доцільності розпаралелювання виконання операції модулярного множення, яка суттєвим чином залежить від розрядності чисел. так, для розрядностей 2048 верхня межа доцільності розпаралелючення дорівнює чотирьом.

Висновки до розділу 2

Проведені в рамках другого розділу бакалаврського диплому дослідження, направлені на розробку методів прискорення обчислювальної реалізації базової для широкого кола алгоритмів криптографічного захисту інформації операції модулярного множення багаторозрядних чисел дозволили отримати результати, що можуть бути сформульовані у вигляді наступних висновків:

1. Одним із основних резервів прискорення обчислювальної реалізації операції модулярного множення полягає в широкому застосуванні передобчислень та суміщенні мікрооперацій. Зокрема перспективним виглядає суміщення в часі виконання операцій множення та модулярної редукції, яке дозволяє не тільки зменшити час виконання цих операцій, але й запобігти роботі з числами, що значно перевищують розрядність модуля.
2. Теоретично обґрунтовано, запропоновано та досліджено метод прискореного модулярного множення, відмінність якого полягає в організації суміщення операцій додавання, що використовуються в процесі множення та редукції Монтгомері, що дозволяє прискорити обчислювальну реалізації цієї важливої для криптографічних застосувань операції. Здійснений теоретичний аналіз ефективності цього методу показав, а проведені експериментальні дослідження підтвердили, що використання запропонованого методу дозволяє прискорити обчислювальну реалізацію модулярного множення чисел, розрядність яких значно перевищує розрядність процесора приблизно на 30% в порівнянні з класичним алгоритмом модулярного множення з редукцією Монтгомері.
3. Розроблено та досліджено метод прискореного модулярного множення, який відрізняється тим, що організується одночасна обробка декілької розряді множника, причому операції додавання множимого і корегуючого коду редукції Монтгомері при цьому суміщаються з використанням

передобчислень, що дозволяє прискорити виконання важливої для криптографічних застосувань операції в 2-3 рази в залежності від кількості розрядів множника, які одночасно аналізуються.

4. Здійснено теоретичний аналіз запропонованого методу, результатом якого стало визначення оптимального, за критерієм прискорення виконання операції модулярного множення, кількості розрядів множника, які оброблюються одночасно і суміщаються з груповою редукцією Монтгомері.
5. Розроблені методи можуть бути ефективно використані для прискорення реалізації криптографічних протоколів з відкритим ключем на малопотужних термінальних мікроконтролерах систем комп'ютерного управління технологічними об'єктами.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНИХ ТА АПАРАТНИХ ЗАСОБІВ ПРИСКОРЕНОГО МОДУЛЯРНОГО МНОЖЕННЯ

Розроблені та теоретично досліджені в попередньому розділи підходи до прискорення базової для широкого кола криптографічних алгоритмів операції - модулярного множення орієнтовані на обчислювальні платформи, що не мають вбудованих засобів апаратної підтримки криптографічних перетворень – криптопроцесорів та криптоприскорювачів. Тобто мова йде, переважно, про малорозрядні вбудовані мікропроцесори, мікроконтролери та смарт-карти, що мають підтримувати діючі криптографічні протоколи. Крім програмної реалізації, запропоновані алгоритми модулярних мультиплікативних операцій можуть бути покладені в основу спеціалізованих апаратних засобів.

3.1 Розробка програми моделювання суміщення множення та редукції Монтгомері

З наукової точки зору, великий інтерес представляє експериментальне дослідження запропонованих модифікацій алгоритмів модулярного множення. Для моделювання було розроблено програму з використанням мови високого рівня C++. Програма дозволяє виконувати модулярне множення чисел зі змінною розрядністю у двох режимах: модулярне множення з використанням редукції Монтгомері за способом, запропонованим у розділі 2.1. Для перевірки ефективності запропонованого методу прискореного модулярного множення з суміщенням додавання множника та модулярної редукції за алгоритмом Монгомері з використанням передобчислень було розроблено дві програми.

Перша програма являється програмним представленням стандартного методу цього множення та перевіряє на правильність виконання цього методу.

Основою цієї програми являється функція int modularMutiplication (int C, int D, int M) , що виконує обчислення модулярного множення за стандартним

методом. Результатом виконання цієї функції являється повернення 1 або 0 в залежності від кінцевого значення обчислення модулярного множення.

Послідовність виконання функції зводиться до таких дій:

1. В допоміжну змінну *R* присвоюється 0 (*R=0*);
2. Знаходиться кількість розрядів числа *D* та записується в допоміжну змінну *n*;
3. Виконується проходження по циклу, що повторюється *n* разів. Цикл починається із обчислення суми змінної *R* на добуток числа *C* із *j* розрядом числа *D*
4. Перевіряється умова на непарність числа *R* , виконання якої зводиться до обчислення суми *R* та *M* (*R+=M*)
5. Виконання циклу закінчується зсувом на один розряд вправо числа (*R=R*/2)
6. Після циклу перевіряється умова чи більше значення змінної *R* за модуль M, виконання якої зводиться до знаходження різниці *R* та *M* в змінну *R*.
7. Знаходиться інверсія (*Q*) по модулю числа, де *i* – кількість розрядів коду числа *M*;
8. Перевіряється остача від ділення добутку *C*, *D*, та *Q* на модуль M. Якщо остача дорівнює 1, то функція повертає 1, якщо ні – то повертає 0.

Розроблена програма призначена для виконання прискореного модулярного множення запропонованого алгоритму за методом Монгомері.

Основою розробленої програми є функція acceleratedModularMultiplication (int C, int D, int M), яка виконує обчислення прискореного модулярного множення. Функції задаються числа C, D та M, що являються основними елементами цієї програми.

Всередині функції виконується така послідовність дій:

1. Задаються допоміжні зміні R (R=0), Q (Q=C+M) та знаходиться n-кількість розрядів D;

2. Виконується цикл n разів, в якому перевіряються умови, після яких виконуються ті чи інші обчислення з R , та виконується зсув коду числа R на один розряд вправо;
3. Знаходиться інверсія по модулю (P) числа , де n – кількість розрядів коду числа M;
4. Перевіряється остача від ділення добутку *C, D*, та *P* на модуль M. Якщо остача дорівнює 1, то функція повертає 1, якщо ні – то повертає 0.

Розробка програми показала, що передобчислення займають не більше 0.8% загального часу модулярного Множення і практично не впливають на час виконання цієї базової для криптографічних алгоритмів з відкритим ключем .

Властивість primes – це список простих чисел від 2 до 2099. Він є статичний масив цілих чисел.

Властивість maxLength показує максимально допустиму довжину якості data. За замовчуванням його значення дорівнює 70. Ця властивість є прихованою константою, тому зміна значення вчасно виконання програми неможлива.

Метод bitCount() повертає ціле число типу int, що відповідає найбільшому значенню позиції значущого біта числа.

Метод getBytes() повертає значення числа як масиву байт. Особливість даного методу полягає в тому, що найстаршому розряду числа відповідає нульовий байт масиву, що повертається. Такий формат чисел як масиву ускладнює роботу, оскільки чисел різної довжини необхідно виконувати попередні перетворення за погодженням значних розрядів. Щоб уникнути подібних проблем, були написані функції, що перетворюють представлення чисел у формат, зручний для виконання математичних операцій. Докладніше про ці функції йтиметься при розгляді класу MainForm.

В режимі модулярного множення користувач задає два постійних числа A і B і модуль M. Полем введення змінної A служить спеціальна комірка введення, що розбиває число, що вводиться на тріади цифр. Для введення

постійних чисел використовується аналогічний осередок, редагування якого контролюється натисканням відповідної кнопки на формі. Це тим, що, на відміну зміни змінних чисел, зміна постійних чисел тягне у себе перерахунок відповідної таблиці передобчислень.

Результат перевірки експериментально розроблених програм. При перевірці часу виконання цих програм було виявлено, що час виконання другої програми менша чим час виконання програми за стандартним методом модулярного множення. Оскільки час виконання обох програм при кожній компіляції різна, то їхню різницю було вирішено представити у відсотках. Для відсоткового представлення різниці часу виконання між першою та другою програмами було складено наступну формулу:

$$\frac{T_1 - T_2}{T_1} \times 100 ,$$

Де T_1 – час виконання першої програми, T_2 – час виконання другої програми. За цією формулою було виявлено, що експериментально розроблена програма за запропонованим методом прискореного модулярного множення зменшує час обчислення модулярного множення на 25 - 30% .

Таким чином, експериментальне дослідження виконане з використанням розроблених програмних засобів в цілому підтвердило отримані в другому розділі теоретичні оцінки прискорення модулярного множення за рахунок суміщення в часі операцій додавання множимого та модулярної редукції за технологією Монтгомері.

3.2 Розробка програм модулювання суміщення обробки групи розрядів множника і групової редукції Монтгомері.

Для практичного використання та експериментального дослідження запропонованого у другому розділі методу прискореного обчислення модулярного добутку з використанням суміщення обробки декількох розрядів

множника і групової редукції за технологією Монтгомері при постійному модулі було розроблено програму мовою Асемблера. Вибір як мови програмування асемблера обумовлений тим, що при цьому досягається найповніше використання ресурсів процесора, а також найточніша відповідність програми теоретичної моделі алгоритму. При використанні асемблера дозволяє створити ефективну програмну модель виконання мультиплікативних модулярних операцій на процесорах довільної розрядності, мікроконтролерів із заданою системою команд та смарт-картах.

У сегменті даних розташовані задані величини A,И,M, а також число M', позначене як mstrih. Визначено області комп'ютерної пам'яті для зберігання там допоміжних величин, які використовуються в процесі обчислень: u_j, help– байтів масив, в якому зберігається або результат від множення $u_j \cdot M$ або $x_j \cdot A$, і C – масив байтів, в якому зберігається кінцевий результат обчислень програми. Довжина масивів C і *help* має бути однаковою, иак як в програмі реалізується їх побайтове арифметичне додавання. В експериментальному зразку програми зарезервовано пам'ять на збереження 130 –ти байтів, на два байти більше, ніж довжина масивів A, B і M з урахуванням того, що при додаванні можуть виникати переноси з молодших байтів.

Для того, щоб забезпечити обробку чисел великої розрядності, яка значно перевищує розрядність процесора, саме число представляється у секцій, довжина яких співпадає з розрядністю процесора. В розробленій програмі є можливість для експериментування гнучко змінювати довжину секцій, починаючи від байту (сама такою є розрядність широкого класу термінальних мікроконтролерів), до 64-х розрядів. В лістингу, наведеному у додатках наведено приклад роботи з форматом подвійних слів (32 біта). Масив представлення числа доповнений двома подвійними словами для забезпечення коректності обчислень кратних модулів.

Дані в розробленій програмі представляються в такому вигляді:

1) mo dd 32 dup (0DDDDDDDDh),2 dup (0) – резервування пам'яті для зберігання модуля до яких додано два байти;
2) te dd 34 dup (0) – масив проміжного зберігання, в якому зберігаються лінійні комбінації модулі в – використовується для реалізації в програмі групової редукції Монтгомері:
3) x dd 2,0 – проміжна змінна X
4) y dd 1,0 – проміжна змінна У
5) ms dd 2 dup (0) – пам'ять для тимчасового зберігання арифметичного добутку двох чисел.

В розробленій програмі використовуються наступні макровизначення:

- sum macro adr1, adr2 – здійснює додавання двох масивів, що мають початкові адреси в пам'яті: adr1, adr2. Результат додавання двох масивів по заданих адресами розміщається в пам'яті за адресою adr1;
- zero macro adr1 – макровизначення, яка здійснює обнуління 34 компоненти масиву, початкова адреса якої задається змінною adr1;
- sum macro adr1,adr2,kol – додає два масива цілих чисел, результат поміщається при цьому розміщуються в пам'яті за начальною адресою adr1, довжина масиву чисел, які додаються задаються змінною kol, яка має формат подвіного слова (32 біти);
- mult macro *a*dr1,*a*dr2,*a*dr3,*a*dr4,kol – здійснює множення подвійного слова (x[j] або *a*[0]) на масив, початкова адреса якого задається параметром adr1, довжина масиву чисел, які множаться задаються змінною kol, яка має формат подвійного слова (32 біти), адреса куди розміщається добуток задається параметром adr4 макровизначення. Крім того, результуючий масив додається до до накопичувального масиву, початкова адреса якого визначається параметром *a*dr;
- zero macro adr1,kol – обнуляє масив подвійних слів, довжина якого визначається параметром kol, а початкова адреса – параметром adr1;

- shift macro adr1,kol – здійснює праворуч звув на одне слово масиву, початкова адреса якого задається параметром adr1, а довжина якого визначається параметром ko.

В розробленій програмі використовуються наступні модулі, процедури та функції:

Процедура MultAdd поєднує у собі множення масиву на байт, і додавання отриманого результату-масиву з масивом А. У процедурі виконуємо такі дії: обнуляем індексний регістр si, щоб використовувати його як лічильник байтів масиву; в регістр cx, який використовуватиметься як лічильник циклу, записуємо довжину масиву, щоб зробити операції над усіма його елементами. У молодший байт регістру dx записуємо байт, який будемо множити масив. У циклі молодшому байту регістру *a*x присвоюємо значення молодшого елемента (байта) масиву. Потім множимо байт на обраний елемент масиву. Встановлюємо прапорець C переносу у нуль. Далі допоміжному масиву привласнюємо отриманий результат, після чого у новому циклі складаємо цей масив з масивом А, таким чином накопичуючи в масиві А необхідну при реалізації алгоритму суму. Параметрами процедури є element – байт, який братиме участь у множенні; mas1 – масив, який братиме участь у множенні; mas2 – масив А, у якому накопичується часткова сума; temp(temporary) – допоміжний масив для зберігання результату множення, та додавання його з масивом А; rozr - Довжина масиву mas1.

Процедура ShiftL – процедура зсуву ліворуч. Вона дозволяє реалізувати розклад множника В на ступні двійки шляхом зсуву молодших розрядів, тобто. ліворуч. Цей зсув ми здійснюється як в рамках одного слова, так між словами масиву, в якому зберігається довге число. Параметрами процедури є тільки розрядність (rozr) масиву А, що зсувається, т.к. за алгоритмом нам необхідно зсувати лише цей масив, адреса якого задається не через параметри, а безпосередньо.

В основній програмі ми множимо нульовий елемент масиву B на a_j, складаємо отриманий результат з нульовим елементом масиву C і множимо отриманий результата на модуль M'. Молодший байт отриманого числа заноситься у змінну u_j.

Для використання програми необхідно в сегменті даних задати 128 байт числа A, B, що перемножуються, а також модуль M. Занесення цих чисел у відповідні масиви має здійснюватися починаючи з молодших байтів. Результат обчислень буде розміщено у пам'яті за адресою масиву C.

Допоміжна програма написана на C++ та призначена для обчислення табличних значень. У програмі створено тип all, що містить максимально можливу довжину результативного масиву байтів M_y. Модуль M може мати розрядність до 4096, у програмі він представлений як 130 байтний масив, з нумерацією від 0 до 127, але так само він має два додаткові байти, для коректного додавання з результативним масивом. Змінна *k* розміром байт задає розрядність процесора. У змінну mstrih записується результат роботи програми, саме M'. Також присутні змінні для проміжних обчислень: x, y, i, val.

У допоміжній програмі використано лише дві процедури. Процедура Add це процедура складання двох масивів. Вона замінює нам процедуру множення масиву модуля на у багаторазовим додаванням у разів. Параметри процедури це: масив приймач, масив джерело, та змінна кількість кроків, у даному випадку y. Процедура Clear – очищає масив завдовжки 130 елементів (байт). Щоб отримати залишок від розподілу на 2^{2n}, коли $n \leq 8$, достатньо аналізувати лише останнє слово всього числа або два останні елементи масиву. Тому для знаходження M' перевіряємо два останні байти на збіг з одиницею для продовження реалізації алгоритму знаходження M'. Для знаходження M' необхідно в тілі програми задати поелементно 128 байт числа модуля, починаючи з молодших байтів, у шістнадцятковій системі.

3.3 Розробка структур апаратних засобів прискореного модулярного множення

Для значної частини застосувань, пов'язаних із завданнями управління в реальному часі, єдино прийнятним варіантом швидкої реалізації протоколів захисту інформації на основі операцій модулярної арифметики є використання спеціалізованих апаратних засобів. Застосування апаратних засобів дозволяє досягти теоретично можливої межі продуктивності обчислень за рахунок можливості їх розпаралелювання.

При апаратної реалізації операцій запропонованих модифікацій модулярного множення слід вивчити питання можливості розпаралелювання базових алгоритмів. При цьому треба розглядати два рівні розпаралелювання:

1. Розпаралелювання операцій у рамках одного циклу обробки секції множника.
2. Розпаралелювання обробки кількох секцій множника.

Класичний алгоритм [25] модулярного множення принципово дозволяє реалізувати розпаралелювання на обох рівнях. Алгоритм модулярного множення з застосуванням редукції Монтгомері не дозволяє організувати одночасну обробку кількох секцій множника, оскільки величина поправки, що використовується на кожному j–тому циклі обчислень, $j \in \{0,2,\ldots,k\text{-}1\}$, що забезпечує можливість виконання рекурсії шляхом зсуву праворуч залежить від проміжного результату модулярного множення отриманого на попередньому, (j-1)-му циклі обчислення модулярного добутку.

Аналіз модифікації модулярного множення, запропонованої в розділі 2.3 показує, що з його апаратної реалізації розпаралелювання обчислень може бути досягнуто на обох зазначених вище рівнях.

При розпаралелювання операцій, що виконуються в рамках одного циклу обробки однієї *k*-розрядної секції множника B необхідно використовувати *k*-розрядні схеми множення. Узагальнена схема блоку модулярного множення представлена на рис.3.1.

Зсувний регістр множимого A складається з s + 1 *k*-розрядних секцій a_0, a_1,...,a_s. При початковому завантаженні множимого заповнюються тільки молодші секції s, а всі розряди старшої секції a_s зсувного регістру встановлюються в нуль. Модулярна редукція виконується з використанням табличного перетворювача (Т), що являє собою швидкодіючу пам'ять, організовану у вигляді 2^k *n*-розрядних слів. Враховуючи відносно великий час спрацьовування такої пам'яті, у запропонованій структурі використовуються два такі перетворювачі: для редукції множимого, що зсувається, і редукції проміжного результату перед його зсувом праворуч. Ці два вузли працюють паралельно. Частково функції редукції реалізуються з використанням модулярного суматора (МС), який виконує додавання двох *n*-розрядних чисел, порівнює отриманий результат з модулем і, якщо сума перевищує модуль М, то здійснює одноразове віднімання модуля від проміжної суми. Для безпосереднього множення секцій множеного на код поточної секції множника A використовується група *s* *k*-розрядних комбінаційних помножувачів (КП), 2·*k*-розрядні виходи яких зі зсувом підключені до відповідних входів модулярного суматора (МС).

Отриманий у результаті додавання всіх компонентів результат редукується кожному циклі з допомогою табличного перетворювача і модулярного суматора, причому редукція множеного і проміжного результату суміщуються у часі. Якщо позначити через t_Y - час виконання множення на комбінаційному помножувачі, $t_Д$ – час додавання часткових добутків на суматорі, $t_Т$ - час читання даних з табличної пам'яті передобчислень, а через $t_{МС}$-

час формування результату модулярним суматором, то загальний час T_0 модулярного множення обчислюється за наступною формулою:

$$T_0 = \frac{n}{k} \cdot (t_Y + t_Д + t_T + t_{MC}). \quad (3.1)$$

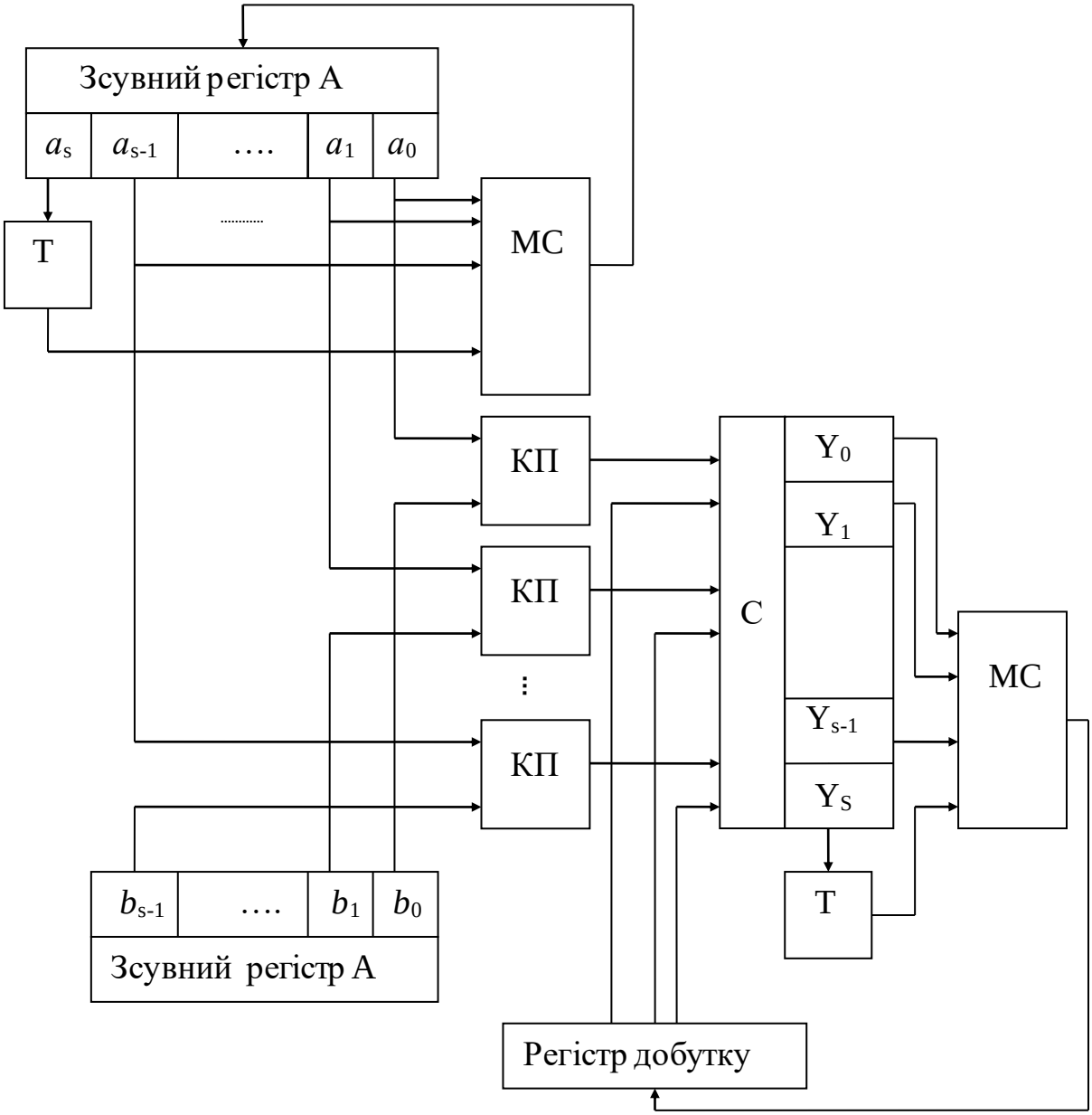


Рис.3.1 Структура апаратних засобів для прискореного модулярного множення

Формула (3.1) не враховує часу редукції множимого при його зсуві праворуч в силу того, що редукція множимого здійснюється одночасно з обчисленням суми часткових добутків і час її реалізації менший за час обчислення суми часткових добутків.

При практичному використанні апаратних засобів прискореного модулярного множення базовою проблемою є вибір оптимального значення розрядності секції. Проведений аналіз можливостей реалізації запропонованих апаратних засобів на програмованих логічних матрицях показав, що найбільш ефективна реалізація забезпечується при значенні розрядності секції k=16.

Істотно більша швидкість обчислення модулярного добутку на основі запропонованого в попередньому розділі методу може бути досягнута при реалізації одночасної обробки всіх секцій множника B.

Оскільки в такому варіанті апаратної реалізації модулярного множення за умови сталого значення модуля одночасно виконується множення всіх секцій множника B на всі секції множного A, то загальна кількість k-розрядних комбінаційних помножувачів (КП) дорівнює s^2. Для врахування при підсумовуванні отриманих при цьому часткових добутків їх ваг, можна використовувати відповідні операції зсуву або реалізувати це за рахунок відповідних комутацій.

Наприклад, при обчисленні добутку молодшої секції a_0 множимого A на старшу секцію b_S множника B на комбінаційному помножувачі необхідно обчислити $b_S \cdot a_0 \cdot 2^{k\cdot(s-1)} \bmod M$. Оскільки $a_0 \cdot 2^{k\cdot(s-1)}$ може бути більше модуля M, то обчислення $a_0 \cdot 2^{k\cdot(s-1)} \bmod M$ виконується з використанням табличного перетворювача T. При обчисленні добутку старшої секції b_S множника B на старшу секцію a_s множимого A, з урахуванням їх ваг обчислюється: $a_s \cdot b_s \cdot 2^{2\cdot k\cdot(s-1)}$ з використанням табличного перетворювача T.

Відповідно, для різних секцій множини використовується різне число табличних перетворювачів. Їх загальна кількість становить $0.5 \cdot k^2 \cdot (k-1)$. Крім помножувачів та табличних перетворювачів значна частина апаратної складності приходиться суматор. Схему суматора, виходячи з отримання можливо більшої швидкодії, доцільно будувати у вигляді $2 \cdot \log_2 s$ шарів, кожен з яких містить двохвходові n-розрядні схеми комбінаційного додавання.

Оскільки результат на виході суматора не перевищує 2^{n+k}, для його модулярної редукції достатньо використовувати один табличний перетворювач Т і модулярний суматор. Загальний час Т′ виконання модулярного добутку з використанням описаних апаратних засобів, визначається наступною формулою:

$$T' = t_Y + t_Д + t_T + t_{МС}. \quad (3.2)$$

Порівняння формули (3.2) з формулою (3.1) свідчить про те, що час модулярного множення зменшується приблизно на порядок. Очевидним недоліком описаного схемного рішення є велика складність залучених для реалізації модулярного множення апаратних засобів.

Висновки до розділу 3

В результаті виконання комплексу теоретичних і експериментальних досліджень, направлених на створення ефективних програмних і апаратних засобів прискореного виконання модулярного добутку згідно з надбаннями, зробленими в другому розділі, можна зробити наступні висновки:

1. Розроблені програмні засоби, що на практиці реалізують запропоновані в другому розділі методи прискореного обчислення модулярного добутку чисел, розрядність яких значно перевищує розрядність процесора.
2. Експериментальне дослідження виконане з використанням розроблених програмних засобів в цілому підтвердило отримані в другому розділі теоретичні оцінки прискорення модулярного множення за рахунок суміщення в часі операцій додавання множимого та модулярної редукції за технологією Монтгомері.
3. Виконані з застосуванням розроблених програмних засобів експериментальні дослідження з організації при виконанні модулярного множення обробки відразу декількох розрядів множника підтвердили отримані теоретичні оцінки, а також довели наявність технологічних обмежень на кількість розрядів множника, що можуть оброблюватися одночасно і суміщатися при цьому з груповою модулярною редукцією Монтгомері.
4. Розроблені структури апаратних засобів для реалізації запропонованих в другому розділі проєкту методів прискореного модулярного множення з суміщенням множення та редукції. Аналіз розроблених структур показав, що вони дозволяють на порядки збільшити швидкість виконання модулярного множення в порівнянні з програмною реалізацією. Розглянуто дві структури, що відрізняються різним ступенем реалізованого в них паралелізму обчислень. Найбільш доцільним варіантом практичної реалізації

запропонованих схемних рішень є використання програмованих логічних матриць.

5. Розроблені програмні засоби прискореного модулярного множення можуть бути, крім цілей експериментальних досліджень, використані прямо для підвищення продуктивності реалізації криптографічних протоколів захисту інформації, зокрема таких як DSS та ISO-979.

ВИСНОВКИ

Дослідження, які було проведено для дипломного проєкту направлені на розробку методу прискореного модулярного множення для систем криптографічного захисту даних приводять до наступних висновків:

1. Проведений аналіз показав, що значна частина сучасного арсеналу криптографічних механізмів захисту інформації використовує в якості базових операцій модулярне множення та модулярне експоненціювання над числами, довжина яких на порядок перевищує розрядність процесора. Найбільшою проблемою ефективності цих механізмів є повільність комп'ютерної реалізації, зумовленою значною обчислювальною складністю вказаних вище мультиплікативних операцій модулярної арифметики. Відповідно, ефективність цих криптографічних механізмів може бути підвищена за рахунок прискорення виконання мультиплікативних операцій модулярної арифметики.
2. Проведений аналіз особливостей практичного використання протоколів, основаних на крипто¬графічних алгоритмах несиметричної криптографії, зокрема, алгоритмів шифрування RSA, El-Gamal, алго¬рит¬му цифрового підпису –DSS показує, а також алгоритмів криптографічно строгої ідентифікації FFSIS, Гіллоу-Квіскватера та Шнорра свідчить, що їх ключі змінюються відносно рідко, оскільки їх відкрита частина ідентифікує абонентів комп’ютерної мережі. Відповідно, модуль M, що є частиною відкритого ключа абонентів, можна розгляди як постійне число і зменшити, за рахунок цього, обчислювальну складність мульти-плікативних операцій модулярної арифметики.
3. Проведений аналіз існуючих алгоритмів обчислення базової операції механізмів захисту інформації з відкритим ключем – модулярного експоненціювання показав, що вони мають строго послідовний характер і не можуть бути розпаралелені: кардинальне прискорення модулярного

експоненціювання може бути здійснене лише на рівні операції модулярного множення.

4. Аналіз процедур секційного множення чисел великої розрядності, що складаються з s фрагментів показав, що кількість процесорних множень за рахунок використання спеціальних методів може бути зменшена з s2 до рівня su, де u стале число, що залежить від методу.
5. Аналітичний огляд існуючих технологій модулярної редукції показав, що найбільший потенціал для подальшого прискорення модулярного множення надає технологія Монтгомері, яка дозволяє сумістити в часі виконання множення та віднаходження залишку.
6. Одним із основних резервів прискорення обчислювальної реалізації операції модулярного множення полягає в широкому застосуванні передобчислень та суміщенні мікрооперацій. Зокрема перспективним виглядає суміщення в часі виконання операцій множення та модулярної редукції, яке дозволяє не тільки зменшити час виконання цих операцій, але й запобігти роботі з числами, що значно перевищують розрядність модуля.
7. Теоретично обґрунтовано, запропоновано та досліджено метод прискореного модулярного множення, відмінність якого полягає в організації суміщення операцій додавання, що використовуються в процесі множення та редукції Монтгомері, що дозволяє прискорити обчислювальну реалізації цієї важливої для криптографічних застосувань операції. Здійснений теоретичний аналіз ефективності цього методу показав, а проведені експериментальні дослідження підтвердили, що використання запропонованого методу дозволяє прискорити обчислювальну реалізацію модулярного множення чисел, розрядність яких значно перевищує розрядність процесора приблизно на 30% в порівнянні з класичним алгоритмом модулярного множення з редукцією Монтгомері.
8. Розроблено та досліджено метод прискореного модулярного множення, який відрізняється тим, що організується одночасна обробка декілької

розряді множника, причому операції додавання множимого і корегуючого коду редукції Монтгомері при цьому суміщаються з використанням передобчислень, що дозволяє прискорити виконання важливої для криптографічних застосувань операції в 2-3 рази в залежності від кількості розрядів множника, які одночасно аналізуються.

9. Здійснено теоретичний аналіз запропонованого методу, результатом якого стало визначення оптимального, за критерієм прискорення виконання операції модулярного множення, кількості розрядів множника, які оброблюються одночасно і суміщаються з груповою редукцією Монтгомері.
10. Розроблені методи можуть бути ефективно використані для прискорення реалізації криптографічних протоколів з відкритим ключем на малопотужних термінальних мікроконтролерах систем комп'ютерного управління технологічними об'єктами.
11. Розроблені програмні засоби, що на практиці реалізують запропоновані в другому розділі методи прискореного обчислення модулярного добутку чисел, розрядність яких значно перевищує розрядність процесора.
12. Експериментальне дослідження виконане з використанням розроблених програмних засобів в цілому підтвердило отримані в другому розділі теоретичні оцінки прискорення модулярного множення за рахунок суміщення в часі операцій додавання множимого та модулярної редукції за технологією Монтгомері.
13. Виконані з застосуванням розроблених програмних засобів експериментальні дослідження з організації при виконанні модулярного множення обробки відразу декількох розрядів множника підтвердили отримані теоретичні оцінки, а також довели наявність технологічних обмежень на кількість розрядів множника, що можуть оброблюватися одночасно і суміщатися при цьому з груповою модулярною редукцією Монтгомері.

14. Розроблені структури апаратних засобів для реалізації запропонованих в другому розділі проєкту методів прискореного модулярного множення з суміщенням множення та редукції. Аналіз розроблених структур показав, що вони дозволяють на порядки збільшити швидкість виконання модулярного множення в порівнянні з програмною реалізацією. Розглянуто дві структури, що відрізняються різним ступенем реалізованого в них паралелізму обчислень. Найбільш доцільним варіантом практичної реалізації запропонованих схемних рішень є використання програмованих логічних матриць.
15. Розроблені програмні засоби прискореного модулярного множення можуть бути, крім цілей експериментальних досліджень, використані прямо для підвищення продуктивності реалізації криптографічних протоколів захисту інформації, зокрема таких як DSS та ISO-979.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 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.
2. 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.
3. Кот О.С. Організація прискореного експоненціювання на полях Галуа з використанням редукції Монтгомері / О.С.Кот, О.П. Марковський // Альманах науки.- 2020.- № 3 (36).- С.34-37.
4. Thangaval M. Improved secure rsacryptosystem (ISRSAC) for data confidentiality in cloud / M. Thangaval, P.Varalakshmi // International Journal of Information Systems and Change Managerment,- 2017.- Vol.9,- No.4, - P.46-53.
5. Калмиков І.А. Розробка методу нелінійного шифрування інформації з використанням операції піднесення до степеня для кінцевого поля Галуа / І.А. Калмиков, Е.С. Степанова, К.Т. Тинчеров// Сучасні наукомісткі технології. - 2019.- № 9. - С.84—89.
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, J.J. Quisquater // Cryptographic Hardware and Embedded System- CHES’2000. LNCS-1965, Springer-Verlag. — 2000.- P. 293-301.
8. 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.

9. Elfard S. Justification of Montgomery Modular Reductions / S. Elfard //Advanced Computing.- 2012. - № 11. – P.41-45.
10. Анисимов А.В. Быстрое прямое вычисление модулярной редукции // Кибернетика и системный анализ.-1999.-№ 4.-С.3-12.
11. Che Wun Chion. Parallel modular multiplication with table look-up. / Che Wun Chion, Ted C.Yang // International Journal of Computer Mathematics.-1998.- Vol.69.-Issue 1-2.- P.22-23.
12. Zuras D., More on squaring and multiplying larges integers, IEEE Transactions on Computers,-1994.- Vol. 43, - № 8,- P. 899–908.
13. Березовский А.И. О тестировании быстродействия алгоритмов и программ вычисления основных операций ассиметричной криптографии/ Березовский А.И., Задирака В.К., Шевчук Л.Б. //Кибернетика и системный анализ №5, 1999. – С. 59-66.
14. 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.
15. Buhrow B. Parallel modular multiplication using 512-bit advanced vector instructions: RSA fault-injection countermeasure via interleaved parallel multiplication / Buhrow B., Gilbert B., Haider C. // Journal of Cryptographic Engineering.-2021.- № 2.- P.46-53.
16. Karatsuba, A.Multiplication of many-digital numbers by automatic computers. Proc. USSR Acad. Sci. =1962.- Vol.145,- P. 293–294.
17. Анісімов А.В. Алгоритмічна теорія великих чисел / А.В. Анісімов // К.: Академперіодика. —2001. — С.153.
18. 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.
19. 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.

20. Laszlo Hars. Long Modular multiplication for Cryptographic Applications.//Cryptographic Hardware and Embedded System- CHES’2004. LNCS-3156, Springer-Verlag, - 2004. - P.45-61.
21. Walter C.D. Faster Modular Multiplication by Operand Scaling / C.D. Walter // Proceeding of Advances in Cryptology-CRYPTO-91, LNCS-576, Springer-Verlag. – 1992. – P. 313-323.
22. 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.
23. Самофалов К.Г. Эффективная реализация мультипликативных операций модулярной арифметики в системах защиты информации/ 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.
24. Харин Ю.С. Математические основы криптологии./ Ю.С.Харин, В.И. Берник, Г.В. Матвеев//Минск.: Изд-во БГУ —1999. — 319 с.
25. Montgomery P.L. Modular multiplication without trial division / P. L. Montgomery // Mathematics of Computation, - 1985.- Vol.44. - № 4.- P.519-523.

.

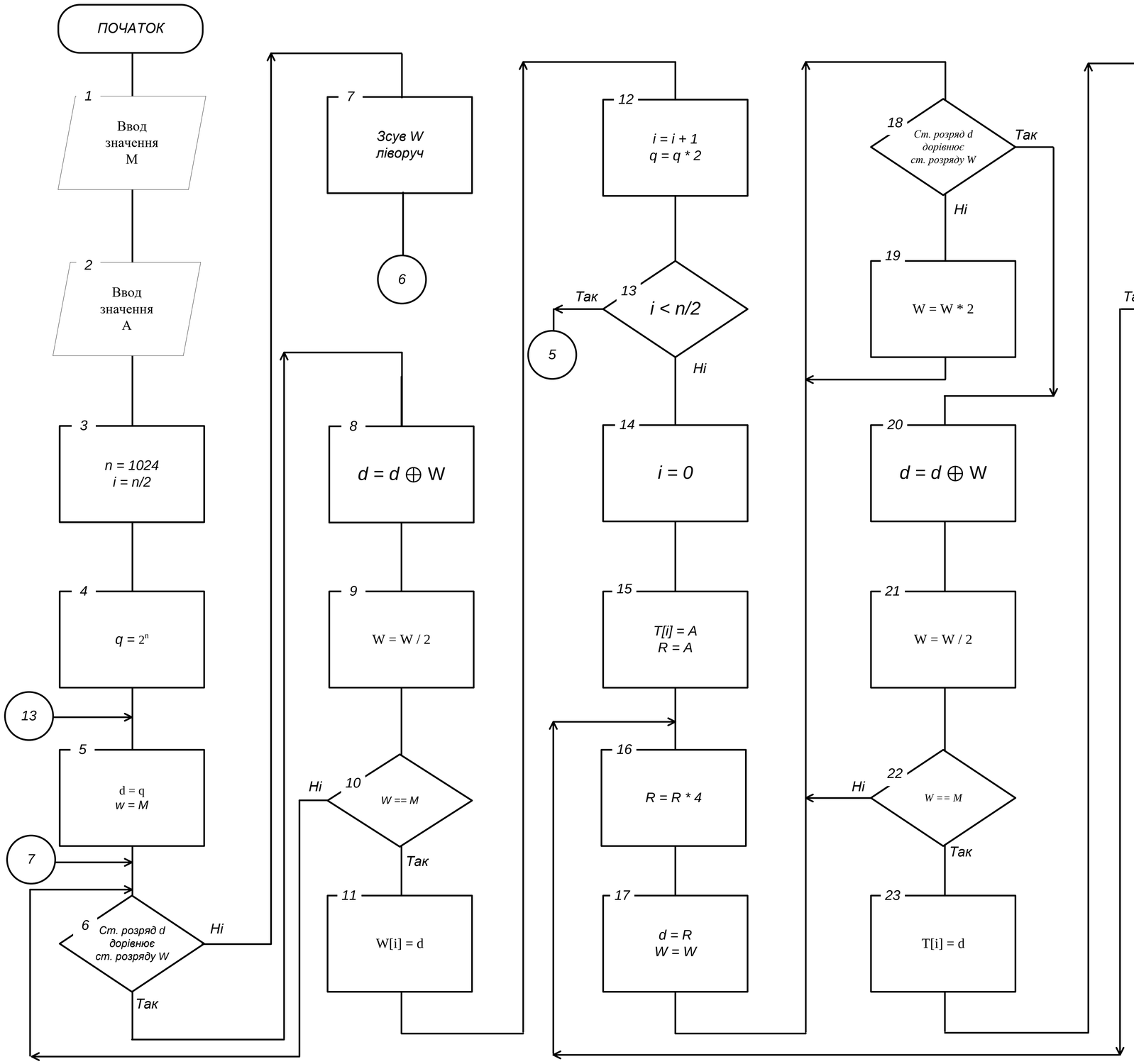
ДОДАТОК 1

Метод прискореного модулярного множення для систем криптографічного захисту даних

Блок-схема алгоритму процедури формування таблиць передобчислень

Аркушів 1

Київ – 2022 р.



					ІАЛЦ.468243.004.Д1	
Зм	Лист	№докум.	Підп.	Дата	Процедура формування таблиць передобчислень Схема алгоритму	Лит. Д / Маса / Масштаб
Розроб.		Кабір Л. А.				Лист 1 / Листів 1
Перев.		Марковський				
Т.контр.						
Н.контр.		Сімоненко В.П			НТУУ "КПІ", ФІОТ, гр. ІП-84	
Затв.		Стіренко С.Г..				

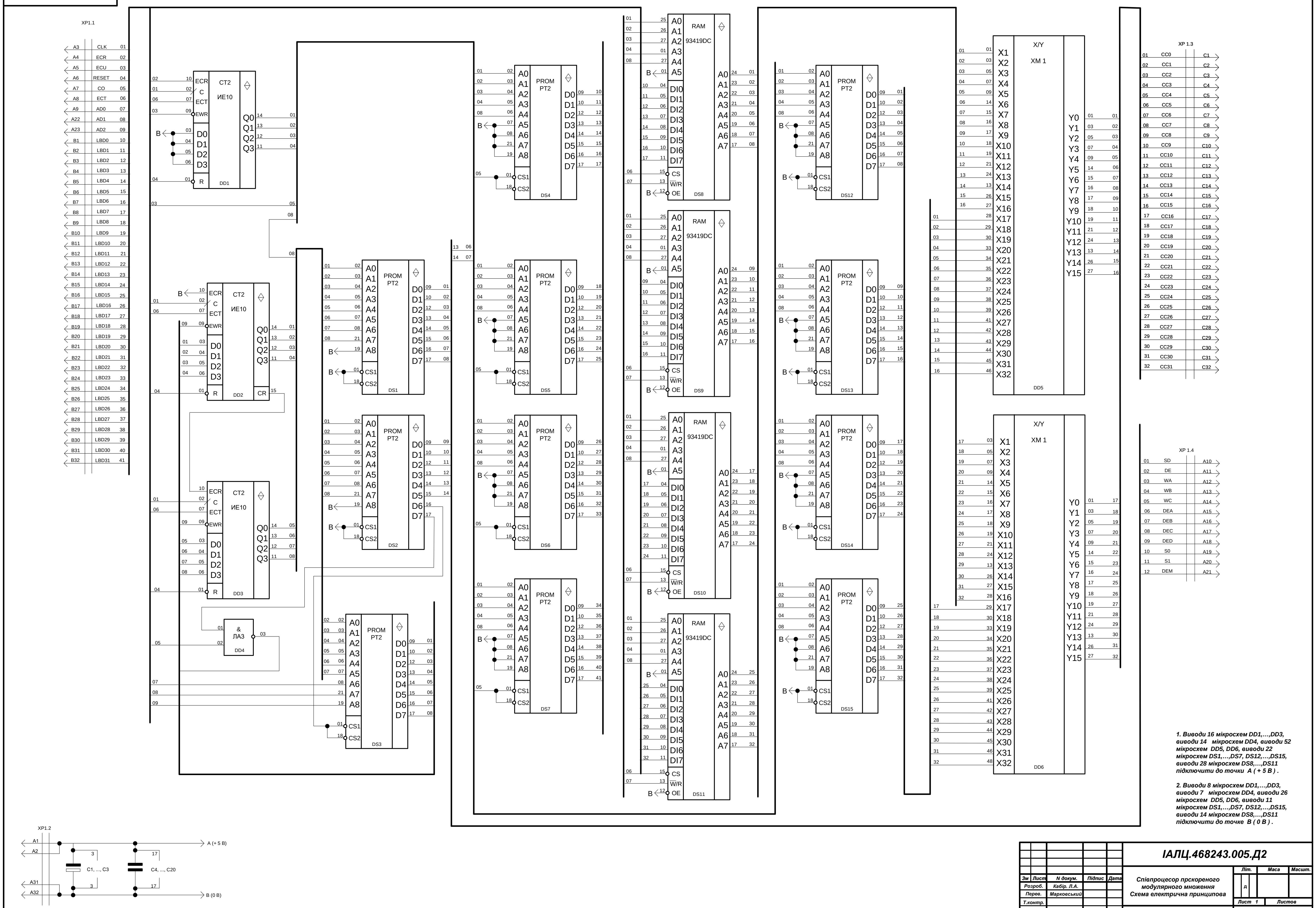
ДОДАТОК 2

Метод прискореного модулярного множення для систем криптографічного захисту даних

Схема електрична принципова співпроцесору прискореного модулярного множення

Аркушів 1

Київ – 2022 р.



1. Виводи 16 мікросхем DD1,…,DD3, виводи 14 мікросхем DD4, виводи 52 мікросхем DD5, DD6, виводи 22 мікросхем DS1,…,DS7, DS12,…,DS15, виводи 28 мікросхем DS8,…,DS11 підключити до точки A (+ 5 B).

2. Виводи 8 мікросхем DD1,…,DD3, виводи 7 мікросхем DD4, виводи 26 мікросхем DD5, DD6, виводи 11 мікросхем DS1,…,DS7, DS12,…,DS15, виводи 14 мікросхем DS8,…,DS11 підключити до точки B (0 B).

ІАЛЦ.468243.005.Д2

Зм	Лист	N докум.	Підпис	Дата
Розроб.		Кабір Л.А.		
Перев.		Марковський		
Т.контр.				
Н.контр.		Сімоненко		
Затверд.		Стіренко		

Співпроцесор прскореного модулярного множення
Схема електрична принципова

Літ.	Маса	Масшт.
Д		

Лист 1 Листов

НТУУ "КПІ", ФІОТ, гр. ІП-84

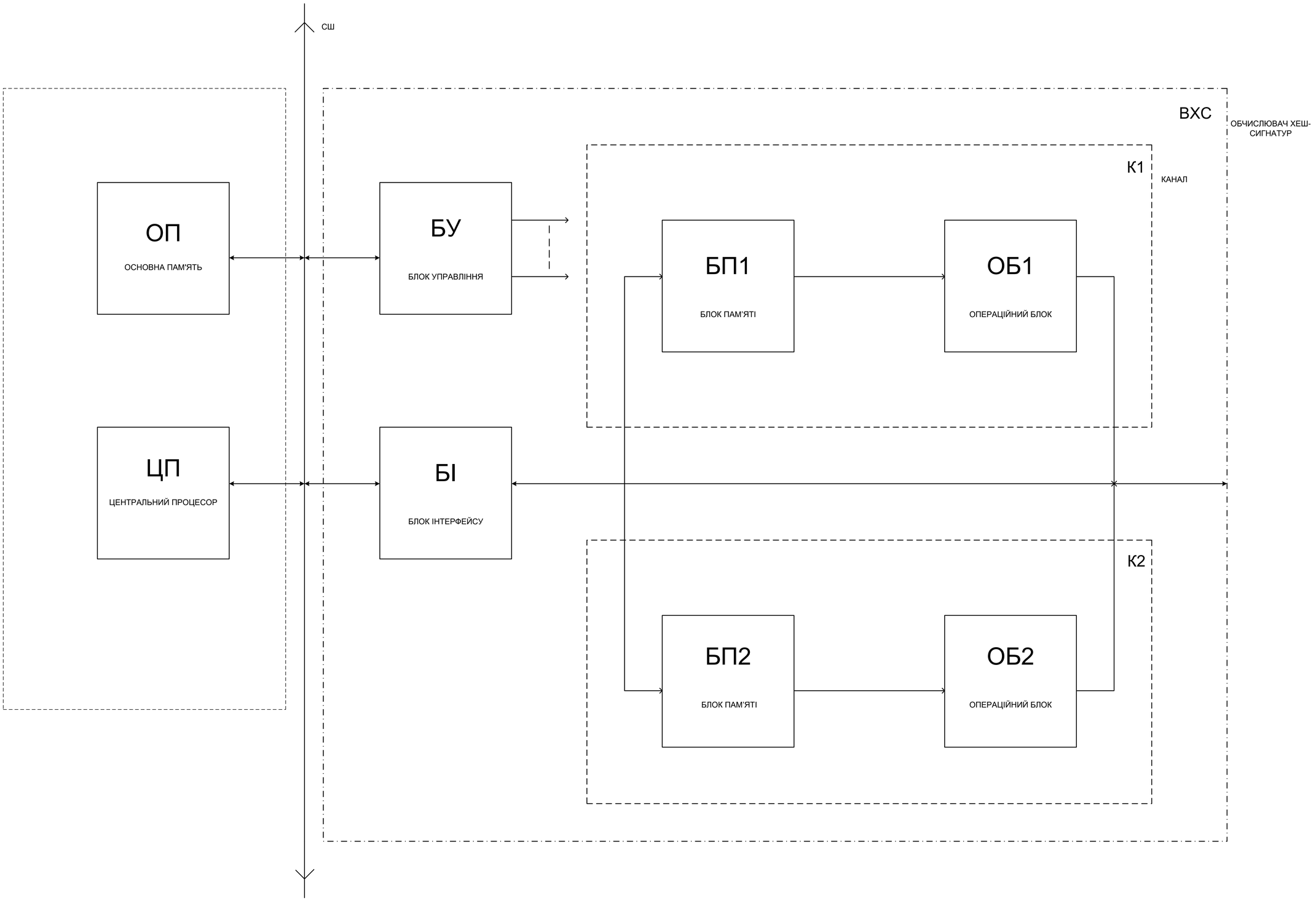
ДОДАТОК З

Метод прискореного модулярного множення для систем криптографічного захисту даних

Схема електрична структурна співпроцесору прискореного модулярного множення

Аркушів 1

Київ – 2022 р.



Зм	Арк	№докум.	Підпис	Дата	ІАЛЦ.468243.006.Д3		
Розроб.		Кабір Л. А.			Співпроцесор прскореного модулярного множення Схема електрична структурна	Лит.	Маса	Масштаб
Перев.		Марковський О.П.				Д		
Т.контр.						Лист 1	Листів 1	
Н.контр.		Сімоненко В.П				НТУУ "КПІ", ФІОТ, гр. ІП-84		
Затв.		Стіренко С.Г..						

ДОДАТОК 4

Метод прискореного модулярного множення для систем криптографічного захисту даних

Лістинг програми для реалізації прискореного модулярного множення для систем криптографічного захисту даних

Аркушів 11

Київ – 2022 р.

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <time.h>
using namespace std;

int acceleratedModularMultiplication(int , int , int );
int main() {
    srand(time(NULL));
    int Start = clock();
for(int i =0; i <= 1000; i++)
{
    int A = rand()%19;
    cout<<"\n A"<<A;
    int B = rand()%19;
    cout<<"\n B"<<B;
    int c = acceleratedModularMultiplication(A, B, 19);
    cout<<"\n c="<<c;
}
 int End = clock();
 cout<<"\n Start "<<Start<<" End "<<End;
    int d;
    cin>>d;
}

int acceleratedModularMultiplication(int C, int D, int M) {
    int R=0;
    int Q = C + M;
    int k = D;
```

```
int n = 0;
while (k!=0) {
    n++;
    k/=2;
}

for(int j = 0; j < n; j++) {
      if((D>>j)%2 == 0 && R&1 == 1)
            R += M;
      else
        if((D>>j)%2 == 1)
            if( (R&1 == 0&& C&1 == 0) || (R&1 == 1&& C&1 == 1))
              R += C;
            else
              R+= Q;
      R /= 2;
}
if(R>=M)
      R-=M;
 k = M;
n = 0;
while (k!=0) {
    n++;
    k/=2;
}
int P = pow(2,n);
int j = -1;
for(int i = 0; i <= M; i++)
      if((P*i)%M == 1)
```

```
                j=i;
    if((C*D*j)%M ==R)
            return 1;
    return 0;
}

.386
assume cs:code,ds:data

MultAdd macro element,mas1,mas2,temp,rozr
local lab,lab1
    xor si,si
    xor dx,dx
    mov cx,rozr
    mov dl,element
lab: mov al,mas1[si]
    mul dl          ;al*dl=ax  al_ah=xjY/al_ah=ujM
    clc
    adc temp[si],al
    adc temp[si+1],ah
    push cx
    push si
    mov cx,rozr+2
    xor si,si
    lab1: mov al,temp[si]
        mov temp[si],0
        adc mas2[si],al
        inc si
        loop lab1
```

```
    pop si
    pop cx
    inc si
    loop lab
endm

ShiftL macro rozr
local lab
    xor si,si
    mov cx,rozr      ;length-1
lab: mov al,a[si+1]
    mov a[si],al
    inc si
    loop lab
    mov a[si],0
endm

data segment use16
x db 128 dup(0aah) ;4 dup(0aah);0abh,0c5h,126 dup(0)
y db 128 dup(0bbh) ;4 dup(0bbh);37h,8ah,126 dup(0)
m db 128 dup(0ddh) ;4 dup(0ddh) ;4dh,0e9h,126 dup(0)
mstrih db 8bh ;7bh
uj db ?
help db 130 dup(?) ;6 dup(?)
a db 130 dup(?) ;6 dup(?)130 dup(?)
data ends

code segment use16
beg: mov ax,data
```

```
    mov ds,ax

    xor ax,ax
    xor cx,cx
    xor di,di

    mov cx,80h ;4;2
    mov bl,y[0]
    mov bh,mstrih
    l1: mov al,x[di]
       mul bl              ;al*bl=ax
       add ax,word ptr a[0]
       mul word ptr bh     ;ax*m'=dx_ax
       mov uj,al
       push cx
       MultAdd x[di],y,a,help,80h ;4;2 ;xjY
       MultAdd uj,m,a,help,80h ;4;2    ;ujM
       ShiftL 81h ;5;3
       pop cx
       inc di
       dec cx
       rol  eax,5  ; зсув змінної A
add        edi,eax       ; додавання до змінної E
mov        eax,edx       ; обчислення функції
xor        eax,ebp       ;  eax <- D xor C
and        eax,ebx       ;  eax <- B(D xor C)
xor        eax,edx       ;  eax <- B(D xor C) xor D
add        edi,[si]      ; +W
add        si,4
```

```
add         edi,eax  ; +f
add         edi,ecx
pop         eax
ror ebx,2
; edi-A, eax-B, ebx-C, ebp-D, edx-E
push        edi
rol edi,5    ; зсув змінної A
add         edx,edi  ; додавання до змінної E
mov         edi,ebp  ; обчислення функції -занесення D
xor         edi,ebx  ;  edi <- D xor C
and         edi,eax  ;  edi <- B(D xor C)
xor         edi,ebp  ;  eax <- B(D xor C) xor D
add         edx,[si] ; +W
add         si,4
add         edx,edi  ; +f
add         edx,ecx
pop         edi
ror eax,2
; edx-A, edi-B, eax-C, ebx-D, ebp-E
push        edx
rol edx,5    ; зсув змінної A
add         ebp,edx  ; додавання до змінної E
mov         edx,ebx  ; обчислення функції
xor         edx,eax  ;  eax <- D xor C
and         edx,edi  ;  eax <- B(B xor C)
xor         edx,ebx  ;  eax <- B(B xor C) xor D
add         ebp,[si] ; +W
add         si,4
add         ebp,edx  ; +f
```

```
add         ebp,ecx
pop         edx
ror edi,2
; ebp-A, edx-B, edi-C, eax-D, ebx-E
push        ebp
rol ebp,5    ; зсув змінної A
add         ebx,ebp  ; додавання до змінної E
mov         ebp,eax  ; обчислення функції
xor         ebp,edi  ;  eax <- D xor C
and         ebp,edx  ;  eax <- B(B xor C)
xor         ebp,eax  ;  eax <- B(B xor C) xor D
add         ebx,[si] ; +W
add         si,4
add         ebx,ebp  ; +f
add         ebx,ecx
pop         ebp
ror edx,2
add         edi,eax         ;       E+A
mov         eax,ebp         ; копіювання змінної C
xor         eax,edx         ; C xor D
and         eax,ebx         ; B(C xor D)
mov         ecx,ebp         ; повторне копіювання змінної C
and         ecx,edx         ; C&D
push ebp ; збереження змінної A
rol ebp,5 ; зсув змінної A
add ebx,ebp      ;       E+A
add ebx,K3       ; додавання константи K3
mov ebp,edi      ; копіювання змінної C
xor ebp,eax      ; C xor D
```

```
and ebp,edx      ; B(C xor D)
mov ecx,edi      ; повторне копіювання змінної C
and ecx,eax      ; C&D
xor ecx,ebp      ; кінець обчислення функції
add ebx,ecx      ; додавання функції
add ebx,[si]; додавання інф.блоку
add si,4
pop ebp     ; відновлення A
ror edx,2   ; зсув B
; ebx-A, ebp-B, edx-C, edi-D, eax-D
push ebx ; збереження змінної A
rol ebx,5 ; зсув змінної A
add eax,ebx      ;       E+A
mov ebx,edx     ; копіювання змінної C
xor ebx,edi      ; C xor D
and ebx,ebp      ; B(C xor D)
add eax,K3       ; додавання константи K3
mov ecx,edx     ; повторне копіювання змінної C
and ecx,edi      ; C&D
xor ecx,ebx      ; кінець обчислення функції
add eax,ecx      ; додавання функції
add ebx,[si]; додавання інф.блоку
add si,4
pop ebx     ; відновлення A
ror ebp,2   ; зсув B
; eax-A, ebx-B, ebp-C, edx-D, edi-D
cmp si,Re
ja  L4
jmp L3
```

```
L4:    mov ecx,K4
xor         ecx,eax        ; кінець обчислення функції
add         edi,ecx        ; додавання функції
add         edi,K3         ; додавання константи
add edi,[si]; додавання інф.блоку
add si,4
pop eax     ; відновлення A
ror ebx,2   ; зсув B
; edi-A, eax-B, ebx-C, ebp-D, edx-D
push edi  ; збереження змінної A
rol eax,5 ; зсув змінної A
add edx,edi       ;       E+A
mov edi,ebp       ; копіювання змінної D
xor edi,ebx       ; C xor D
and edi,eax       ; B(C xor D)
mov ecx,ebp       ; повторне копіювання змінної D
and ecx,ebx       ; C&D
xor ecx,edi       ; кінець обчислення функції
add edx,K3        ; додавання константи
add edx,ecx       ; додавання функції
add edx,[si]; додавання інф.блоку
add si,4
pop edi     ; відновлення A

loop l1
mov ax,4c00h
int 21h
code ends
end beg
```

Допоміжна програма знаходження модулярної інверсії

```
Program finding;
Uses crt;
Type
all=array[0..129] of byte;
Var
k,i:byte;
x,y,value,mstrih:word;
m,My:all;

Procedure Add(var dest:all;sour:all;till:word);
var
i,j:byte;
buf:word;
c:boolean;
begin
c:=false;
for j:=1 to till do
 begin
  c:=false;
  for i:=0 to 1 do
   begin
    buf:=dest[i]+sour[i];
    if c=true then buf:=buf+1;
    if buf>255 then c:=true else c:=false;
    dest[i]:=Lo(buf);
   end;
 end;
end;
```

```
Procedure clear (var sour:all);
var i:byte;
begin
 for i:=0 to 129 do
  sour[i]:=0;
end;

Begin
clrscr;
k:=8;
for i:=0 to 127 do m[i]:=$DD; m[128]:=0;m[129]:=0;

x:=2;y:=1;
for i:=2 to k do
 begin
  clear(My);
  add(My,m,y);
  value:=My[0]+256*My[1];
  if (value mod (2*x))<>1 then y:=y+x;
  x:=2*x;
  write(x);
 end;
mstrih:=x-y;
writeln('Mstrih',mstrih);
readln;
End.
```