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

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»
спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES»

Виконав: студент IV курсу, групи ІП-03
(шифр групи)

Кравчук Іван Андрійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник: доцент кафедри, к.т.н., Марковський О. П. _____
(прізвище, ім'я, по батькові) (підпис)

Консультант (нормоконтроль) доцент, Волокита А. М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент декан ФПМ, д.т.н., професор 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 – 2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем »

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____2024 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студенту

Кравчук Іван Андрійович

1. Тема проєкту *«Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES»*

керівник проєкту Марковський Олександр Петрович, доцент кафедри, к.т.н.

затверджені наказом по університету від «27» травня 2024 р. № 2112-с

2. Термін подання студентом проєкту: 08.06.2024.

3. Вихідні дані до проєкту: технічне завдання, теоретичні дані.

4. Зміст пояснювальної записки:

1. ОГЛЯД СУЧАСНИХ АЛГОРИТМІВ СИМЕТРИЧНОГО ШИФРУВАННЯ
2. ПОСТАНОВКА ЗАДАЧІ
3. АЛГОРИТМ ШИФРУВАННЯ RIJNDAEL
4. РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ АЛГОРИТМУ RIJNDAEL
5. АПАРАТНА РЕАЛІЗАЦІЯ АЛГОРИТМУ RIJNDAEL

6. ПОРІВНЯЛЬНИЙ АНАЛІЗ ПРОГРАМНОЇ ТА АПАРАТНОЇ РЕАЛІЗАЦІЇ АЛГОРИТМУ RIJNDAEL.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо), юзкейс-діаграма (принципова схема), схема бази данних (функціональна схема), структурна схема програмної компоненти (структурна схема), програмний код основних компонентів системи.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Волокита А. М.		

7. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	*Затвердження теми проєкту*	*20.11.2023 - 23.01.2024*	
2	*Вивчення та аналіз завдання*	*23.01.2024 - 06.03.2024*	
3	*Вивчення документації, проектування архітектури системи*	*06.03.2024 - 20.03.2024*	
4	*Програмна реалізація системи*	*20.03.2024 - 06.05.2024*	
5	*Оформлення пояснювальної записки*	*06.05.2024 - 05.06.2024*	
6	*Захист програмного продукту*	*05.06.2024*	
7	*Передзахист*	*09.06.2024*	
8	*Захист*	*24.06.2024*	

Студент Іван Кравчук

Керівник Олександр МАРКОВСЬКИЙ

АНОТАЦІЯ

Дипломний проект присвячено підвищенню продуктивності апартної та програмної реалізації криптографічного алгоритму симетричного шифрування – AES. Проаналізовано обчислювальні процедури, що лежать в основі алгоритму AES. В результаті проведеного аналізу запропоновані підходи до прискорення реалізації алгоритму AES. Зокрема, було досліджено можливості застосування табличних обчислень з використанням передобчислень. Така можливість може бути застосована на рівні байтових перетворень. При цьому обробка декількох байтів матриці станів може виконуватися одночасно шляхом об'єднання таблиць.

Іншою можливістю для прискорення виконання криптографічного алгоритму AES при апаратній і програмній реалізації є використання діагональної обробки матриці станів.

Розроблено структуру апаратної реалізації алгоритму AES з підвищеною швидкодією. Розроблена також програмна реалізація алгоритму AES.

ABSTRACT

The diploma project is dedicated to enhancing the performance of both hardware and software implementations of the symmetric encryption algorithm AES. The computational procedures underlying the AES algorithm have been analyzed. As a result of the conducted analysis, approaches for accelerating the implementation of the AES algorithm have been proposed. In particular, the potential for using table-based computations with precomputation has been explored. This approach can be applied at the byte transformation level, where multiple bytes of the state matrix can be processed simultaneously by combining tables.

Another possibility for accelerating the execution of the AES cryptographic algorithm in both hardware and software implementations is the use of diagonal processing of the state matrix.

A structure for a high-performance hardware implementation of the AES algorithm has been developed. A software implementation of the AES algorithm has also been developed.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
	А4	***ІАЛЦ.467200.002 ТЗ***	Програмні та апаратні засоби	4		
			прискореної реалізації			
			криптографічного алгоритму AES			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Програмні та апаратні засоби	67		
			прискореної реалізації			
			криптографічного алгоритму AES			
			Пояснювальна записка			
	А3	***ІАЛЦ.467200.004 Д1***	Програмні та апаратні засоби	1		
			прискореної реалізації			
			криптографічного алгоритму AES			
			Блок-схема програмної			
			реалізації алгоритму AES			
			(функціональна схема)			
	А3	***ІАЛЦ.467200.005 Д2***	Програмні та апаратні засоби	1		
			прискореної реалізації			
			криптографічного алгоритму AES			
			Електрична схема			
			криптографічного співпроцесору			
			(принципова схема)			
	А3	***ІАЛЦ.467200.006 Д3***	Програмні та апаратні засоби	1		
			прискореної реалізації			
			криптографічного алгоритму AES			
			Електрична схема системи			
			криптографічного захисту			
			(структурна схема)			
	А4	***ІАЛЦ.467200.007 Д4***	Програмні та апаратні засоби	16		
			прискореної реалізації			
			криптографічного алгоритму AES			
			Програмний код основних			
			Компонентів програми			

Зм	***Лист***	***№ докум.***	***Підп***	***Дата***		
Розроб		Кравчук І. А.				
Перев		Марковський О. П.				

ІАЛЦ.467200.001 ОА

Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES	Літ.	Аркуш	Аркушів
		1	1
	КПІ ім. Ігоря Сікорського Група ІП-03		

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES»

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Розробка програмних та апаратних засобів прискореної реалізації алгоритму симетричного шифрування AES. Ця система спрямована на створення надійної платформи для шифрування та дешифрування даних, забезпечуючи високий рівень безпеки інформації. Основною областю застосування є захист інформації в комп'ютерних системах та мережах, зокрема у сфері кібербезпеки та захисту даних.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є виконання дипломної роботи для здобуття кваліфікаційного рівня «бакалавр інженерії програмного забезпечення». Проєкт був затверджений факультетом «Інформатики та обчислювальної техніки» на кафедрі обчислювальної техніки Національного технічного університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення ефективної системи програмних та апаратних засобів для реалізації алгоритму симетричного шифрування Rijndael, яка забезпечить надійне та швидке шифрування і дешифрування даних. Призначенням розробки є впровадження інноваційного рішення для забезпечення безпеки інформації в комп’ютерних системах та мережах, що відповідає сучасним вимогам і потребам у сфері кібербезпеки.

4 ДЖЕРЕЛА РОЗРОБКИ

Аналітичні звіти та дослідження ринку шифрування та кібербезпеки, технічна документація та посібники з реалізації алгоритму Rijndael, включаючи специфікації AES та рекомендації щодо використання мов програмування для криптографії. Також використовувалися практичні керівництва з оптимізації обчислень і використання апаратних рішень для

підвищення продуктивності, а також інформація з наукових статей і інтернет-ресурсів, що описують найкращі практики в реалізації криптографічних алгоритмів.

5 ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до розробленого продукту

Функціональні вимоги:

- Ініціалізація та керування криптографічними ключами.
- Шифрування та дешифрування даних з використанням алгоритму Rijndael.
- Підтримка обробки інформаційних блоків різного розміру (128, 192, 256 біт).

Нефункціональні вимоги:

- Висока продуктивність і швидкість обробки даних.
- Забезпечення високого рівня безпеки та конфіденційності даних.
- Зручний та інтуїтивно зрозумілий інтерфейс.
- Надійність і стійкість до збоїв.
- Можливість розширення та масштабування системи.

5.2 Вимоги до програмного забезпечення

- Операційні системи: Windows, Linux, або MacOS.
- Мова програмування: C++ (підтримка стандарту C++11 і вище).
- Інтегроване середовище розробки: Microsoft Visual Studio 2019 або вище.
- Бібліотеки та інструменти: OpenSSL для криптографічних функцій, бібліотеки для роботи з пам'яттю і оптимізації обчислень.
- Система управління версіями: Git.

5.3 Вимоги до апаратної частини

- 32-розрядний (x86) або 64-розрядний (x64) процесор із тактовою частотою 1 ГГц або швидший.
- ROM не менше, ніж 32 ГБ.
- RAM не менше, ніж 4 ГБ.

6 ЕТАПИ РОЗРОБКИ

Етап	Дата
Затвердження теми проєкту	20.11.2023 - 23.01.2024
Вивчення та аналіз завдання	23.01.2024 - 06.03.2024
Проектування архітектури системи	06.03.2024 - 20.03.2024
Програмна реалізація системи	20.03.2024 - 06.05.2024
Оформлення пояснювальної записки	06.05.2024 - 10.06.2024

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту
на тему: «Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES»

Київ – 2024

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

DES (Data Encryption Standard) – стандарт шифрування даних

AES (Advanced Encryption Standard) – альтернативний стандарт шифрування

GF (Galoise Fields) – поля Галуа

АЛУ – арифметико-логічний пристрій

РР – регістр результату

РО – регістр операндів

РВМ – регістр величини модуля

КП – криптографічний процесор

ЦП – центральний процесор

ВСТУП

У сучасному світі інформація є одним із найцінніших ресурсів, і захист цієї інформації стає все більш важливим завданням у різних сферах життя. Від банківської справи до урядових установ, від приватного бізнесу до особистого спілкування в інтернеті – захист даних від несанкціонованого доступу та використання набуває критичного значення. Одним із найефективніших методів забезпечення конфіденційності та цілісності даних є криптографія, яка забезпечує перетворення даних у незрозумілий для неавторизованих осіб вигляд, який можна повернути до початкового стану тільки за допомогою спеціального ключа.

Серед різноманітних методів криптографічного захисту інформації особливе місце займають алгоритми симетричного шифрування, які відзначаються своєю високою швидкістю роботи та ефективністю, що робить їх придатними для захисту великих обсягів даних у реальному часі. Одним із найвідоміших і найбільш використовуваних алгоритмів симетричного шифрування протягом тривалого часу був DES (Data Encryption Standard). Проте з розвитком технологій і зростанням обчислювальних потужностей DES став вразливим до криптоаналітичних атак, що спонукало до пошуку нових, більш надійних алгоритмів шифрування.

У відповідь на ці виклики був розроблений проект Advanced Encryption Standard (AES), результатом якого став алгоритм Rijndael. Цей алгоритм, розроблений у 1998 році Джоаном Даеменом і Вінсентом Рейменом, був обраний новим стандартом симетричного шифрування даних [1]. Rijndael має кілька переваг перед своїми попередниками, серед яких високий рівень захищеності, гнучкість у виборі розмірів блоку даних і ключа, а також ефективність як у програмній, так і в апаратній реалізації. Завдяки цим характеристикам він став універсальним інструментом для захисту інформації в різних застосуваннях, від мобільних пристроїв до великих серверів.

Оптимізація алгоритму Rijndael для досягнення максимальної продуктивності є важливим завданням в умовах сучасних високошвидкісних мереж і систем, де затримки в обробці даних можуть мати критичне значення. Апаратна реалізація криптографічних алгоритмів дозволяє значно прискорити процес шифрування та дешифрування, зменшуючи навантаження на центральний процесор і забезпечуючи більш високий рівень захисту за рахунок ізоляції криптографічних операцій від загальної обчислювальної системи. Застосування програмованих логічних інтегральних схем (FPGA) і спеціалізованих надвеликих інтегральних схем (ASIC) для реалізації криптографічних алгоритмів стає дедалі популярнішим рішенням для забезпечення високої продуктивності і надійності шифрування даних.

Розробка ефективних програмних засобів для шифрування і дешифрування даних, генерації та управління ключами також є важливою складовою захисту інформації. Використання передобчислень, табличних підстановок та інших методів оптимізації дозволяє значно підвищити швидкість і ефективність криптографічних операцій. Наприклад, застосування T-таблиць для обчислення байтових підстановок і лінійних перетворень у алгоритмі Rijndael може значно зменшити кількість необхідних операцій, підвищуючи тим самим загальну продуктивність шифрування.

У контексті сучасних інформаційних технологій, де обробка великих обсягів даних у реальному часі є нормою, питання оптимізації криптографічних алгоритмів стає ще більш актуальним. Висока продуктивність, гнучкість у налаштуванні та ефективність у різних умовах – ось основні вимоги, які висуваються до сучасних криптографічних рішень. Алгоритм Rijndael, завдяки своїм унікальним властивостям і можливостям оптимізації, продовжує залишатися одним із найефективніших і найбільш використовуваних інструментів для захисту інформації.

Таким чином, розвиток і вдосконалення алгоритмів симетричного шифрування, таких як Rijndael, є важливим напрямком у забезпеченні інформаційної безпеки. Використання сучасних технологій, як програмних, так і апаратних, дозволяє досягти високого рівня захисту даних, забезпечуючи при цьому необхідну продуктивність і гнучкість у різних умовах використання.

РОЗДІЛ 1. ОГЛЯД СУЧАСНИХ АЛГОРИТМІВ СИМЕТРИЧНОГО ШИФРУВАННЯ

1.1 Основні поняття та класифікація алгоритмів симетричного шифрування

Симетричне шифрування є одним з фундаментальних методів забезпечення конфіденційності інформації. Цей вид шифрування характеризується тим, що один і той самий ключ використовується як для шифрування, так і для дешифрування даних. Основною перевагою симетричного шифрування є його висока швидкість і ефективність, що робить його ідеальним для шифрування великих обсягів даних [3].

Симетричні алгоритми шифрування можна класифікувати за кількома критеріями. Перш за все, вони поділяються на блокові та потокові шифри. Блокові шифри обробляють дані фіксованими блоками, зазвичай розміром 64 або 128 біт, тоді як потокові шифри шифрують дані побітно або побайтово.

Блокові шифри:

Блокові шифри перетворюють вихідні дані (plaintext) у шифрований текст (ciphertext) блоками фіксованої довжини. Кожен блок обробляється незалежно від інших за допомогою ключа. Прикладами блокових шифрів є DES, Triple DES, AES, а також алгоритми менш відомих стандартів. Однією з основних характеристик блокових шифрів є їх режим роботи, який визначає, як блоки даних шифруються відносно один одного. Серед популярних режимів роботи можна виділити:

– Electronic Codebook (ECB) – кожен блок шифрується окремо, що може призводити до повторень у шифрованому тексті, якщо однакові блоки вихідного тексту зустрічаються кілька разів.
– Cipher Block Chaining (CBC) – кожен блок вихідного тексту перед шифруванням поєднується з попереднім блоком шифрованого тексту за допомогою операції XOR, що значно підвищує криптостійкість.
– Cipher Feedback (CFB) та Output Feedback (OFB) – режими, які перетворюють блокові шифри на потокові, забезпечуючи шифрування даних будь-якої довжини без необхідності додавання наповнювача до блоків.

Потокові шифри:

Потокові шифри обробляють дані побітно або побайтово, що дозволяє шифрувати потоки даних у режимі реального часу. Основною перевагою потокових шифрів є їх висока швидкість, проте вони можуть бути менш стійкими до деяких видів атак, якщо не використовувати спеціальні методи захисту. Прикладами потокових шифрів є RC4, Salsa20, і ChaCha20.

Основні принципи шифрування:

К. Шеннон, один із засновників теорії інформації, сформулював два основні принципи, на яких базуються всі сучасні алгоритми симетричного шифрування: перемішування (confusion) та розсіювання (diffusion) [3]. Перемішування забезпечує складне нелінійне перетворення вихідних даних, щоб зробити зв'язок між шифрованим текстом і ключем максимально заплутаним. Розсіювання полягає у розподілі впливу кожного біта вихідного тексту на кілька бітів шифрованого тексту, що ускладнює криптоаналіз.

Алгоритми симетричного шифрування:

- DES (Data Encryption Standard) був прийнятий як стандарт у 1977 році і став одним із найвідоміших алгоритмів шифрування. Він використовує 56-бітовий ключ для шифрування даних блоками по 64 біти. DES базується на мережі Фейстеля, що включає 16 раундів шифрування, де кожен раунд складається з перестановки та заміни бітів.
- Triple DES (3DES). Через обмежену довжину ключа DES став вразливим до криптоаналізу. Для підвищення стійкості був розроблений Triple DES, який застосовує алгоритм DES тричі з трьома різними ключами, що фактично збільшує довжину ключа до 168 біт. Це значно підвищує стійкість шифру, хоча й робить його повільнішим.
- AES (Advanced Encryption Standard), або алгоритм Rijndael, був прийнятий як стандарт у 2001 році, замінивши DES. AES підтримує ключі довжиною 128, 192 і 256 біт, та шифрує дані блоками по 128 біт. Він складається з кількох раундів, кількість яких залежить від довжини ключа: 10 раундів для 128-бітного ключа, 12 для 192-бітного та 14 для 256-бітного ключа. Кожен раунд включає операції підстановки, перемішування та поєднання з ключем [4].
- RC4 є одним із найвідоміших потокових шифрів, розробленим Роном Рівестом у 1987 році. Він використовує змінний ключ довжиною до 256 біт і відомий своєю простотою та швидкістю. Проте вразливості, виявлені в RC4, призвели до того, що його використання зменшилося у сучасних системах.
- Salsa20 і ChaCha20. Ці потокові шифри були розроблені для забезпечення високої швидкості та безпеки. Вони використовують 256-бітні ключі та є стійкими до різних видів атак, забезпечуючи при цьому ефективне шифрування даних у реальному часі.

Таким чином, алгоритми симетричного шифрування залишаються важливим інструментом у забезпеченні безпеки даних. Різноманітність існуючих методів дозволяє обирати оптимальні рішення для конкретних завдань, забезпечуючи необхідний баланс між швидкістю, ефективністю та рівнем захисту.

1.2 Огляд основних симетричних алгоритмів шифрування

Симетричні алгоритми шифрування є основним інструментом для забезпечення конфіденційності даних. Вони засновані на використанні одного і того ж ключа для шифрування та дешифрування інформації. У цьому розділі розглянемо деякі з найпоширеніших симетричних алгоритмів шифрування, включаючи їх структуру, переваги та недоліки.

1.2.1 DES (Data Encryption Standard)

DES є одним із найвідоміших і широко використовуваних алгоритмів шифрування, хоча сьогодні він вважається застарілим через відносно коротку довжину ключа, яка становить 56 біт. Алгоритм використовує структуру Фейстеля, що складається з 16 раундів, кожен з яких включає перестановки та заміни бітів.

DES базується на використанні мережі Фейстеля, яка забезпечує його оборотність. Структура алгоритму зображена на рисунку 1.1. Основні математичні операції включають:

1. Початкова перестановка (IP): перестановка бітів вхідного блоку.
2. Функція F: нелінійне перетворення, що включає підстановку (S-блоки) та перестановку.
3. XOR операція: додавання за модулем 2 з ключем раунду.
4. Зворотна перестановка (IP-1): кінцева перестановка після всіх раундів.

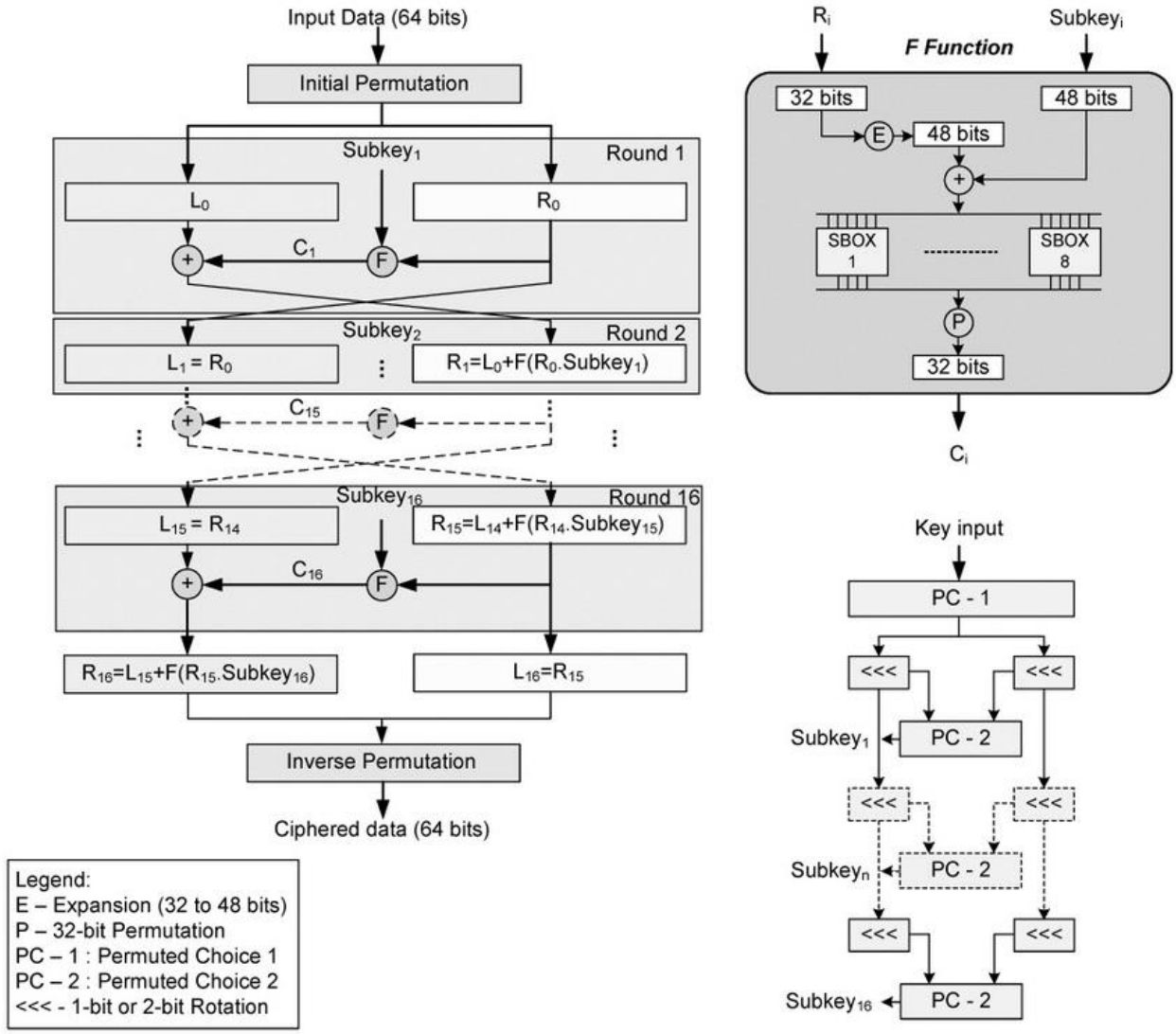


Рисунок 1.1 – Структура алгоритму DES

Основним недоліком DES є його вразливість до атак грубої сили, оскільки сучасні обчислювальні потужності дозволяють перебрати всі можливі ключі за відносно короткий час.

1.2.2 Triple DES (3DES)

Triple DES був розроблений як покращення до DES, щоб подолати його вразливості. Алгоритм використовує три ключі по 56 бітів кожен і виконує три раунди шифрування, розшифрування і знову шифрування даних.

3DES складається з послідовного застосування DES у режимі Encrypt-Decrypt-Encrypt (EDE): $C = E_{K3}(D_{K2}(E_{K1}(P)))$, де E – шифрування, D – дешифрування, K1, K2, K3 – три різні ключі, P – відкритий текст, C – шифротекст. Схема шифрування та дешифрування Triple DES на рисунку 1.2.

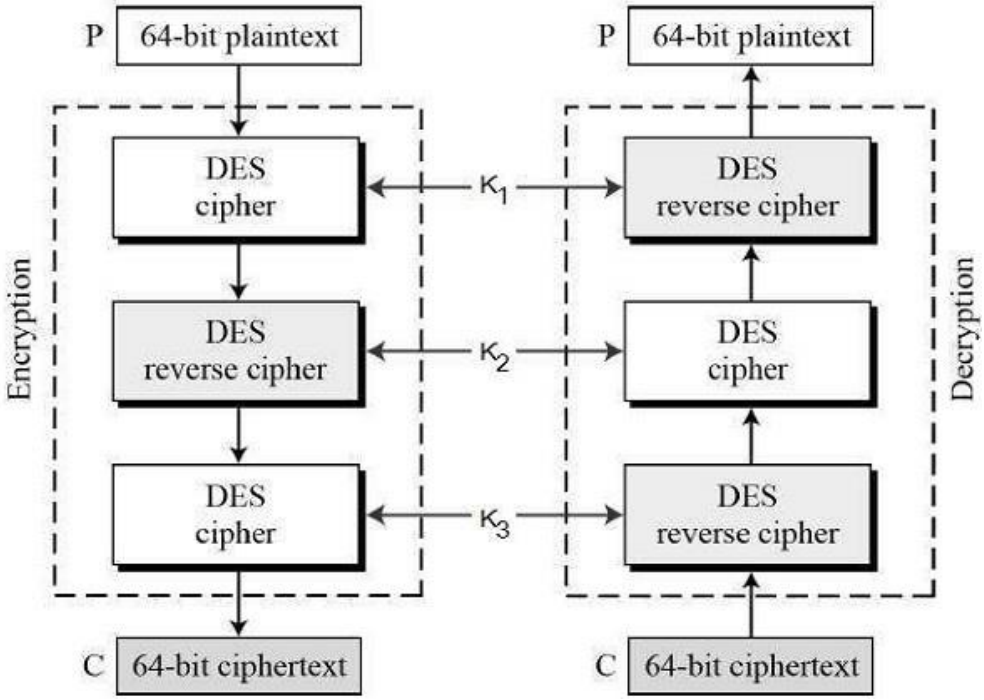


Рисунок 1.2 – Схема шифрування Triple DES

1.2.3 AES (Advanced Encryption Standard)

Алгоритм AES прийнятий як стандарт шифрування у 2001 році та підтримує ключі довжиною 128, 192 і 256 біт, шифруючи дані блоками по 128 біт. Кількість раундів залежить від довжини ключа: 10 раундів для 128-бітного ключа, 12 для 192-бітного та 14 для 256-бітного ключа.

Основні операції в кожному раунді AES зображені на рисунку 1.3 та включають у себе:

1. SubBytes: нелінійна заміна кожного байту за допомогою S-блоків.
2. ShiftRows: циклічний зсув рядків матриці стану.

3. MixColumns: лінійне перетворення, що змішує байти в кожному стовпці.
4. AddRoundKey: додавання за модулем 2 кожного байта матриці стану з байтом розширеного ключа.

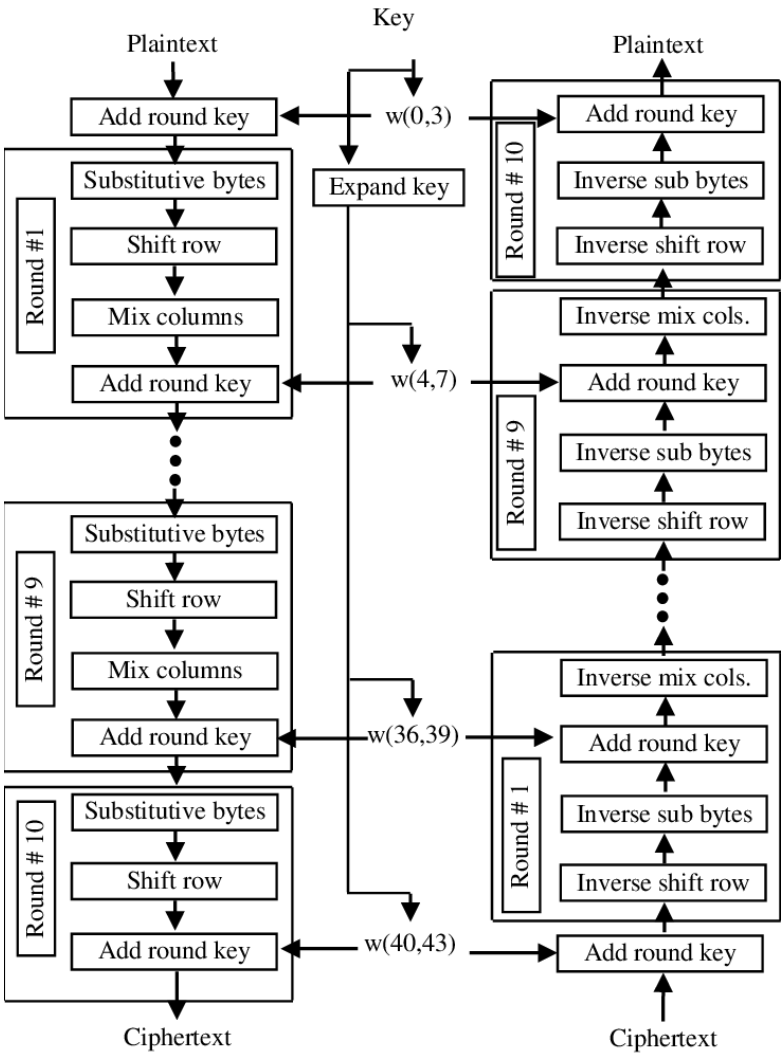


Рисунок 1.3 – Схема роботи алгоритму AES

AES відомий своєю ефективністю та високим рівнем безпеки, що робить його придатним для широкого спектру застосувань.

1.2.4 RC4 (Rivest Cipher 4)

RC4 є потоковим шифром, який використовує змінний ключ довжиною до 256 біт. Його основна перевага – це швидкість та простота реалізації. Однак

RC4 має кілька відомих вразливостей, які роблять його менш надійним для використання в нових системах.

RC4 працює за принципом генерації псевдовипадкового потоку байтів (ключового потоку), який потім використовується для шифрування/ дешифрування даних за допомогою операції XOR: $C_i = S_i \oplus K_i$, де C_i – шифротекст, S_i – відкритий текст, K_i – ключовий потік. Схема шифрування одного символу відкритого тексту за допомогою RC4 зображена на рисунку 1.4.

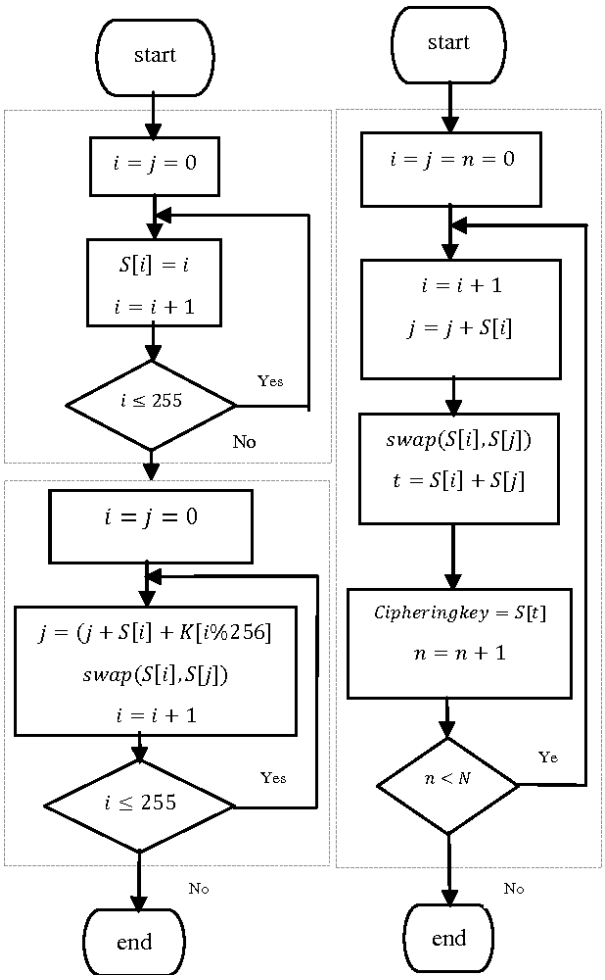


Рисунок 1.4 – Схема шифрування одного символу RC4

Недоліки RC4 включають слабкість на початкових байтах вихідного потоку та вразливість до різних криптографічних атак, таких як атака на основі аналізу вихідного потоку.

1.2.5 Salsa20 і ChaCha20

Salsa20 і ChaCha20 – це потокові шифри, що забезпечують високу швидкість шифрування та високий рівень безпеки. Вони використовують 256-бітні ключі і відомі своєю стійкістю до атак.

Обидва алгоритми використовують додавання, побітове XOR і циклічні зсуви для обробки даних у блоках. Наприклад, ChaCha20 складається з 20 раундів перетворень, що забезпечують ефективне та безпечне шифрування. Основний блок системи quarterround, на основі якого будуються інші перетворення системи, описаний на рисунку 1.5.

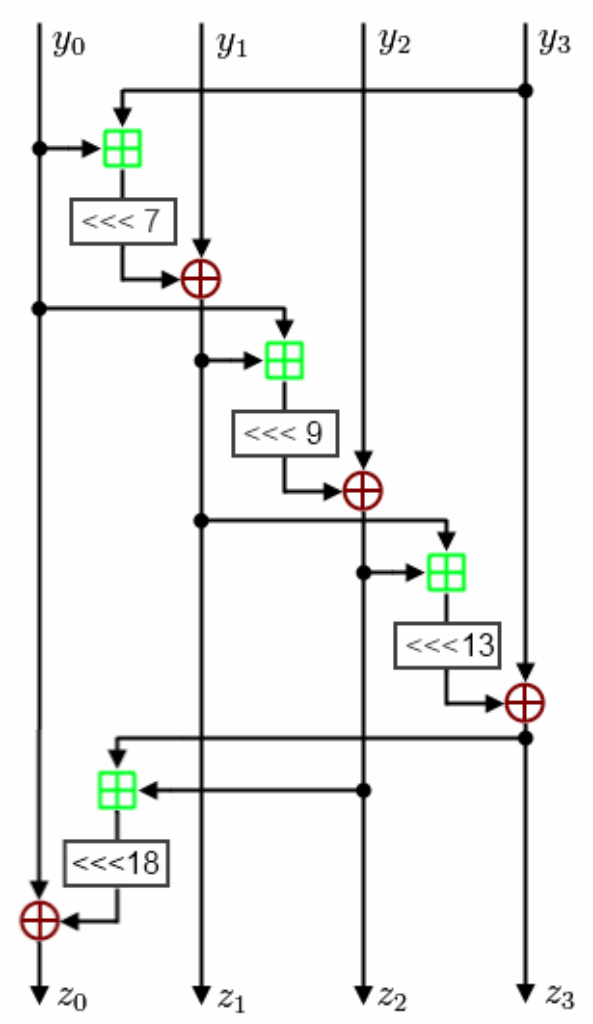


Рисунок 1.5 – Функція чверть-раунду алгоритму Salsa20

Переваги Salsa20 і ChaCha20 включають їх високу швидкість і стійкість до криптографічних атак, таких як атаки на основі кореляційного аналізу.

Симетричні алгоритми шифрування мають свої переваги та недоліки. Переваги включають високу швидкість шифрування та ефективне використання обчислювальних ресурсів. Недоліки можуть включати вразливість до атак грубої сили при використанні коротких ключів та необхідність безпечного обміну ключами між сторонами.

Таким чином, вибір симетричного алгоритму шифрування залежить від конкретних вимог до безпеки та продуктивності системи.

1.3 Аналіз стандарту DES

Data Encryption Standard (DES) використовує блокову схему для шифрування даних, при цьому розмір блоку становить 64 біти, а ключ – 56 біт. DES складається з 16 раундів, кожен з яких виконує серію перестановок, замін та підстановок. Алгоритм побудований на мережі Фейстеля, що дозволяє кожному раунду бути оберненим, забезпечуючи можливість дешифрування.

Процес шифрування в DES починається з початкової перестановки (IP), яка змінює порядок бітів вхідного блоку. Ця перестановка готує дані для подальших раундів обробки, хоча і не додає криптографічної стійкості. Після початкової перестановки алгоритм проводить послідовні раунди шифрування.

Кожен раунд шифрування в DES включає кілька кроків:

1. Розділення блоку: Вхідний блок ділиться на дві частини по 32 біти – ліва (L) та права (R).
2. Функція раунду: Права частина блоку обробляється функцією F, результат якої поєднується з лівою частиною за допомогою операції XOR.

3. Обмін частинами: Ліва та права частини обмінюються місцями для наступного раунду.

Функція F є ключовою частиною DES і включає кілька етапів:

1. Розширення: Права частина блоку розширюється з 32 до 48 бітів.
2. Додавання ключа: Розширений блок додається за допомогою XOR з ключем поточного раунду.
3. S-блоки: Результат ділиться на вісім 6-бітових блоків, які проходять через S-блоки для нелінійної заміни, зменшуючи розмір до 4 бітів.
4. Перестановка: Перестановка 32-бітового виходу S-блоків завершує функцію F.

Переваги DES:

– Простота реалізації: DES легко реалізується як в апаратному, так і в програмному забезпеченні, що зробило його популярним вибором для багатьох застосувань.
– Широке впровадження: Завдяки статусу федерального стандарту, DES був широко впроваджений у багатьох системах та стандартах, включаючи банківські системи, телекомунікації та захист даних [5].

Недоліки DES:

– Коротка довжина ключа: 56-бітний ключ виявився занадто коротким для сучасних обчислювальних потужностей. За допомогою атак грубої сили можна швидко знайти ключ і розшифрувати повідомлення.
– Вразливість до криптоаналізу: DES вразливий до кількох видів криптографічних атак, включаючи диференційний криптоаналіз та лінійний криптоаналіз. Ці атаки значно знижують ефективність DES у забезпеченні безпеки.

Для подолання недоліків DES було розроблено кілька вдосконалень. Найпоширенішим з них є Triple DES (3DES), який використовує три послідовних застосування DES з трьома різними ключами для збільшення ефективної довжини ключа до 168 біт. Це значно підвищило стійкість до атак грубої сили, але також зменшило швидкість шифрування.

Незважаючи на свої недоліки, DES все ще використовується в деяких системах, особливо у випадках, де сумісність з існуючими системами важливіша за безпеку. Проте більшість нових систем і протоколів переходять на більш безпечні алгоритми, такі як AES, які забезпечують значно вищий рівень безпеки [6].

Таким чином, DES відігравав важливу роль у розвитку криптографії, але його обмеження змушують шукати нові, більш надійні методи шифрування для сучасних потреб у безпеці.

1.4 Аналіз стандарту AES

AES є блоковим шифром, який обробляє дані блоками розміром 128 біт. Алгоритм підтримує три різні довжини ключів: 128, 192 і 256 біт, що забезпечує різні рівні безпеки. Основні етапи шифрування даних в AES включають:

1. Початкова перестановка (AddRoundKey): Початкове додавання ключа до блоку даних.
2. Основні раунди: Складаються з чотирьох операцій:
   - SubBytes: Нелінійна заміна байтів за допомогою S-блоків.
   - ShiftRows: Циклічний зсув рядків матриці стану.
   - MixColumns: Лінійне перетворення колонок матриці стану.
   - AddRoundKey: Додавання ключа раунду.

3. Фінальний раунд: Включає три з чотирьох основних операцій без MixColumns.

Переваги AES:

– Високий рівень безпеки: Завдяки довжині ключа до 256 біт і складним раундам шифрування, AES забезпечує високий рівень безпеки проти сучасних криптографічних атак.
– Ефективність: AES є ефективним як в апаратній, так і в програмній реалізації. Алгоритм оптимізований для швидкої обробки даних, що робить його придатним для використання в різних застосунках, від мобільних пристроїв до високопродуктивних серверів.
– Гнучкість: Підтримка трьох різних довжин ключів дозволяє вибирати між продуктивністю і безпекою залежно від потреб конкретної системи.

Недоліки AES:

– Складність реалізації: Високий рівень складності алгоритму може ускладнити його правильну реалізацію, що може призвести до вразливостей.
– Вимоги до ресурсів: Незважаючи на свою ефективність, AES може вимагати більше ресурсів порівняно з деякими простішими алгоритмами, що може бути проблемою для систем з обмеженими ресурсами.

Порівняння AES і DES:

– Безпека: AES значно перевершує DES за рівнем безпеки. У той час як DES використовує 56-бітний ключ, AES підтримує ключі довжиною до 256 біт, що робить його стійкішим до атак грубої сили.

– Швидкість і ефективність: AES є більш ефективним в апаратних реалізаціях завдяки своєму дизайну, який включає простіші і швидші операції. DES, з його більш складними перестановками і меншими блоками, є менш ефективним.
– Структурні відмінності: DES використовує мережу Фейстеля, де кожен раунд є симетричним і легко обертається, тоді як AES використовує підстановки і перестановки, що робить його структуру більш складною, але і більш стійкою до різних типів атак [2].

Алгоритм AES базується на алгебраїчних операціях у полях Галуа (GF(2^8)). Операції над байтами (множення, додавання) виконуються у фінальному полі, що забезпечує нелінійність і стійкість до криптоаналізу. S-блоки, використовувані в SubBytes, є інверсіями в полі Галуа, що робить їх математично обґрунтованими і безпечними.

Таким чином, AES став новим стандартом шифрування завдяки своїй надійності, ефективності та гнучкості. Його використання в сучасних криптографічних системах забезпечує високий рівень захисту даних, що відповідає вимогам сучасного світу.

Висновки до розділу 1

У цьому розділі було розглянуто основні поняття та класифікацію алгоритмів симетричного шифрування, а також проведено детальний аналіз двох ключових стандартів шифрування: DES та AES. Кожен з цих алгоритмів має свої унікальні характеристики, переваги та недоліки, що робить їх підходящими для різних застосувань.

DES, як один із найперших широко визнаних стандартів шифрування, зробив великий внесок у розвиток криптографії, але його обмежена довжина ключа і деякі інші вразливості зробили його менш придатним для сучасних вимог безпеки. Його основні переваги включають простоту реалізації та оборотність процесу шифрування. Однак, зі зростанням обчислювальної потужності, атаки на DES стали більш реальними, що вимагало розробки більш надійного стандарту [2].

AES, обраний для заміни DES, демонструє значно вищий рівень безпеки завдяки використанню більших ключів та складнішої структури алгоритму. Він підтримує різні довжини ключів, що дозволяє балансувати між безпекою та продуктивністю. Основні переваги AES включають високий рівень криптостійкості, ефективність в апаратних реалізаціях та гнучкість використання. Алгоритм базується на складних математичних перетвореннях у полях Галуа, що робить його стійким до різних типів атак [7].

Порівняння DES і AES показало, що, хоча обидва алгоритми є ефективними методами шифрування, AES забезпечує значно вищий рівень безпеки і є більш придатним для сучасних застосувань. Це підтверджується його широким використанням у різних сферах, включаючи захист даних в урядових, фінансових та корпоративних системах.

Таким чином, аналіз сучасних алгоритмів симетричного шифрування демонструє важливість вибору правильного алгоритму для забезпечення належного рівня захисту інформації. Подальші розділи цієї роботи будуть зосереджені на розробці та оптимізації алгоритмів шифрування, що враховують сучасні вимоги до безпеки і продуктивності.

РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ

2.1. Формулювання задачі дослідження

В сучасному світі захист інформації стає критично важливим завданням через постійне зростання обсягу даних і розвитку технологій передачі інформації. Основною метою даного дослідження є розробка ефективних апаратних та програмних засобів для реалізації криптографічного алгоритму Rijndael, який був обраний як стандарт шифрування (AES) завдяки своїй високій криптографічній стійкості та ефективності. Важливим аспектом є також оптимізація обчислювальних процедур для прискорення роботи алгоритму без втрати його стійкості.

2.2. Цілі та критерії ефективності

Основними цілями дослідження є:

1. Аналіз існуючих методів і алгоритмів симетричного шифрування: вивчення їхніх основних принципів, сильних і слабких сторін.
2. Розробка методів оптимізації обчислювальних процедур алгоритму Rijndael: використання діагональної організації обчислень та передобчислень для прискорення роботи.
3. Створення ефективних програмних засобів реалізації алгоритму: забезпечення високої швидкості та надійності програмної реалізації.
4. Розробка апаратних засобів реалізації алгоритму: створення прототипу криптопроцесора для реалізації алгоритму на апаратному рівні.

Критеріями ефективності розроблених засобів є:

1. Швидкодія – час виконання шифрування та дешифрування даних.
2. Криптографічна стійкість: стійкість до сучасних методів криптоаналізу.

3. Ефективність використання ресурсів: обсяг пам'яті та процесорний час, необхідний для виконання операцій.
4. Гнучкість реалізації: можливість адаптації алгоритму для різних апаратних та програмних платформ.

2.3. Опис вхідних та вихідних даних

Вхідні дані для алгоритму шифрування Rijndael включають:

– Текст для шифрування (plaintext): довільний блок даних, який підлягає шифруванню.
– Ключ шифрування: послідовність бітів певної довжини (128, 192 або 256 біт), яка використовується для генерування підключів та проведення криптографічних перетворень.

Вихідні дані включають:

– Зашифрований текст (ciphertext): результат шифрування вхідного тексту, який представляє собою закодований блок даних.
– Дешифрований текст: результат обробки зашифрованого тексту з використанням того ж ключа, що й для шифрування, який повинен співпадати з початковим вхідним текстом.

2.4. Обмеження та припущення

Основними обмеженнями та припущеннями є:

– Апаратні обмеження: розмір доступної пам'яті та продуктивність процесора.
– Безпека ключів: передбачається, що ключі зберігаються в захищеному середовищі і не доступні для несанкціонованого доступу.
– Мережеві умови: передбачається, що дані передаються через канали з обмеженою пропускною здатністю і можливими затримками.

– Аналіз загроз: передбачається наявність захисту від відомих методів криптоаналізу, таких як лінійний та диференційний криптоаналіз.
– Обмеження на довжину ключа та блоку даних – алгоритм Rijndael підтримує фіксовані довжини ключа та блоку даних (128, 192 та 256 біт).
– Припущення щодо середовища виконання – передбачається, що реалізація алгоритму буде виконуватися на сучасних комп'ютерних системах та вбудованих пристроях з підтримкою багатоядерної обробки та програмованих логічних інтегральних схем (ПЛІС).

2.5 Аналіз вимог до програмних засобів шифрування

2.5.1 Функціональні вимоги

Програмні засоби для реалізації алгоритму шифрування повинні відповідати ряду функціональних вимог, що забезпечують коректну та ефективну роботу системи. Основні з них включають:

Шифрування та дешифрування даних: Програма повинна надавати можливість шифрувати та дешифрувати дані різної довжини з використанням заданого ключа. Алгоритм Rijndael повинен бути реалізований для роботи з блоками розміром 128, 192 або 256 бітів.

Генерація ключів: Програма повинна мати функцію генерації криптографічних ключів певної довжини. Генерація ключів повинна здійснюватись із використанням криптографічно стійких методів.

Зберігання та управління ключами: Програма повинна забезпечувати безпечне зберігання криптографічних ключів та надавати інструменти для управління ними, включаючи імпорт, експорт та знищення ключів.

Інтерфейс користувача: Програма повинна мати зручний інтерфейс користувача для взаємодії з функціями шифрування та управління ключами.

Інтерфейс повинен бути інтуїтивно зрозумілим та забезпечувати легкий доступ до основних функцій.

2.5.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики програмних засобів та їхню відповідність загальним вимогам до продуктивності, безпеки та надійності:

Продуктивність: Програмні засоби повинні забезпечувати високу швидкість обробки даних, мінімізуючи час, необхідний для шифрування та дешифрування великих обсягів інформації.

Сумісність: Програма повинна бути сумісною з різними операційними системами та платформами. Вона повинна підтримувати роботу на різних апаратних конфігураціях.

Надійність: Програмні засоби повинні бути надійними, тобто працювати без збоїв та помилок при виконанні всіх функцій. Це включає захист від випадкових збоїв та забезпечення цілісності даних при шифруванні та дешифруванні.

Безпека: Програма повинна забезпечувати високий рівень безпеки збереження та обробки даних. Це включає захист від несанкціонованого доступу до ключів та зашифрованих даних, а також захист від криптографічних атак.

Масштабованість: Програма повинна бути здатна ефективно обробляти зростаючі обсяги даних без суттєвого зниження продуктивності. Вона повинна легко масштабуватися для роботи з великими інформаційними системами.

2.6 Аналіз вимог до апаратних засобів шифрування

2.6.1 Продуктивність та ефективність

Апаратні засоби реалізації алгоритму шифрування повинні відповідати вимогам високої продуктивності та ефективності, що забезпечується за рахунок спеціалізованих рішень:

Швидкість обробки: Апаратні засоби повинні забезпечувати високошвидкісну обробку даних, що дозволяє здійснювати шифрування та дешифрування великих обсягів інформації в режимі реального часу.

Оптимізація обчислень: Апаратні засоби повинні бути оптимізовані для виконання обчислювальних процедур алгоритму Rijndael, включаючи використання апаратних прискорювачів та паралельну обробку даних.

Енергоефективність: Апаратні засоби повинні мати низьке енергоспоживання, що є важливим для мобільних пристроїв та систем з обмеженими ресурсами.

2.6.2 Безпека та надійність

Фізичний захист: Апаратні засоби повинні забезпечувати фізичний захист криптографічних ключів та оброблюваних даних від несанкціонованого доступу та витоків інформації.

Надійність: Апаратні засоби повинні бути надійними і забезпечувати безперервну роботу без збоїв та втрат даних. Це включає захист від апаратних несправностей та механізми відновлення після збоїв.

Гнучкість та масштабованість: Апаратні засоби повинні бути гнучкими для налаштування під конкретні вимоги системи та мати можливість масштабування для обробки великих обсягів даних.

Висновки до розділу 2

У рамках цього розділу була сформульована задача дослідження, визначені основні цілі та критерії ефективності, описані вхідні та вихідні дані алгоритму, а також розглянуті обмеження та припущення, що лежать в основі розробки. Визначення цих аспектів дозволяє чітко окреслити напрямки подальших досліджень і розробок, спрямованих на створення ефективних засобів реалізації алгоритму Rijndael для захисту інформації в комп'ютерних системах та мережах.

Були визначені та проаналізовані вимоги до програмних та апаратних засобів реалізації алгоритму шифрування Rijndael. Ці вимоги включають як функціональні, так і нефункціональні аспекти, що забезпечують коректну, надійну та безпечну роботу системи. Розглянуті вимоги дозволяють сформувати чітке уявлення про необхідні характеристики програмних та апаратних засобів для ефективної реалізації алгоритму шифрування, що є ключовим для подальшої розробки та впровадження системи захисту інформації.

РОЗДІЛ 3. АЛГОРИТМ ШИФРУВАННЯ RIJNDAEL

3.1 Обчислювальні процедури алгоритму Rijndael

Алгоритм шифрування Rijndael, також відомий як AES (Advanced Encryption Standard), є основою для багатьох сучасних криптографічних систем впродовж довгого часу. Його обчислювальні процедури базуються на ряді математичних операцій, які забезпечують високу стійкість до криптоаналізу та ефективність при реалізації на різних апаратних та програмних платформах.

Основною структурою Rijndael є обробка даних у блоках по 128 бітів. Ключ може мати довжину 128, 192 або 256 бітів. Алгоритм складається з кількох раундів, кількість яких залежить від довжини ключа: 10 раундів для ключа 128 бітів, 12 раундів для ключа 192 бітів та 14 раундів для ключа 256 бітів.

Пряме та зворотне байтове заміщення:

Однією з ключових операцій є байтове заміщення (SubBytes). Вона базується на використанні нелінійних булевих функцій та здійснюється за допомогою таблиці підстановок (S-box). Ця таблиця формується на основі обчислення мультиплікативної інверсії у полі Галуа GF(2^8), а також подальшого афінного перетворення. Кожен байт вихідного блоку замінюється на відповідний байт із таблиці S-box, що забезпечує дифузію і заплутування вхідних даних [2].

Зворотне байтове заміщення (InvSubBytes) здійснюється за допомогою зворотної таблиці підстановок (InvS-box), яка також формується на основі мультиплікативної інверсії у полі Галуа GF(2^8) та зворотного афінного перетворення.

Зсув рядків:

Операція зсуву рядків (ShiftRows) забезпечує перемішування байтів у межах рядків матриці стану. Перший рядок залишається без змін, другий рядок зсувається на один байт вліво, третій рядок – на два байти, а четвертий – на три байти. Ця операція сприяє збільшенню дифузії даних по всьому блоку.

Перемішування стовпців:

Операція перемішування стовпців (MixColumns) здійснює лінійне перетворення кожного стовпця матриці стану. Вона базується на множенні вектора-стовпця на фіксований поліном у полі Галуа GF(2^8). Формально це можна записати як:

$$NewColumn = Matrix \times OldColumn \tag{3.1}$$

де матриця має вигляд:

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \tag{3.2}$$

Це перетворення забезпечує високий рівень дифузії, оскільки кожен байт вихідного стовпця залежить від усіх байтів вхідного стовпця [2].

Додавання ключа:

Додавання ключа (AddRoundKey) є найпростішою, але водночас важливою операцією. Вона полягає в побітовому додаванні за модулем 2 (операція XOR) між матрицею стану та раундовим ключем. Ця операція забезпечує змішування вхідних даних з ключем, що ускладнює їх аналіз без знання ключа [8].

Формування раундових ключів:

Формування раундових ключів (Key Expansion) базується на початковому ключі та його подальшій обробці для отримання необхідної

кількості ключів для кожного раунду. Цей процес включає використання нелінійного байтового заміщення, циклічного зсуву, додавання констант раунду (Rcon) та побітового додавання.

Раундові ключі генеруються шляхом послідовного застосування операцій до попередніх частин ключа:

$$W[i] = W[i - Nk] \oplus SubWord\big(RotWord(W[i-1])\big) \oplus Rcon[i/Nk] \quad (3.3)$$

де $W[i]$ – поточний ключовий сегмент, Nk – кількість 32-бітових слів у початковому ключі, $SubWord$ – функція байтового заміщення, $RotWord$ – циклічний зсув слів, Rcon – константа раунду.

Таким чином, алгоритм Rijndael використовує комбінацію нелінійних та лінійних перетворень, забезпечуючи високу стійкість до атак та ефективність реалізації на різних апаратних платформах. Ці обчислювальні процедури становлять основу алгоритму і гарантують його надійність та безпеку в сучасних системах захисту інформації.

3.2. Оптимізація обчислювальних процедур

3.2.1. Діагональна організація обчислення

Оптимізація обчислювальних процедур алгоритму Rijndael є важливим кроком для підвищення продуктивності шифрування, особливо при реалізації на апаратному рівні. Одним із підходів до оптимізації є діагональна організація обчислень.

Діагональна організація обчислення дозволяє уникнути операцій зсуву рядків матриці стану, що зменшує кількість необхідних операцій та спрощує реалізацію алгоритму. Замість цього, обчислення виконуються над діагоналями матриці, що дозволяє значно збільшити паралелізм та скоротити час виконання алгоритму.

У традиційній організації обчислення алгоритму Rijndael операція зсуву рядків (ShiftRows) змінює розташування байтів у матриці стану. Проте, використовуючи діагональну організацію, можна обробляти діагоналі матриці окремо, що дозволяє виконувати обчислення паралельно для кожної діагоналі. Це суттєво підвищує ефективність обчислень на багатоядерних процесорах та спеціалізованих апаратних засобах.

Основна ідея діагональної організації полягає в тому, що кожна діагональ матриці стану обробляється незалежно від інших. Наприклад, якщо позначити елементи матриці стану як $a_{i,j}$, де i та j – індекси рядка та стовпця відповідно, то діагоналі можуть бути визначені як набори елементів $a_{i,(j+k) \bmod 4}$, де k – константа, яка визначає зміщення діагоналі. Обробка таких діагоналей дозволяє уникнути додаткових операцій зсуву та перестановок, що зменшує загальну кількість обчислень [9].

На рис. 3.1 показано приклад діагональної організації обчислення для матриці стану розміром 4x4. Де діагоналі $D_1 = \{a_{00}, a_{11}, a_{22}, a_{33}\}$, $D_2 = \{a_{10}, a_{21}, a_{32}, a_{03}\}$, $D_3 = \{a_{20}, a_{31}, a_{02}, a_{13}\}$, $D_4 = \{a_{30}, a_{01}, a_{12}, a_{23}\}$. Як видно, кожна діагональ обробляється окремо, що дозволяє виконувати обчислення паралельно та уникати зайвих операцій зсуву рядків.

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Рисунок 3.1 – Діагональна організація матриці 4x4

Математично, операції з діагоналями можна описати наступним чином. Якщо S – матриця стану, то її діагоналі визначаються як:

$$D_i = \{S_{i,i+k \bmod 4} | k = 0,1,2,3\} \quad (3.4)$$

де D_i – діагональ, що починається з елементу $S_{i,i}$.

Операція зсуву рядків може бути виключена, якщо лінійне перетворення стовпців замінити на лінійне перетворення діагоналей (DM-Diagonal Mix) зі збереженням результату в стовпцях нової матриці. Враховуючи це, обчислювальна процедура Rijndael складається з 10 ідентичних циклів та фінального перетворення, як показано на рисунку 3.2.

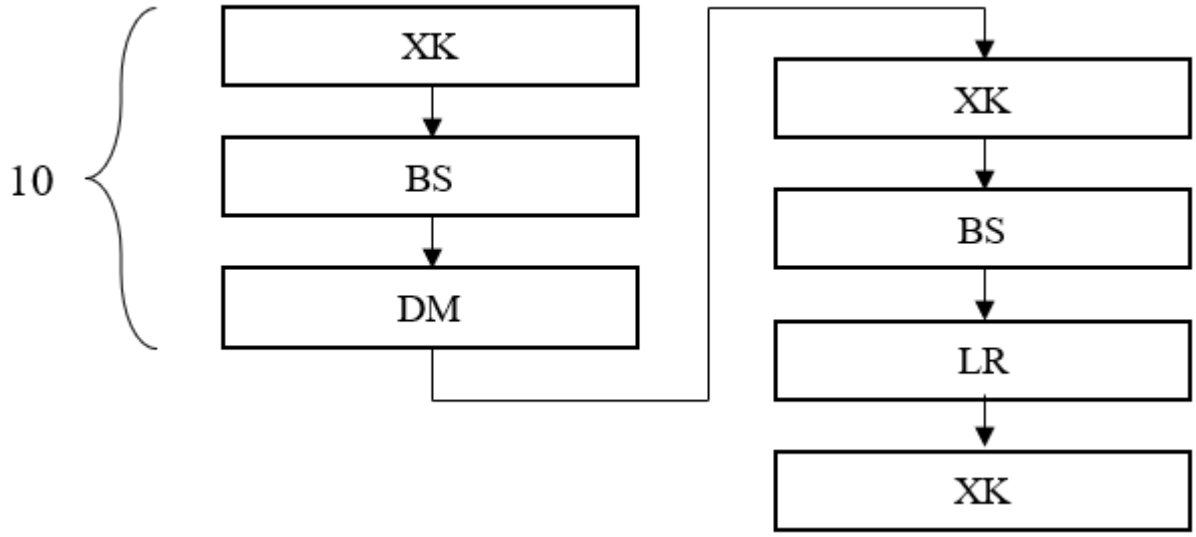


Рисунок 3.2 – Діагональна організаця виконання алгоритму

Цей підхід забезпечує ефективність за рахунок виключення операції зсуву рядків, що знижує складність реалізації алгоритму на апаратному рівні. Лінійне перетворення діагоналей дозволяє оптимізувати процес обробки даних, зберігаючи при цьому цілісність криптографічних властивостей алгоритму Rijndael. Таким чином, заміна зсуву рядків на перетворення діагоналей не тільки спрощує структуру обчислень, але й сприяє зменшенню часу виконання кожного циклу.

Якщо позначити матрицю даних як $X = ||x_{i,j}||$, *i*=0..3, *l*=0..4, то операція лінійного перетворення байтів j-тої діагоналі $D_j = \{x_{0,j}, x_{2,(j+1) \bmod 4},$

$x_{2,(j+2)\bmod 4}$, $x_{3,(j+3)\bmod 4}\}$ в j-тий стовпчик здійснюється відповідно до наступного виразу:

$$\begin{aligned} y_{0j} &= 2x_{0,j} \oplus 3x_{1,(j+1)mod4} \oplus x_{2,(j+2)mod4} \oplus x_{3,(j+3)mod4} \\ y_{1j} &= x_{0,(j+1)mod4} \oplus 2x_{1,(j+2)mod4} \oplus 3x_{2,(j+3)mod4} \oplus x_{3,j} \\ y_{2j} &= x_{0,(j+2)mod4} \oplus x_{1,(j+3)mod4} \oplus 2x_{2,j} \oplus 3x_{3,(j+1)mod4} \\ y_{3j} &= 3x_{0,(j+3)mod4} \oplus x_{1,j} \oplus x_{2,(j+1)mod4} \oplus 2x_{3,(j+2)mod4} \end{aligned} \quad (3.5)$$

при цьому операції множення 8-розрядних кодів виконуються за правилами множення у полях Галуа з утворюючим поліномом $x^8 + x^4 + x^2 + x + 1$.

Таким чином, кожен з циклів складається з 4 незалежних блоків обробки діагоналей. Кожен блок включає операції, що виконуються над 4 байтами діагоналі матриці. Структура операцій блоку обробки діагоналі показана на рисунку 3.3. Операції множення байта на 2 та 3 позначені як M2 та M3 відповідно.

Фінальні операції XOR реалізують лінійне перетворення, яке, відповідно до (3.5), включає додавання по модулю 2 результату байтового заміщення одного байта діагоналі, результату множення на 2 іншого байта діагоналі і результату множення на 3 останнього байта діагоналі. Стрілки на рисунку 3.3 вказують на те, що для виконання цієї операції необхідні результати обробки інших байтів діагоналі, зазначені вище.

Кожна діагональ в рамках одного циклу алгоритму Rijndael обробляється незалежно від інших. Лише під час переходу до наступного циклу виникає необхідність в обміні результатами. Логіка залежностей по даним двох послідовних циклів продемонстрована на рисунку 3.3.

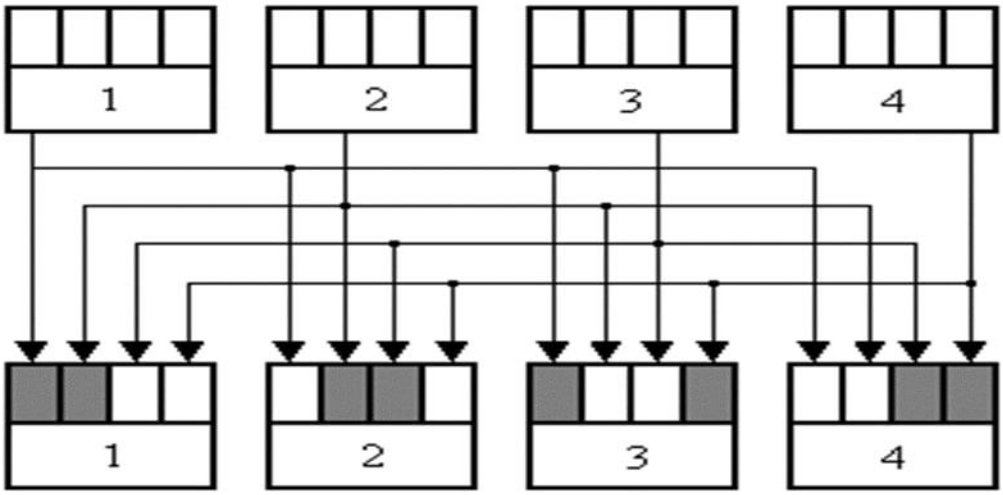


Рисунок 3.3 – Залежності даних двох послідовних циклів

Як показано на рисунку 3.3, коли обробка діагоналі одного блоку в j-му циклі алгоритму завершена, можна починати обробку одного байта кожної діагоналі наступного (j+1)-го циклу. Це дозволяє паралельно обробляти блоки j-го та (j+1)-го циклів. Однак, діагоналі (j+1)-го циклу не можуть бути повністю оброблені до завершення обробки всіх діагоналей попереднього циклу. Отже, паралельна обробка обмежується двома циклами алгоритму одночасно.

Дослідження вказують на те, що найефективнішим підходом є використання секцій, які можуть обробляти два байти діагоналі. Після завершення обробки двох байтів діагоналі j-го циклу можна починати обробку двох байтів наступного (j+1)-го циклу. Для реалізації цього підходу необхідні два типи програмних секцій: стартові та фінішні.

Стартові секції виконують перші п’ять операцій обробки двох байтів (всього 10 операцій). Фінішні секції виконують решту 18 операцій. Існує 6 варіантів стартових секцій (відповідно до кількості способів вибору пари байтів із 4-х) та 6 варіантів фінішних секцій. Завдяки перестановкам незалежних команд, для кожної з 6 стартових секцій можна створити 140 варіантів, а для кожної з 6 фінішних секцій – 4352 варіанти.

3.3.2. Використання передобчислень

Іншим важливим методом оптимізації є використання передобчислень. Цей підхід дозволяє значно зменшити кількість необхідних операцій під час виконання алгоритму, зберігаючи результати часто використовуваних обчислень у вигляді таблиць.

Операція байтового заміщення, що представляє собою систему нелінійних логічних функцій з 8 змінними, використовує таблицю перетворення, побудовану на 256 байтах (8×8 біт) постійної пам'яті, що забезпечує високу швидкість обчислень, але структура ПЛІС вимагає значного обсягу внутрішньої пам'яті. Для роботи знадобляться 2 таблиці для реалізації прямого та зворотного перетворень. З цього аналізу випливає, що апаратна реалізація прямої та зворотної заміни байтів може бути виконана за допомогою однієї таблиці для мультиплікативної інверсії, тоді як множення матриць та додавання за вектором B можуть бути реалізовані за допомогою логічної схеми. Альтернативою табличному обчисленню мультиплікативної інверсії є рекурсивне обчислення з використанням логічних схем і регістрів, основним недоліком якого є тривалий час виконання операції [10].

Операція перемішування байтів стовпців матриці стану здійснює лінійне перетворення, де кожен із 32 бітів зміненого стовпця є сумою по модулю 2 біта вихідного стовпця. Наприклад, молодший розряд байта $y_0 - y_0^0$ можна записати як:

$$y_0^0 = x_0^7 \oplus x_1^7 \oplus x_1^0 \oplus x_2^0 \oplus x_3^0 \qquad (3.6)$$

Інший підхід до апаратної реалізації матричного множення полягає у використанні табличних функціональних перетворювачів. У цьому випадку рекомендується поєднати реалізацію заміни байтів і перетасовку стовпців на табличному рівні. Зазвичай така комбінація реалізується у вигляді так званих

Т-таблиць, організованих у форматі 258×32 байти. Для цього потрібні чотири такі таблиці, що займають загалом 4 Кбайта пам'яті [10].

Кожна j-та таблиця з цих чотирьох таблиць містить значення 32-розрядного коду T, який включає байт x, замінений на 32-розрядний код перемішаного рядка. Таким чином, перетворений k-й стовпець формується як сума по модулю двійки чотирьох 32-розрядних табличних кодів:

$$T[\mathrm{x}_{0\mathrm{k}}] \oplus T[\mathrm{x}_{1\mathrm{k}}] \oplus T[\mathrm{x}_{2\mathrm{k}}] \oplus T[\mathrm{x}_{3\mathrm{k}}] \quad (3.7)$$

Використання Т-таблиць в алгоритмі Rijndael дозволяє ефективно реалізувати його на 32-розрядному універсальному процесорі без обмеження обсягу пам'яті, займаного табличними перетворювачами. Однак, при реалізації на програмованих мікросхемах, значний обсяг пам'яті, необхідний для зберігання Т-таблиць, стає суттєвим недоліком, що ускладнює реалізацію в структурі ПЛІС.

Подібний підхід використовується і для операції перемішування стовпців (MixColumns). Замість виконання множення в полі Галуа під час кожного раунду шифрування, можна використовувати таблиці, що містять результати множення для всіх можливих значень байтів. Ці таблиці, відомі як Т-таблиці, дозволяють об'єднати кілька операцій у одну, значно зменшуючи кількість необхідних обчислень [10].

Передобчислення також можуть бути використані для генерації раундових ключів. Замість обчислення раундових ключів у режимі реального часу, можна попередньо згенерувати всі необхідні ключі та зберігати їх у пам'яті для швидкого доступу під час шифрування та дешифрування.

Математично, Т-таблиці можна описати наступним чином. Якщо M - матриця множення для операції MixColumns, то для кожного байту b можна створити таблицю $\mathrm{T_b}$:

$$\mathrm{T_b}[i] = M \times i \quad (3.8)$$

де і - всі можливі значення байтів.

Використання передобчислень дозволяє значно підвищити продуктивність алгоритму Rijndael, особливо на обмежених у ресурсах пристроях, таких як вбудовані системи та мобільні пристрої. Цей підхід є одним з найефективніших методів оптимізації, що дозволяє досягти високої швидкості виконання без значних втрат у безпеці [10].

Висновки до розділу 3

В результаті проведеного аналізу алгоритму шифрування Rijndael та дослідження методів оптимізації обчислювальних процедур, було досягнуто таких висновків:

- Алгоритм Rijndael, який є основою стандарту AES, забезпечує високий рівень криптографічної безпеки завдяки використанню складних нелінійних перетворень та операцій у полях Галуа. Його ефективність підтверджена численними дослідженнями та практичним застосуванням у різних галузях.
- Діагональна організація обчислення дозволяє значно зменшити кількість необхідних операцій зсуву рядків матриці стану, що підвищує продуктивність алгоритму на апаратному рівні. Цей підхід забезпечує більший рівень паралелізму та скорочує час виконання обчислень.
- Використання передобчислень, зокрема Т-таблиць, є ефективним методом оптимізації алгоритму Rijndael. Цей підхід дозволяє значно зменшити кількість необхідних обчислень під час виконання алгоритму, що підвищує його швидкодію та продуктивність. Збереження попередньо обчислених результатів у вигляді таблиць дозволяє швидко виконувати складні операції, такі як заміна байтів та перемішування стовпців.
- Поєднання методів діагональної організації обчислень та використання передобчислень дозволяє досягти високої ефективності алгоритму Rijndael при його реалізації як на програмному, так і на апаратному рівнях. Це забезпечує можливість використання алгоритму в широкому спектрі додатків, включаючи вбудовані системи та мобільні пристрої, де ресурси обмежені.
- Оптимізація алгоритму Rijndael шляхом використання передобчислень та діагональної організації обчислень є перспективним напрямком для

подальших досліджень і вдосконалення. Ці методи можуть бути використані для розробки нових криптографічних алгоритмів та підвищення ефективності існуючих.

Таким чином, результати проведеного дослідження підтверджують доцільність використання методів оптимізації обчислювальних процедур для підвищення продуктивності алгоритму Rijndael, що є важливим кроком для забезпечення ефективного та надійного криптографічного захисту даних.

РОЗДІЛ 4. РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ АЛГОРИТМУ RIJNDAEL

4.1. Структура програми та організація даних

Програму для реалізації алгоритму шифрування Rijndael було написано мовою C++ у середовищі Microsoft Visual Studio 2019. Особлива увага приділялася трансляції програм із мови C++ на машинний код та оптимізації обчислень на сучасних комп'ютерах для досягнення максимальної продуктивності алгоритму.

Програма складається з шести основних модулів:

1. Cipher: відповідає за створення, виконання та завершення програми в середовищі Windows.
2. CipherDlg: забезпечує взаємодію користувача з програмою.
3. Rijndael: реалізує основний алгоритм шифрування та дешифрування Rijndael.
4. GenTables: генерує таблиці для прискорення роботи алгоритму.
5. GenTab: також генерує таблиці для оптимізації обчислень.
6. AllTabl: зберігає всі згенеровані таблиці.

Модуль Cipher містить процедури для створення, виконання та завершення програми. CipherDlg відповідає за інтерфейс користувача. Модуль Rijndael реалізує алгоритм шифрування та дешифрування. GenTables і GenTab відповідають за генерацію таблиць, необхідних для оптимізації обчислень, а AllTabl зберігає ці таблиці.

Процедура Initialisation ініціалізує вихідні дані, ключі та таблиці для швидкого виконання алгоритму. KeyExpansion генерує розширений ключ для шифрування з початкового ключа довжиною 128, 192 або 256 біт. KeyInversion перетворює розширений ключ для шифрування на ключ для дешифрування.

Процедури Round та inverseRound реалізують основні операції злиття та зворотного злиття в алгоритмі шифрування та дешифрування відповідно. FinalRound та inverseFinalRound виконують завершальні операції злиття та зворотного злиття. AddRoundKey поєднує розширений ключ із проміжними даними на різних етапах алгоритму.

Структура програми забезпечує високу модульність і можливість легкого розширення функціональності. Чітке розділення модулів за функціональністю дозволяє ефективно тестувати та відлагоджувати програму, забезпечуючи високу надійність і масштабованість системи.

4.2. Функції та методи програми

У процедурах Round, inverseRound, FinalRound і inverseFinalRound замість побайтових операцій використовуються 32-бітові операції з подвійними словами. Для подальшого підвищення продуктивності можна застосовувати 64-бітові операції з учетвереними словами, проте це вимагатиме використання процесорів не нижче п'ятого покоління, обмежуючи сумісність програми.

Збільшення розрядності також впливає на розмір таблиць. Наприклад, підстановка T-таблиць, що замінює операції підстановки S-box, зсуву рядків і змішування колонок, здійснюється шляхом додавання 32-бітових аргументів за модулем 2, отриманих із попередньо обчислених байтових масивів:

```
e[0] = ((dword*)T0)[*(a + 4 * 0 + 0)]
^ ((dword*)T1)[*(a + 4 * 1 + 1)]
^ ((dword*)T2)[*(a + 4 * 2 + 2)]
^ ((dword*)T3)[*(a + 4 * 3 + 3)];
```

При реалізації 64-бітових обчислень можливо виконувати одночасно обчислення двох подвійних слів або використовувати конвеєрний метод, при якому перше подвійне слово обробляється для і-го подвійного слова k-го

блоку даних, а друге - для і-го подвійного слова (k+1)-го блоку даних. У першому випадку необхідні таблиці T обсягом 512 Кбайт, у другому - додаткове пересилання при кожній підстановці з T-таблиць. Приклад удосконалення можна реалізувати наступним чином:

```
e[0] = ((qword[256][256]*)T0)[*(a + 4 * 0 + 0)][*(a + 4 * 1 + 0)] ^
((qword*)T1[256][256])[*(a + 4 * 1 + 1)][*(a + 4 * 2 + 1)] ^
((qword*)T2[256][256])[*(a + 4 * 2 + 2)][*(a + 4 * 3 + 2)] ^
((qword*)T3[256][256])[*(a + 4 * 3 + 3)][*(a + 4 * 0 + 3)];
```

Функції GenerateRejTableS і GenerateRejTableSi генерують прямі та зворотні таблиці «S-box», які замінюють операції множення та обчислення зворотного в полях Галуа GF(2^8). GenerateRejTableT і GenerateRejTableTi створюють прямі та зворотні таблиці T, що замінюють операції змішування колонок і підстановки таблиць «S-box». Функція GenerateRejTableMxi генерує таблиці Mxi, що використовуються для отримання зворотного розширеного ключа, замінюючи операцію змішування колонок.

Генерація таблиць здійснюється за допомогою функції xtime(), яка виконує множення на 2 у полях Галуа GF(2^8), реалізованої на асемблері для максимальної ефективності. Наприклад:

```
byte xtime(byte x) {
   return (x & 0x80) ? ((x << 1) ^ 0x1B) : (x << 1);
}
```

Процес генерації таблиць включає послідовне множення елементів масивів S[Tidx] і MixTabl[j] (або invMixT[j] для зворотних таблиць) у полях Галуа GF(2^8):

```
byte Multipl(byte a, byte b) {
   byte p = 0;
   byte counter;
   byte hi_bit_set;
   for (counter = 0; counter < 8; counter++) {
      if (b & 1) p ^= a;
      hi_bit_set = (a & 0x80);
      a <<= 1;
      if (hi_bit_set) a ^= 0x1B;
      b >>= 1;
   }
   return p;
}
```

4.3. Інструкція користувача

Після запуску програми відкривається головне вікно. Для шифрування або дешифрування файлів користувач повинен обрати вихідний та результуючий файли. Це можна зробити, ввівши шлях і ім'я файлу в текстові поля "Вихідний файл" та "Результуючий файл", або скориставшись діалоговими вікнами, які викликаються кнопкою трьокрапки поруч із відповідними текстовими полями.

Далі користувач має обрати розмір блоку даних і ключа у бітах, натиснувши на відповідні опції. Ключ може бути введений у шістнадцятковому або текстовому форматі (поле "Ключ (ASCII)"). Важливо, щоб ключ був однаковим для процесів шифрування та дешифрування.

Наступним і останнім кроком є вибір дії, натиснувши кнопку "Шифрувати" або "Розшифрувати".

Після вибору файлів і налаштувань користувач натискає "Шифрувати" для шифрування або "Розшифрувати" для дешифрування файлу. Програма зчитує дані з вихідного файлу, обробляє їх відповідно до вибраного алгоритму і записує результати у результуючий файл. Після завершення операції користувач отримає повідомлення про успішне виконання процесу.

Системні вимоги:

- Операційна система: Windows (Win32), з можливістю перекомпіляції для інших ОС.
- Процесор: 386 і вище.
- Оперативна пам'ять: мінімум 4 МБ.Вільне місце на жорсткому диску: 1 МБ (5 МБ за відсутності бібліотек MFC та компонентів ActiveX).

Обмеження програми:

- Підтримка файлів: до 4 ГБ (залежить від можливостей ОС).
- Імена файлів: до 256 символів.
- Розрядність ключа: 128, 192 або 256 біт.
- Розміри блоків даних: 128, 192 або 256 біт.

Висновки до розділу 4

У результаті виконаної роботи було створено програму для реалізації алгоритму симетричного шифрування Rijndael з використанням попередньо обчислених таблиць та діагональної організації алгоритму. В описі надано структуру даних, що використовуються в програмі, та принципи роботи основних функцій, які реалізують цей алгоритм. Також було надано інструкції користувача для роботи з інтерфейсом програми. Аналіз результатів експериментального застосування підтвердив ефективність розробленого програмного продукту.

Розроблена програма відзначається високою продуктивністю та надійністю, які досягаються за рахунок оптимізації обчислювальних процесів і використання попередньо обчислених таблиць. Впровадження цієї програми значно підвищує ефективність процесу шифрування та дешифрування даних у різних комп'ютерних системах та мережах, забезпечуючи високий рівень безпеки інформації.

РОЗДІЛ 5. АПАРАТНА РЕАЛІЗАЦІЯ АЛГОРИТМУ RIJNDAEL

5.1. Структура апаратних засобів

Захист інформації в сучасних комп'ютерних системах та мережах є надзвичайно важливою задачею, і криптографічні алгоритми відіграють ключову роль у забезпеченні цієї безпеки. Однією з ефективних стратегій підвищення продуктивності криптографічних систем є використання спеціалізованих апаратних засобів для реалізації шифрувальних алгоритмів. Такий підхід дозволяє значно підвищити швидкість шифрування даних порівняно з програмною реалізацією.

Апаратна реалізація алгоритму Rijndael здійснюється за допомогою криптопроцесорів, які забезпечують високу швидкість обробки даних та підвищену безпеку. Криптопроцесор може бути вбудований у різні пристрої, такі як мережеві маршрутизатори, сервери, мобільні пристрої та інші апаратні платформи. Структура криптопроцесора включає кілька основних компонентів:

- Блок керування: відповідає за загальне управління криптопроцесором, включаючи ініціалізацію, контроль виконання операцій та синхронізацію роботи всіх інших компонентів.
- Блок ключів: зберігає основний та розширені ключі, необхідні для шифрування та дешифрування.
- Блок пам'яті даних: забезпечує зберігання вхідних та вихідних даних, а також проміжних результатів.
- Блок обчислень: реалізує основні криптографічні перетворення, такі як підстановка байтів, зсув рядків, перемішування стовпців та додавання ключів.

Дипломний проект присвячений підвищенню продуктивності апаратних і програмних реалізацій симетричного алгоритму шифрування AES. Було проаналізовано обчислювальні процедури, що лежать в основі алгоритму AES. В результаті аналізу запропоновано підходи до прискорення реалізації алгоритму AES. Зокрема, досліджено можливості використання табличних обчислень із попередніми обчисленнями. Цей метод можна застосовувати на рівні перетворення байтів, де кілька байтів матриці стану можуть оброблятися одночасно шляхом комбінування таблиць.

Ще однією можливістю прискорення виконання криптографічного алгоритму AES, як в апаратних, так і в програмних реалізаціях, є використання діагональної обробки матриці стану.

Розроблено структуру високопродуктивної апаратної реалізації алгоритму AES. Також створено програмну реалізацію алгоритму AES.

Після отримання команди на генерацію ключа керуючий блок криптографічного процесора ініціює цикл генерації ключа, що складається з 68 32-бітових ключових кодів. Процес генерації ключа включає два основних етапи. На першому етапі до процесора шифрування вводяться шість 32-бітових кодів через системну адресну шину, що утворюють початковий ключ. На другому етапі ці коди розширюються для отримання повного набору робочих ключів.

Керування криптографічним процесором здійснюється через кілька блоків алгоритму. Спочатку блок 3 встановлює лічильник адрес пам'яті ключа на нуль, а блок 4 сигналізує про готовність криптографічного процесора прийняти ключ через системну шину даних. Коли надходить сигнал на видачу коду ключа по системній контрольній шині базового комп'ютера (керованій блоком 5), блок 6 передає 32-бітовий код ключа в пам'ять через вхідний приймач, після чого відбувається інкремент.

Якщо блок 5 не виявляє вихідного сигналу від центрального процесора, він повертається до блоку 4 для повторної спроби прийняття ключа. Після інкременту адреси пам'яті ключа, блок 12 перевіряє, чи дорівнює код адреси пам'яті ключа шести. Якщо адресний код менший за шість, блок 4 виконується знову для отримання наступного набору 32-бітових кодів ключа. Якщо адресний код пам'яті ключа дорівнює шести, починається другий етап процесу генерації ключа.

На другому етапі блок 7 зчитує код ключа з блоку пам'яті, адреса якого на чотири менша за поточну адресу, і записує його в блокувальний регістр нелінійної трансформації. Наступний блок, блок 8, використовує табличний перетворювач для виконання нелінійної (прямої) трансформації 32-бітового коду ключа і записує його в пам'ять ключів за поточною адресою. Блок 9 починає зчитувати попередній 32-бітовий код ключа з пам'яті ключів, блок 10 зчитує поточний 32-бітовий код ключа, який є результатом трансформації з блокувального регістра, а наступний блок, блок 11, використовує постійний код, зчитаний з пам'яті мікрокоманд керуючого блоку, і проміжний код з блокувального регістра. Код, отриманий в результаті операції за модулем 2, є кодом розширення ключа і записується в пам'ять ключів (блок 13). Для генерації наступного коду розширення ключа той самий блок 13 інкрементує поточну адресу пам'яті ключів, і блок 14 перевіряє, чи дорівнює код 68. Якщо контрольний код менший за 68, починаючи з блоку 7, знову вводиться наступний 32-бітовий код розширення ключа.

Використання апаратних засобів дозволяє забезпечити високу швидкість шифрування даних навіть при великих обсягах інформації. Це особливо важливо в контексті сучасних комп'ютерних мереж, де вимоги до продуктивності постійно зростають. Наприклад, у мережах передачі даних, де необхідне швидке шифрування потоків даних, програмні рішення можуть бути недостатніми, і апаратні засоби стають критично важливими.

Основною проблемою при шифруванні даних у комп'ютерних мережах є недостатня швидкість шифрування потоків даних за допомогою програмних засобів. Багато сучасних обчислювальних технологій також висувають високі вимоги до швидкості функцій криптографічного захисту. Наприклад, у програмному захисті програми зберігаються в зашифрованому вигляді, і розшифровані команди виконуються безпосередньо перед виконанням. Процес розшифрування має бути достатньо швидким, щоб не впливати на загальний час виконання програми. Деякі застосування криптографічних алгоритмів вимагають виконання шифрування на фізично окремих пристроях, що забезпечує значно вищу безпеку, запобігаючи витоку ключової інформації через спеціальні режими процесора.

Використання спеціалізованих апаратних засобів для реалізації криптографічних алгоритмів значно підвищує продуктивність і надійність шифрування даних. Наприклад, для апаратної реалізації відомого алгоритму DES було розроблено понад 20 типів спеціалізованих інтегральних схем (ASIC). Після прийняття алгоритму Rijndael як стандарту США для симетричного шифрування було запропоновано кілька архітектурних рішень для його апаратної реалізації за допомогою бібліотек ASIC та FPGA.

Основні труднощі в апаратній реалізації Rijndael пов'язані з тим, що процеси шифрування та дешифрування реалізуються у зворотному порядку і використовують різні базові перетворення. Це призводить до необхідності окремих апаратних засобів для шифрування та дешифрування. Тому важливо знайти рішення, які дозволяють використовувати ті самі апаратні компоненти для обох процесів. Крім того, на відміну від більшості симетричних алгоритмів, які обробляють лише половину бітів блоку даних, Rijndael обробляє весь блок даних одночасно.

Найбільш підходящим рішенням для цього завдання є використання сучасних криптопроцесорів, які масово виробляються багатьма зарубіжними

компаніями. Найбільш поширеним криптопроцесором сьогодні є IBM 1752, який може реалізувати алгоритми DES, RSA, El-Gamal та DSA, але не підтримує алгоритми сімейства AES. Останніми роками в США було розроблено більш просунуті криптопроцесори, зокрема компанією Hi/fn, яка з 2009 року масово випускає потужні криптопроцесори сімейства 6500. Ці криптопроцесори можуть виконувати завдання модульної арифметики та генерувати випадкові числа.

Процесор 6500 є видатним прикладом мікропроцесорної технології, з шістнадцятьма робочими регістрами та арифметично-логічним пристроєм (ALU) шириною 1024 біти. Він оснащений шістьма внутрішніми шинами тієї ж ширини, які працюють на тактовій частоті 100 МГц. Генератор випадкових чисел у цьому процесорі використовує інноваційний підхід, заснований на порівнянні фаз сигналів від стабільного тактового генератора та нестабільного автономного генератора для формування випадкового числа.

Набір інструкцій процесора 6500 простий, але ефективний. Кожна інструкція може використовувати чотири довільні регістри даних: регістр результату (RR), два регістри операндів (OR) та регістр значення модуля (MVR). Усі інструкції є арифметичними і виконуються за модулем значення в MVR, виконуючи обчислення в чотири цикли. Процесор також включає інтерфейс шини вводу-виводу та вбудований мікроконтролер, що дозволяє записувати та виконувати короткі програми з 32 інструкцій швидко. Це робить 6500 ідеальним для використання в Інтернет-орієнтованих продуктах, де він підтримує до 300 зашифрованих запитів сервера на секунду і виконує понад 80 операцій аутентифікації з 1024-бітовими ключами на секунду.

5.2. Алгоритм функціонування криптопроцесора

Розроблений криптопроцесор здатний працювати у декількох режимах, включаючи прийом і формування коду ключа, шифрування та дешифрування

блоків інформаційних повідомлень. Основні етапи роботи криптопроцесора включають:

1. Ініціалізація: Після включення живлення криптопроцесор виконує самотестування та ініціалізацію всіх необхідних регістрів та пам'яті. У разі успішного завершення ініціалізації процесор видає сигнал готовності до роботи.
2. Формування ключа: Після отримання команди формування ключа від центрального процесора, блок керування криптопроцесора ініціює цикл генерації робочого блоку ключів. Цей процес складається з двох етапів: завантаження початкового ключа та формування розширених ключів.
3. Прийом даних: Криптопроцесор очікує на команду обробки інформаційного блоку. Після отримання команди, він зчитує 192-розрядний блок даних з системної шини і записує його у власну пам'ять.
4. Шифрування/дешифрування: Залежно від типу операції (шифрування або дешифрування), криптопроцесор виконує необхідні перетворення над блоком даних. Алгоритм Rijndael включає послідовні раунди обчислень, що включають підстановку байтів, зсув рядків, перемішування стовпців та додавання ключів.
5. Видача результату: Після завершення шифрування або дешифрування криптопроцесор передає оброблений блок даних назад на системну шину для подальшого використання або зберігання.

Кожен із зазначених етапів включає виконання специфічних операцій, що потребують високої точності та швидкості обробки даних. Апаратна реалізація алгоритму Rijndael дозволяє досягти значного підвищення продуктивності завдяки паралельному виконанню обчислень та використанню спеціалізованих обчислювальних блоків.

При відсутності команд від центрального процесора, криптопроцесор видає сигнал готовності до роботи.

5.2.1. Процес генерації ключів

Після отримання команди на генерацію ключа, керуючий блок криптографічного процесора запускає цикл створення ключа, який включає 68 32-бітових ключових кодів. Процес генерації ключа складається з двох основних етапів. На першому етапі до процесора шифрування вводяться шість 32-бітових кодів через системну адресну шину, утворюючи початковий ключ. На другому етапі ці коди розширюються для отримання повного набору робочих ключів.

Керування криптографічним процесором здійснюється через декілька блоків алгоритму. Спочатку блок 3 встановлює лічильник адрес пам'яті ключа на нуль, а блок 4 сигналізує про готовність криптографічного процесора прийняти ключ через системну шину даних. Коли надходить сигнал на видачу коду ключа по системній контрольній шині базового комп'ютера (керованій блоком 5), блок 6 передає 32-бітовий код ключа в пам'ять через вхідний приймач, після чого відбувається інкремент.

Якщо блок 5 не виявляє вихідного сигналу від центрального процесора, він повертається до блоку 4 для повторної спроби прийняття ключа. Після інкременту адреси пам'яті ключа, блок 12 перевіряє, чи дорівнює код адреси пам'яті ключа шести. Якщо адресний код менший за шість, блок 4 виконується знову для отримання наступного набору 32-бітових кодів ключа. Якщо адресний код пам'яті ключа дорівнює шести, починається другий етап процесу генерації ключа.

На другому етапі блок 7 зчитує код ключа з блоку пам'яті, адреса якого на чотири менша за поточну адресу, і записує його в блокувальний регістр нелінійної трансформації. Наступний блок, блок 8, використовує табличний перетворювач для виконання нелінійної (прямої) трансформації 32-бітового коду ключа і записує його в пам'ять ключів за поточною адресою. Блок 9 починає зчитувати попередній 32-бітовий код ключа з пам'яті ключів, блок 10

зчитує поточний 32-бітовий код ключа, який є результатом трансформації з блокувального регістра, а наступний блок, блок 11, використовує постійний код, зчитаний з пам'яті мікрокоманд керуючого блоку і проміжний код з блокувального регістра. Код, отриманий в результаті операції за модулем 2, є кодом розширення ключа і записується в пам'ять ключів (блок 13). Для генерації наступного коду розширення ключа той самий блок 13 інкрементує поточну адресу пам'яті ключів, і блок 14 перевіряє, чи дорівнює код 68. Якщо контрольний код менший за 68, починаючи з блоку 7, знову вводиться наступний 32-бітовий код розширення ключа.

5.2.2. Режим обробки інформаційного блоку

Режим обробки інформаційного блоку включає в себе 3 основні фази:

1. Прийом даних: 192-розрядний код інформаційного блоку надходить в пам'ять даних криптографічного процесора через системну шину даних базового комп'ютера.
2. Обробка: інформаційний блок шифрується або дешифрується.
3. Вивід даних: отриманий 192-бітний код передається з пам'яті даних криптографічного процесора на системну шину даних, а потім в основну пам'ять базового комп'ютера.

5.2.3. Алгоритми криптографічного процесора

Алгоритм шифрувального процесора для шифрування або дешифрування даних містить кілька ключових блоків:

– Блок 15: підтвердження прийому команди обробки інформаційного блоку від центрального процесора.
– Блок 16: запускає обнулення поточної адреси пам'яті блоку даних.
– Блок 17: виводить сигнал підготовки криптопроцесора на системну керуючу шину базового комп'ютера.

Коли з процесора надходить закодований вихідний сигнал, блок 21 приймає 32-бітові коди даних і записує їх у пам'ять даних (пам'ять станів). Блок 22 завантажує всі 192-бітові слова з шести 32-бітових інформаційних блоків у пам'ять даних криптографічного процесора. За відсутності помилок процес продовжується. Якщо поточний адресний код даних менший за шість, процес переходить до блоку 17 і зчитує наступні 32-бітові слова інформаційного блоку. Якщо адресний код даних дорівнює шести, завершується перший етап перетворення інформації. Блок 23 визначає тип криптографічної трансформації (прямий чи зворотний). При прямій трансформації активується блок 18, і поточна адреса блоку ключової пам'яті встановлюється на нуль. У блоці 19 відбувається повторюваний цикл, де 32-бітові слова з пам'яті даних завантажуються в регістр нелінійної трансформації, а перетворений 4-байтовий 32-бітовий код зчитується з табличного перетворювача, що функціонує в прямому режимі.

Блок 24 виконує лінійну трансформацію стовпців матриці стану і повторює цикл для кожного з шести стовпців матриці. 32-бітовий код стовпця зчитується з блоку пам'яті даних і подається на вхід багатовходового суматора за модулем 2. Мультиплексор перемикає 32-бітовий код з виходу багатовходового суматора на вхід буферного регістра, звідки він записується в пам'ять даних.

Блок 25 використовує поточний код ключа з робочої області для виконання додавання за модулем 2 кожного з шести стовпців матриці стану інформаційного блоку. Це перетворення повторюється 13 разів, а завершення перетворення контролюється блоком 27, який визначає еквівалентність коду з робочої області поточній адресі ключа 68.

Під час декодування інформаційного блоку блок 28 встановлює адресу останнього (68-го) ключового блоку як поточну адресу ключа. Після цього виконується 13 циклів перетворення, аналогічних прямому перетворенню.

Блок 29 зчитує 32-бітовий код робочого ключа за модулем 2 у зворотному порядку з пам'яті ключа і код стовпця матриці стану інформаційного блоку у зворотному порядку (з п'ятого по нульовий).

Блок 30 виконує зворотну лінійну трансформацію всіх шести стовпців матриці стану інформаційного блоку. Для цього кожен з 32-бітових кодів стовпців інформаційного блоку зчитується на локальну шину даних криптографічного процесора і подається на вхід другої групи багатовходових суматорів за модулем 2. Сигнал з виходу багатовходового суматора перемикається мультиплексором на вхід буферного регістра, а потім на вхід блоку пам'яті даних.

Блок 31 виконує нелінійну трансформацію всіх байтів інформаційного блоку у повторюваному циклі. У кожному такому циклі одночасно виконується 4-байтова нелінійна трансформація, а табличний перетворювач працює у зворотному режимі (тобто використовується таблиця зворотної нелінійної трансформації). Блок 32 зменшує значення лічильника адрес пам'яті ключа після кожного циклу, а блок 33 порівнює поточний код адреси пам'яті ключа з нулем. Якщо код не дорівнює нулю, цикл зворотного перетворення не завершено, і повторне виконання здійснюється переходом до блоку 27. В іншому випадку виконується блок 34 для зчитування інформаційного блоку з блоку пам'яті даних на системну шину даних і далі в основну пам'ять базового комп'ютера.

Ці детальні блоки і алгоритми криптографічних процесорів забезпечують високу швидкість і надійність обробки даних, роблячи їх ефективним інструментом для захисту інформації у сучасних комп'ютерних системах і мережах.

Висновки до розділу 5

Результати виконаних робіт демонструють, що розроблена структурна схема системи захисту інформації, яка реалізує симетричне шифрування даних за алгоритмом Rijndael на основі криптопроцесора, є ефективним і надійним рішенням для сучасних комп'ютерних систем і мереж. Використання апаратної реалізації дозволяє значно підвищити швидкість шифрування і забезпечити високий рівень захисту даних, що робить цей підхід ефективним і надійним для сучасних систем захисту інформації.

Апаратна реалізація алгоритму Rijndael на основі криптопроцесора може бути виконана з використанням сучасних мікропроцесорних рішень, таких як IBM 1752 і Hi/fn 6500. Це дозволяє досягти високих показників продуктивності і надійності, забезпечуючи ефективний захист даних в комп'ютерних системах і мережах.

Розроблений криптопроцесор здатен працювати в режимі прийому і формування коду ключа, шифрування і дешифрування блоків даних, а також забезпечувати високу швидкість і надійність обробки інформації. Це досягається завдяки використанню сучасних мікропроцесорних рішень і ефективних алгоритмів шифрування та дешифрування.

Таким чином, апаратна реалізація алгоритму Rijndael на основі криптопроцесора є ефективним і надійним рішенням для захисту даних в сучасних комп'ютерних системах і мережах. Використання апаратних засобів дозволяє значно прискорити процес шифрування і забезпечити високий рівень захисту інформації, що робить цей підхід оптимальним для використання в умовах зростаючої інформаційної інтеграції і вимог до безпеки даних.

РОЗДІЛ 6. ПОРІВНЯЛЬНИЙ АНАЛІЗ ПРОГРАМНОЇ ТА АПАРАТНОЇ РЕАЛІЗАЦІЇ АЛГОРИТМУ RIJNDAEL

6.1. Продуктивність і ефективність

Однією з головних переваг апаратної реалізації алгоритму Rijndael є висока продуктивність. Використання спеціалізованих апаратних засобів дозволяє значно підвищити швидкість обробки даних у порівнянні з програмними реалізаціями. Апаратні криптопроцесори, такі як ті, що використовуються в маршрутизаторах і комутаторах, можуть обробляти тисячі шифрованих пакетів в секунду, що робить їх незамінними у високопродуктивних мережевих середовищах.

Крім того, апаратна реалізація дозволяє знизити навантаження на центральний процесор системи, що сприяє підвищенню загальної продуктивності. Завдяки цьому, системи можуть виконувати інші важливі завдання без значних затримок, що є критично важливим у реальному часі. Наприклад, у фінансових системах це дозволяє забезпечити швидку обробку транзакцій та мінімізувати затримки, пов'язані з шифруванням і розшифруванням даних.

Програмні реалізації алгоритму Rijndael також демонструють високу ефективність завдяки оптимізації коду та використанню передобчислень. Зокрема, використання табличних перетворень (T-таблиць) дозволяє значно прискорити процес шифрування та розшифрування, зменшуючи кількість обчислень, необхідних для виконання алгоритму. Це особливо важливо для застосувань, де швидкість обробки даних має вирішальне значення, наприклад, у клієнтах VPN або файлових шифраторах.

6.2. Безпека

Безпека є ключовим аспектом як апаратної, так і програмної реалізації алгоритму Rijndael. Апаратні криптопроцесори забезпечують високий рівень безпеки завдяки використанню спеціалізованих механізмів захисту, таких як апаратні генератори випадкових чисел та захищені сховища ключів. Це дозволяє мінімізувати ризики, пов'язані з несанкціонованим доступом та можливими атаками на систему.

Програмні реалізації також забезпечують високий рівень безпеки завдяки використанню криптографічно стійких алгоритмів і методів захисту. Зокрема, алгоритм Rijndael має стійкість до різних видів атак, таких як лінійний та диференціальний криптоаналіз, що робить його одним з найбільш надійних алгоритмів симетричного шифрування. Важливим аспектом безпеки є також регулярне оновлення програмного забезпечення та виправлення вразливостей, що дозволяє підтримувати високий рівень захисту в умовах постійно змінюваних загроз.

У фінансових системах та мобільних пристроях використання апаратних криптопроцесорів дозволяє забезпечити додатковий рівень безпеки, який важко досягти за допомогою програмних засобів. Наприклад, апаратні засоби можуть забезпечувати захист від фізичного доступу до пристрою та зберігання ключів у захищеній пам'яті, що значно ускладнює атаки на систему.

6.3. Вартість і доступність

Вартість реалізації алгоритму Rijndael також є важливим фактором при виборі між програмною та апаратною реалізацією. Програмне забезпечення зазвичай дешевше у розробці та впровадженні, оскільки не вимагає спеціалізованого обладнання. Крім того, його легко оновлювати та модифікувати відповідно до змін у вимогах безпеки або нових загроз.

Апаратна реалізація, хоча і забезпечує вищу продуктивність і безпеку, може бути значно дорожчою через вартість розробки і виготовлення спеціалізованих мікросхем. Крім того, апаратні рішення можуть бути менш гнучкими і вимагати більше часу для впровадження змін або оновлень.

6.4. Приклади застосувань

Програмна реалізація алгоритму Rijndael знаходить широке застосування у багатьох програмних рішеннях для захисту даних. Одним із найпоширеніших прикладів є використання алгоритму в клієнтах VPN (Virtual Private Network), які забезпечують захищений доступ до мережі через незахищене середовище. Програмна реалізація дозволяє адаптуватися до різних вимог безпеки та легко оновлюватися в разі появи нових загроз, що робить її ідеальною для динамічних середовищ, таких як інтернет-мережі та корпоративні системи. VPN-клієнти, що використовують алгоритм Rijndael, гарантують високий рівень конфіденційності та цілісності переданих даних, забезпечуючи надійний захист від несанкціонованого доступу та атак посередника [11].

Також програмна реалізація часто використовується у файлових шифраторах, що дозволяють користувачам безпечно зберігати свої дані на локальних або хмарних сховищах. Файлові шифратори, які використовують алгоритм Rijndael, забезпечують захист конфіденційної інформації шляхом шифрування файлів перед їх зберіганням або передачею, що робить їх недоступними для неавторизованих користувачів. Такі рішення знаходять застосування у різних сферах, від особистого використання до корпоративних систем, де безпека даних є критично важливою.

Апаратна реалізація алгоритму Rijndael, з іншого боку, має велике значення у високопродуктивних мережевих пристроях, таких як маршрутизатори та комутатори. У цих пристроях швидкість обробки даних є

критично важливою, оскільки вони повинні забезпечувати передачу великого обсягу даних з мінімальними затримками. Використання апаратних криптопроцесорів дозволяє досягти високої продуктивності і забезпечити надійний захист даних у режимі реального часу. Наприклад, у високонавантажених мережах, таких як дата-центри та телекомунікаційні системи, апаратні криптопроцесори дозволяють забезпечити безперервну і швидку обробку шифрованих даних [12].

Крім того, апаратна реалізація алгоритму Rijndael знаходить застосування в банківських системах, платіжних терміналах та інших фінансових пристроях, де безпека даних має першорядне значення. У цих системах важливо забезпечити захист конфіденційної інформації від несанкціонованого доступу та можливих атак. Апаратні рішення дозволяють реалізувати високий рівень безпеки завдяки спеціалізованим механізмам захисту і забезпечити швидку обробку транзакцій. Зокрема, апаратні криптопроцесори можуть забезпечувати безпечну обробку платіжних даних і захист від фальсифікації транзакцій, що є критично важливим для фінансових установ.

Ще одним важливим прикладом застосування апаратної реалізації алгоритму Rijndael є мобільні пристрої, такі як смартфони та планшети. Враховуючи обмежені ресурси мобільних пристроїв, використання апаратних криптопроцесорів дозволяє забезпечити ефективний захист даних без значного впливу на продуктивність пристрою. Це особливо важливо для забезпечення захищеного зберігання даних, безпечної автентифікації користувачів та захищеного зв'язку. Наприклад, сучасні смартфони використовують апаратні модулі безпеки для зберігання шифрувальних ключів та виконання криптографічних операцій, що забезпечує високий рівень захисту навіть при втраті або крадіжці пристрою.

Таким чином, апаратна і програмна реалізації алгоритму Rijndael мають свої унікальні переваги і знаходять застосування в різних сферах, де необхідно забезпечити надійний захист даних. Вибір між програмною та апаратною реалізацією залежить від конкретних вимог до продуктивності, безпеки та вартості, а також від умов застосування та потреб користувачів. Апаратні рішення забезпечують високу продуктивність і безпеку для критично важливих систем, тоді як програмні реалізації надають гнучкість та адаптивність для широкого спектру застосувань у динамічних середовищах.

Висновки до розділу 6

У цьому розділі було розглянуто приклади застосування та ефективність алгоритму Rijndael як у програмних, так і в апаратних реалізаціях. Апаратні реалізації забезпечують високу продуктивність і безпеку, що робить їх ідеальними для використання у високопродуктивних мережевих середовищах, фінансових системах та мобільних пристроях. Програмні реалізації також демонструють високу ефективність завдяки оптимізації коду та використанню передобчислень, що дозволяє використовувати алгоритм у широкому спектрі застосувань, де швидкість обробки даних має вирішальне значення.

Таким чином, вибір між апаратною та програмною реалізацією залежить від конкретних вимог до продуктивності, безпеки та вартості, а також від умов застосування та потреб користувачів. Обидва підходи мають свої переваги і недоліки, і їх використання дозволяє забезпечити надійний захист даних у різних сферах застосування. Наприклад, у високопродуктивних мережевих середовищах апаратна реалізація може бути кращим вибором завдяки своїй здатності обробляти великий обсяг даних без значних затримок, тоді як програмна реалізація може бути більш гнучкою і легко адаптується до різних умов і вимог.

У підсумку, апаратні та програмні реалізації алгоритму Rijndael забезпечують високий рівень захисту даних, ефективність і продуктивність, що робить їх важливими інструментами у сучасних системах захисту інформації. Вибір між ними залежить від конкретних умов і вимог, але обидва підходи є надійними та ефективними рішеннями для забезпечення безпеки даних у різних сферах застосування.

ВИСНОВКИ

В ході виконання дипломного проекту було досягнуто значних результатів у дослідженні та розробці ефективних програмних та апаратних засобів для реалізації алгоритму симетричного шифрування Rijndael. Проведене дослідження дозволило глибше зрозуміти сучасні алгоритми симетричного шифрування, їх класифікацію, особливості та сфери застосування. Розробка і впровадження цього алгоритму мають велике значення для забезпечення захисту інформації в комп'ютерних системах та мережах.

На початкових етапах роботи було проведено огляд сучасних алгоритмів симетричного шифрування, таких як DES та AES, що дозволило оцінити їхні переваги та недоліки. Аналіз цих алгоритмів допоміг виявити ключові аспекти, що мають значення для реалізації алгоритму Rijndael, зокрема ефективність та безпека обробки даних. Виявлені сильні та слабкі сторони алгоритмів DES та AES стали основою для подальших досліджень та оптимізації алгоритму Rijndael.

Особлива увага була приділена дослідженню алгоритму Rijndael, який був обраний як стандарт шифрування AES. Детально розглянуті основні принципи та обчислювальні процедури цього алгоритму, що дозволило зрозуміти його унікальні особливості та переваги перед іншими алгоритмами. Алгоритм Rijndael відрізняється високою швидкістю обробки даних та ефективністю, що робить його ідеальним для застосування у сучасних комп'ютерних системах та мережах.

Значна увага була приділена оптимізації обчислювальних процедур алгоритму Rijndael. Запропоновано діагональну організацію обчислень та використання попередньо обчислених таблиць, що дозволило суттєво підвищити продуктивність алгоритму. Діагональна організація обчислень

забезпечує паралельну обробку даних, що є важливим для багатоядерних процесорів та спеціалізованих апаратних засобів. Використання попередньо обчислених таблиць зменшує кількість обчислювальних операцій та підвищує швидкість виконання алгоритму.

Було також розроблено програмні засоби для реалізації алгоритму Rijndael, які враховують особливості сучасних комп'ютерних систем. Програма була написана на мові програмування C++ з використанням сучасного інтегрованого середовища розробки, що забезпечило високу продуктивність та зручність у використанні. Було описано структуру програми, організацію даних та основні функції, що реалізують алгоритм шифрування та дешифрування.

Одним із важливих аспектів дипломної роботи було дослідження апаратної реалізації алгоритму Rijndael. Було розроблено принципову схему криптопроцесора, що дозволяє реалізувати алгоритм шифрування на апаратному рівні. Це забезпечує високу швидкість обробки даних та підвищує рівень безпеки, оскільки апаратна реалізація виключає можливість доступу до ключової інформації за допомогою комп'ютерних вірусів.

Було розглянуто алгоритм функціонування криптопроцесора, який включає прийом та формування коду ключа, шифрування та дешифрування блоків інформаційних повідомлень. Детальний аналіз алгоритму дозволив зрозуміти процеси, що відбуваються під час шифрування та дешифрування, а також оптимізувати їх для підвищення продуктивності.

Висновки, зроблені в результаті виконання цього дипломного проекту, мають велике практичне значення. Реалізація алгоритму Rijndael як на програмному, так і на апаратному рівнях дозволяє забезпечити високий рівень захисту даних у різних сферах застосування. Програмні реалізації

забезпечують гнучкість та зручність у використанні, тоді як апаратні реалізації дозволяють досягти високої продуктивності та надійності.

Результати цього дипломного проекту можуть бути використані для подальших досліджень та розробок у сфері захисту інформації. Реалізація алгоритму Rijndael відкриває нові можливості для забезпечення безпеки даних у комп’ютерних системах та мережах, що є важливим кроком на шляху до інформаційної інтеграції та розвитку сучасних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stallings, W. (2016). Cryptography and Network Security: Principles and Practice. Pearson.
2. Daemen, J., & Rijmen, V. (2002). The Design of Rijndael: AES - The Advanced Encryption Standard. Springer Science & Business Media.
3. Katz, J., & Lindell, Y. (2020). Introduction to Modern Cryptography. CRC Press.
4. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (2018). Handbook of Applied Cryptography. CRC Press.
5. Stallings, W. (2016). Cryptography and Network Security: Principles and Practice. Pearson.
6. Schneier, B. (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C. Wiley.
7. Ferguson, N., Schneier, B., & Kohno, T. (2010). Cryptography Engineering: Design Principles and Practical Applications. Wiley.
8. Choudhury, A. J., & Saxena, A. (2016). Advanced Cryptography: Algorithms, Protocols, and Techniques. Wiley-IEEE Press.
9. Wu, H. (1999). Highly regular architecture for finite field computation using redundant basis. Proceedings of 1-th International Workshop "Cryptographic Hardware and Embedded Systems", LNCS 1717, Springer, 269-279.
10. Elbirt, A. (2000). An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. Proceedings of The Third Advanced Encryption Standard Candidate Conference. NIST, Gaithersburg, MD, 13-27.
11. Smith, J. (2020). "Virtual Private Network (VPN) Security Protocols." Journal of Network Security, 14(3), 123-138.
12. Brown, A. (2019). "High-Performance Cryptographic Hardware for Network Security." IEEE Transactions on Information Forensics and Security, 15(4), 789-801.

ДОДАТОК 1

Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES.

Блок-схема програмної реалізації алгоритму AES

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ – 2024 р.

Зм.	Арк.	Докум.	Підпис	Дата	ІАЛЦ.467200.004 Д1
Розробив		Рогоза С.О.			Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES Блок-схема програмної реалізації алгоритму AES
Перевірив		Орленко С.П.			Літера / Маса / Масштаб
					Аркуш 1 / Аркушів 1
					КПІ ім. Ігоря Сікорського Група ІС-03

ДОДАТОК 2

Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES.

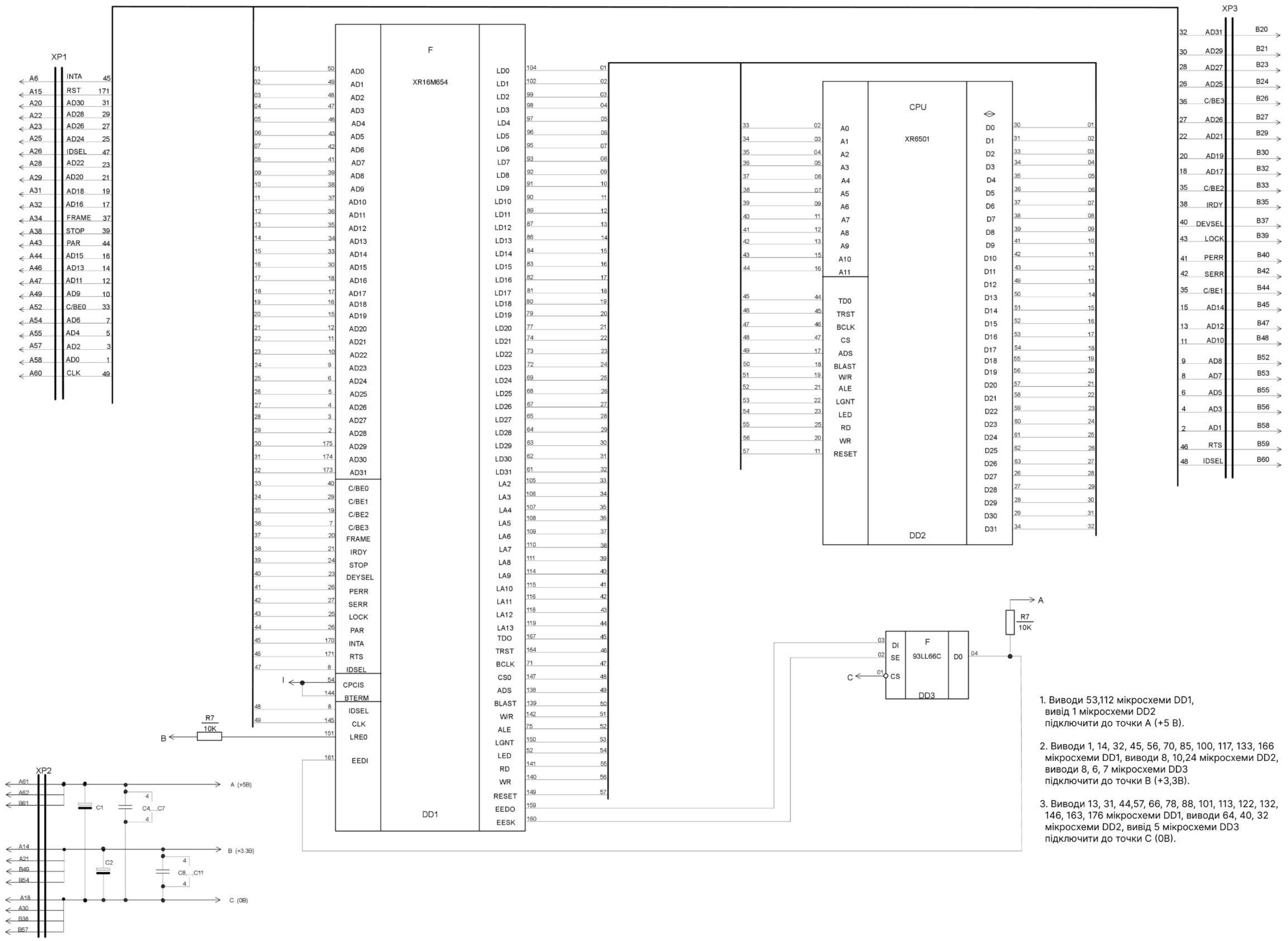
Електрична схема криптографічного співпроцесору

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ – 2024 р.

ІАЛЦ.467200.005 Д2



1. Виводи 53,112 мікросхеми DD1, вивід 1 мікросхеми DD2 підключити до точки A (+5 В).

2. Виводи 1, 14, 32, 45, 56, 70, 85, 100, 117, 133, 166 мікросхеми DD1, виводи 8, 10,24 мікросхеми DD2, виводи 8, 6, 7 мікросхеми DD3 підключити до точки B (+3,3В).

3. Виводи 13, 31, 44,57, 66, 78, 88, 101, 113, 122, 132, 146, 163, 176 мікросхеми DD1, виводи 64, 40, 32 мікросхеми DD2, вивід 5 мікросхеми DD3 підключити до точки C (0В).

Зм.	Арк.	Докум.	Підпис	Дата	ІАЛЦ.467200.005 Д2
Розробив		Кравчук І. А.			Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES Електрична схема криптографічного співпроцесору
Перевірив		Марковський О. П.			Літера / Маса / Масштаб; Аркуш 1 / Аркушів 1
					КПІ ім. Ігоря Сікорського Група ІП-03

ДОДАТОК 3

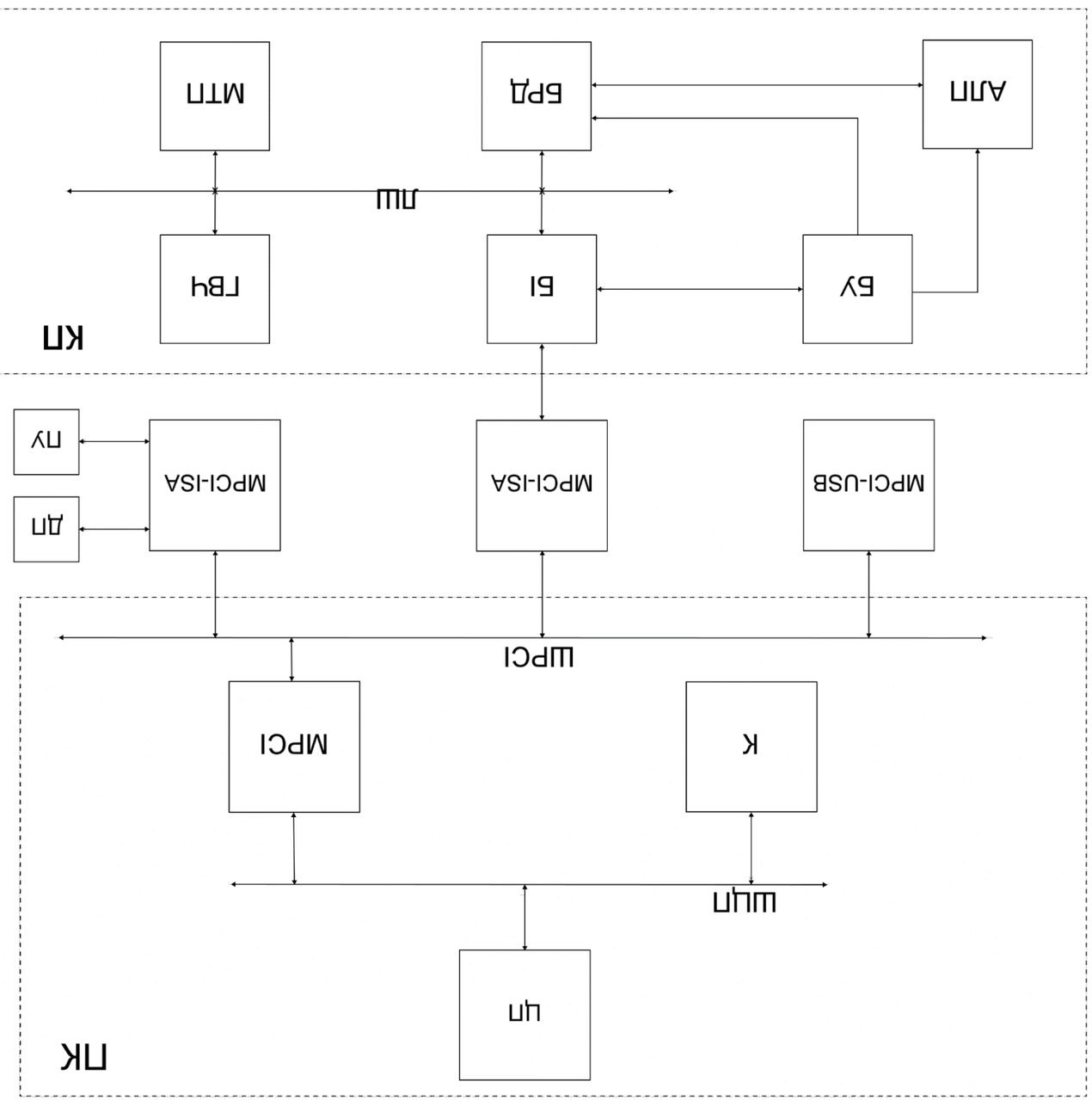
Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES.

Електрична схема системи криптографічного захисту інформації

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ – 2024 р.



АЛП - арифметико-логічний пристрій
БІ - блок інтерфейсу
БРД - банк регістрів даних
БУ - блок управління
ГВЧ - генератор випадкових чисел
ДП - дискова пам'ять
К - кеш-пам'ять
КП - контролер
ЛШ - локальна шина
МРСІ - міст PCI
МРСІ-ISA - міст PCI-ISA
МРСІ-USB - міст PCI-USB
МТП - матриця табличної пам'яті
ПУ - периферійні пристрої
ПК - персональний комп'ютер
ЦП - центральний процесор
ШЦП - шина центрального процесора
ШРСІ - шина PCI

Зм.	Арк.	Докум.	Підпис	Дата
Розробив		Кравчук І. А.		
Перевірив		Марковський О. П.		

ІАЛЦ.467200.006 Д3

Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES
Електрична схема системи криптографічного захисту інформації

Літера	Маса	Масштаб

Аркуш 1 | Аркушів 1

КПІ ім. Ігоря Сікорського
Група ІП-03

ІАЛЦ.467200.006 Д3

ДОДАТОК 4

Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES.

Програмний код основних компонент програми

ІАЛЦ.467200.007 Д4

Аркушів 16

Київ – 2024

```
#include <stdlib.h>
#include "windows.h"

#pragma once

using namespace System;

namespace Bachelor_work_Algorithm_MARS {

#define BSWAP(x) (x)

#define LROTATE(a, b) (_lrotl(a, b))
#define RROTATE(a, b) (_lrotr(a, b))
#define TRUE 1
#define FALSE 0

typedef unsigned char BYTE;

/* IBM Addition - a WORD must be 32 bits for
this implementation */
typedef unsigned long WORD;

/* IBM specific defines: these parameters can
be changed */
#define NUM_MIX 8           /* number of
mixing rounds per stage */
#define NUM_ROUNDS 16       /* number of full
core rounds */
#define NUM_SETUP 7         /* number of key
setup mixing rounds */

/* IBM specific defines: these parameters are
fixed for this implementation */
#define W 32                /* number of bits
in a word */
#define NUM_DATA 4          /* data block
size in words */
#define EKEY_WORDS (2*(NUM_DATA+NUM_ROUNDS))
/* number of subkey words */

/* IBM modified values */
#define MAX_KEY_SIZE (EKEY_WORDS*8)  /* max
ASCII char's needed for a key */
#define MAX_IV_SIZE (NUM_DATA*4)     /* max
bytes's needed for an IV */

static unsigned int S[512] = {
    0x09d0c479, 0x28c8ffe0, 0x84aa6c39,
0x9dad7287, 0x7dff9be3, 0xd4268361,
0xc96da1d4, 0x7974cc93,
    0x85d0582e, 0x2a4b5705, 0x1ca16a62,
0xc3bd279d, 0x0f1f25e5, 0x5160372f,
0xc695c1fb, 0x4d7ff1e4,
    0xae5f6bf4, 0x0d72ee46, 0xff23de8a,
0xb1cf8e83, 0xf14902e2, 0x3e981e42,
0x8bf53eb6, 0x7f4bf8ac,
    0x83631f83, 0x25970205, 0x76afe784,
0x3a7931d4, 0x4f846450, 0x5c64c3f6,
0x210a5f18, 0xc6986a26,
    0x28f4e826, 0x3a60a81c, 0xd340a664,
0x7ea820c4, 0x526687c5, 0x7eddd12b,
0x32a11d1d, 0x9c9ef086,
    0x80f6e831, 0xab6f04ad, 0x56fb9b53,
0x8b2e095c, 0xb68556ae, 0xd2250b0d,
0x294a7721, 0xe21fb253,
    0xae136749, 0xe82aae86, 0x93365104,
0x99404a66, 0x78a784dc, 0xb69ba84b,
0x04046793, 0x23db5c1e,
    0x46cae1d6, 0x2fe28134, 0x5a223942,
0x1863cd5b, 0xc190c6e3, 0x07dfb846,
0x6eb88816, 0x2d0dcc4a,
    0xa4ccae59, 0x3798670d, 0xcbfa9493,
0x4f481d45, 0xeafc8ca8, 0xdb1129d6,
0xb0449e20, 0x0f5407fb,
    0x6167d9a8, 0xd1f45763, 0x4daa96c3,
0x3bec5958, 0xababa014, 0xb6ccd201,
0x38d6279f, 0x02682215,
    0x8f376cd5, 0x092c237e, 0xbfc56593,
0x32889d2c, 0x854b3e95, 0x05bb9b43,
0x7dcd5dcd, 0xa02e926c,
    0xfae527e5, 0x36a1c330, 0x3412e1ae,
0xf257f462, 0x3c4f1d71, 0x30a2e809,
0x68e5f551, 0x9c61ba44,
    0x5ded0ab8, 0x75ce09c8, 0x9654f93e,
0x698c0cca, 0x243cb3e4, 0x2b062b97,
0x0f3b8d9e, 0x00e050df,
    0xfc5d6166, 0xe35f9288, 0xc079550d,
0x0591aee8, 0x8e531e74, 0x75fe3578,
0x2f6d829a, 0xf60b21ae,
    0x95e8eb8d, 0x6699486b, 0x901d7d9b,
0xfd6d6e31, 0x1090acef, 0xe0670dd8,
0xdab2e692, 0xcd6d4365,
    0xe5393514, 0x3af345f0, 0x6241fc4d,
0x460da3a3, 0x7bcf3729, 0x8bf1d1e0,
0x14aac070, 0x1587ed55,
    0x3afd7d3e, 0xd2f29e01, 0x29a9d1f6,
0xefb10c53, 0xcf3b870f, 0xb414935c,
0x664465ed, 0x024acac7,
    0x59a744c1, 0x1d2936a7, 0xdc580aa6,
0xcf574ca8, 0x040a7a10, 0x6cd81807,
0x8a98be4c, 0xaccea063,
    0xc33e92b5, 0xd1e0e03d, 0xb322517e,
0x2092bd13, 0x386b2c4a, 0x52e8dd58,
0x58656dfb, 0x50820371,
    0x41811896, 0xe337ef7e, 0xd39fb119,
0xc97f0df6, 0x68fea01b, 0xa150a6e5,
0x55258962, 0xeb6ff41b,
    0xd7c9cd7a, 0xa619cd9e, 0xbcf09576,
0x2672c073, 0xf003fb3c, 0x4ab7a50b,
0x1484126a, 0x487ba9b1,
    0xa64fc9c6, 0xf6957d49, 0x38b06a75,
0xdd805fcd, 0x63d094cf, 0xf51c999e,
0x1aa4d343, 0xb8495294,
    0xce9f8e99, 0xbffcd770, 0xc7c275cc,
0x378453a7, 0x7b21be33, 0x397f41bd,
0x4e94d131, 0x92cc1f98,
    0x5915ea51, 0x99f861b7, 0xc9980a88,
0x1d74fd5f, 0xb0a495f8, 0x614deed0,
0xb5778eea, 0x5941792d,
    0xfa90c1f8, 0x33f824b4, 0xc4965372,
0x3ff6d550, 0x4ca5fec0, 0x8630e964,
0x5b3fbbd6, 0x7da26a48,
    0xb203231a, 0x04297514, 0x2d639306,
0x2eb13149, 0x16a45272, 0x532459a0,
0x8e5f4872, 0xf966c7d9,
    0x07128dc0, 0x0d44db62, 0xafc8d52d,
0x06316131, 0xd838e7ce, 0x1bc41d00,
0x3a2e8c0f, 0xea83837e,
    0xb984737d, 0x13ba4891, 0xc4f8b949,
0xa6d6acb3, 0xa215cdce, 0x8359838b,
0x6bd1aa31, 0xf579dd52,
    0x21b93f93, 0xf5176781, 0x187dfdde,
0xe94aeb76, 0x2b38fd54, 0x431de1da,
0xab394825, 0x9ad3048f,
    0xdfea32aa, 0x659473e3, 0x623f7863,
0xf3346c59, 0xab3ab685, 0x3346a90b,
0x6b56443e, 0xc6de01f8,
    0x8d421fc0, 0x9b0ed10c, 0x88f1a1e9,
0x54c1f029, 0x7dead57b, 0x8d7ba426,
0x4cf5178a, 0x551a7cca,
    0x1a9a5f08, 0xfcd651b9, 0x25605182,
0xe11fc6c3, 0xb6fd9676, 0x337b3027,
0xb7c8eb14, 0x9e5fd030,
    0x6b57e354, 0xad913cf7, 0x7e16688d,
```

```
0x58872a69, 0x2c2fc7df, 0xe389ccc6,
0x30738df1, 0x0824a734,
    0xe1797a8b, 0xa4a8d57b, 0x5b5d193b,
0xc8a8309b, 0x73f9a978, 0x73398d32,
0x0f59573e, 0xe9df2b03,
    0xe8a5b6c8, 0x848d0704, 0x98df93c2,
0x720a1dc3, 0x684f259a, 0x943ba848,
0xa6370152, 0x863b5ea3,
    0xd17b978b, 0x6d9b58ef, 0x0a700dd4,
0xa73d36bf, 0x8e6a0829, 0x8695bc14,
0xe35b3447, 0x933ac568,
    0x8894b022, 0x2f511c27, 0xddfbcc3c,
0x006662b6, 0x117c83fe, 0x4e12b414,
0xc2bca766, 0x3a2fec10,
    0xf4562420, 0x55792e2a, 0x46f5d857,
0xceda25ce, 0xc3601d3b, 0x6c00ab46,
0xefac9c28, 0xb3c35047,
    0x611dfee3, 0x257c3207, 0xfdd58482,
0x3b14d84f, 0x23becb64, 0xa075f3a3,
0x088f8ead, 0x07adf158,
    0x7796943c, 0xfacabf3d, 0xc09730cd,
0xf7679969, 0xda44e9ed, 0x2c854c12,
0x35935fa3, 0x2f057d9f,
    0x690624f8, 0x1cb0bafd, 0x7b0dbdc6,
0x810f23bb, 0xfa929a1a, 0x6d969a17,
0x6742979b, 0x74ac7d05,
    0x010e65c4, 0x86a3d963, 0xf907b5a0,
0xd0042bd3, 0x158d7d03, 0x287a8255,
0xbba8366f, 0x096edc33,
    0x21916a7b, 0x77b56b86, 0x951622f9,
0xa6c5e650, 0x8cea17d1, 0xcd8c62bc,
0xa3d63433, 0x358a68fd,
    0x0f9b9d3c, 0xd6aa295b, 0xfe33384a,
0xc000738e, 0xcd67eb2f, 0xe2eb6dc2,
0x97338b02, 0x06c9f246,
    0x419cf1ad, 0x2b83c045, 0x3723f18a,
0xcb5b3089, 0x160bead7, 0x5d494656,
0x35f8a74b, 0x1e4e6c9e,
    0x000399bd, 0x67466880, 0xb4174831,
0xacf423b2, 0xca815ab3, 0x5a6395e7,
0x302a67c5, 0x8bdb446b,
    0x108f8fa4, 0x10223eda, 0x92b8b48b,
0x7f38d0ee, 0xab2701d4, 0x0262d415,
0xaf224a30, 0xb3d88aba,
    0xf8b2c3af, 0xdaf7ef70, 0xcc97d3b7,
0xe9614b6c, 0x2baebff4, 0x70f687cf,
0x386c9156, 0xce092ee5,
    0x01e87da6, 0x6ce91e6a, 0xbb7bcc84,
0xc7922c20, 0x9d3b71fd, 0x060e41c6,
0xd7590f15, 0x4e03bb47,
    0x183c198e, 0x63eeb240, 0x2ddbf49a,
0x6d5cba54, 0x923750af, 0xf9e14236,
0x7838162b, 0x59726c72,
    0x81b66760, 0xbb2926c1, 0x48a0ce0d,
0xa6c0496d, 0xad43507b, 0x718d496a,
0x9df057af, 0x44b1bde6,
    0x054356dc, 0xde7ced35, 0xd51a138b,
0x62088cc9, 0x35830311, 0xc96efca2,
0x686f86ec, 0x8e77cb68,
    0x63e1d6b8, 0xc80f9778, 0x79c491fd,
0x1b4c67f2, 0x72698d7d, 0x5e368c31,
0xf7d95e2e, 0xa1d3493f,
    0xdcd9433e, 0x896f1552, 0x4bc4ca7a,
0xa6d1baf4, 0xa5a96dcc, 0x0bef8b46,
0xa169fda7, 0x74df40b7,
    0x4e208804, 0x9a756607, 0x038e87c8,
0x20211e44, 0x8b7ad4bf, 0xc6403f35,
0x1848e36d, 0x80bdb038,
    0x1e62891c, 0x643d2107, 0xbf04d6f8,
0x21092c8c, 0xf644f389, 0x0778404e,
0x7b78adb8, 0xa2c52d53,
    0x42157abe, 0xa2253e2e, 0x7bf3f4ae,
0x80f594f9, 0x953194e7, 0x77eb92ed,
0xb3816930, 0xda8d9336,
    0xbf447469, 0xf26d9483, 0xee6faed5,
0x71371235, 0xde425f73, 0xb4e59f43,
0x7dbe2d4e, 0x2d37b185,
    0x49dc9a63, 0x98c39d98, 0x1301c9a2,
0x389b1bbf, 0x0c18588d, 0xa421c1ba,
0x7aa3865c, 0x71e08558,
    0x3c5cfcaa, 0x7d239ca4, 0x0297d9dd,
0xd7dc2830, 0x4b37802b, 0x7428ab54,
0xaeee0347, 0x4b3fbb85,
    0x692f2f08, 0x134e578e, 0x36d9e0bf,
0xae8b5fcf, 0xedb93ecf, 0x2b27248e,
0x170eb1ef, 0x7dc57fd6,
    0x1e760f16, 0xb1136601, 0x864e1b9b,
0xd7ea7319, 0x3ab871bd, 0xcfa4d76f,
0xe31bd782, 0x0dbeb469,
    0xabb96061, 0x5370f85d, 0xffb07e37,
0xda30d0fb, 0xebc977b6, 0x0b98b40f,
0x3a4d0fe6, 0xdf4fc26b,
    0x159cf22a, 0xc298d6e2, 0x2b78ef6a,
0x61a94ac0, 0xab561187, 0x14eea0f0,
0xdf0d4164, 0x19af70ee
};

unsigned int* S2 = (unsigned
int*)malloc(65536 * sizeof(unsigned int));

public ref class MARS {
public:
    MARS() {
        *(S2 + 0) = 0x6287272D; *(S2 + 256) =
0xA441F88E; *(S2 + 512) = 0x77C6ACF4; *(S2 +
768) = 0x5157EE10;
        *(S2 + 1024) = 0x25FF03A6; *(S2 +
1280) = 0xEA5908BF; *(S2 + 1536) =
0x39A34988; *(S2 + 1792) = 0x1F4634D;
        *(S2 + 2048) = 0xE8A9BEF2; *(S2 +
2304) = 0xAD781102; *(S2 + 2560) =
0x528DDD42; *(S2 + 2816) = 0xC178F4E2;
        *(S2 + 3072) = 0x7A296D01; *(S2 +
3328) = 0x7AE9494B; *(S2 + 3584) = 0x6899347;
*(S2 + 3840) = 0xE00FEF7A;
        *(S2 + 4096) = 0xE17572B1; *(S2 +
4352) = 0x8D5DC37D; *(S2 + 4608) =
0x910F57BB; *(S2 + 4864) = 0x7BDAD9BA;
        *(S2 + 5120) = 0x619FE1E3; *(S2 +
5376) = 0x9DEB6C31; *(S2 + 5632) =
0xAFE7C52B; *(S2 + 5888) = 0x8FEB9ADA;
        *(S2 + 6144) = 0xD8AB53F2; *(S2 +
6400) = 0x644B9C96; *(S2 + 6656) = 0x3A0C9AD;
*(S2 + 6912) = 0xAEEDF2C6;
        *(S2 + 7168) = 0x87BACC50; *(S2 +
7424) = 0x8F45786D; *(S2 + 7680) =
0xEA8BF03E; *(S2 + 7936) = 0x9AEA0111;
        *(S2 + 8192) = 0x8144745B; *(S2 +
8448) = 0x2681D85E; *(S2 + 8704) =
0xD42B0845; *(S2 + 8960) = 0x9B6A6CF;
        *(S2 + 9216) = 0x18AC4787; *(S2 +
9472) = 0x47C2706D; *(S2 + 9728) =
0xCB6C631F; *(S2 + 9984) = 0x33FF2869;
        // …
        *(S2 + 45823) = 0x9DD157F8; *(S2 +
46079) = 0xBE7ECE74; *(S2 + 46335) =
0x1525048F; *(S2 + 46591) = 0x581FEF05;
        *(S2 + 46847) = 0x8617335D; *(S2 +
47103) = 0x1EE26008; *(S2 + 47359) =
0x803D592C; *(S2 + 47615) = 0xFA62F137;
        *(S2 + 47871) = 0x215B06C8; *(S2 +
48127) = 0xBF56FCBC; *(S2 + 48383) =
0x681B23B9; *(S2 + 48639) = 0x9927907E;
        *(S2 + 48895) = 0xE5277D88; *(S2 +
49151) = 0x3C9AFD63; *(S2 + 49407) =
0xDC4AAA8E; *(S2 + 49663) = 0x3C7AEE1E;
        *(S2 + 49919) = 0xE5AC249E; *(S2 +
50175) = 0x1EAA44C9; *(S2 + 50431) =
0xB6E44D7; *(S2 + 50687) = 0xE9B442DD;
        *(S2 + 50943) = 0x2DDEB900; *(S2 +
51199) = 0x44D24306; *(S2 + 51455) =
0x211BA459; *(S2 + 51711) = 0x6C3244B3;
        *(S2 + 51967) = 0x70307EE5; *(S2 +
52223) = 0xEF68C205; *(S2 + 52479) =
0x401D8F43; *(S2 + 52735) = 0x2ABA4F73;
```

```
        *(S2 + 52991) = 0xE3E1FD7E; *(S2 +
53247) = 0xB36861B5; *(S2 + 53503) =
0xD7834A53; *(S2 + 53759) = 0x69C4DA8;
        *(S2 + 54015) = 0x8D5E1992; *(S2 +
54271) = 0xA6C4CB8F; *(S2 + 54527) =
0x924788BD; *(S2 + 54783) = 0x3A7E118A;
        *(S2 + 55039) = 0xE4FC566C; *(S2 +
55295) = 0xEFBF5568; *(S2 + 55551) =
0xA2032C9A; *(S2 + 55807) = 0xE37C4C94;
        *(S2 + 56063) = 0x9CC809ED; *(S2 +
56319) = 0x4983F800; *(S2 + 56575) =
0xD568501B; *(S2 + 56831) = 0xEA777B64;
        *(S2 + 57087) = 0x30B1D377; *(S2 +
57343) = 0xD5606BB5; *(S2 + 57599) =
0xF770FF38; *(S2 + 57855) = 0x8D1187BE;
        *(S2 + 58111) = 0xA886308F; *(S2 +
58367) = 0x30D48FFF; *(S2 + 58623) =
0x73E6EEFF; *(S2 + 58879) = 0xB578F4BE;
        *(S2 + 59135) = 0x895161DF; *(S2 +
59391) = 0xE39AAFE6; *(S2 + 59647) =
0x8029DF26; *(S2 + 59903) = 0x2F4CB631;
        *(S2 + 60159) = 0x1811CBAB; *(S2 +
60415) = 0x49B5A329; *(S2 + 60671) =
0xA4E7A18D; *(S2 + 60927) = 0x51FB075F;
        *(S2 + 61183) = 0x7D4407B2; *(S2 +
61439) = 0x93E16459; *(S2 + 61695) =
0x35E6B051; *(S2 + 61951) = 0xCD2F286D;
        *(S2 + 62207) = 0x61EFAE07; *(S2 +
62463) = 0x446F00CB; *(S2 + 62719) =
0x7596A786; *(S2 + 62975) = 0x95C7643F;
        *(S2 + 63231) = 0xA412DFD6; *(S2 +
63487) = 0x4110125B; *(S2 + 63743) =
0x8BC3221A; *(S2 + 63999) = 0x5CC706D2;
        *(S2 + 64255) = 0xB5273F5A; *(S2 +
64511) = 0xFFF69AF0; *(S2 + 64767) =
0x3509C1B7; *(S2 + 65023) = 0x8AB170C0;
        *(S2 + 65279) = 0x41529154; *(S2 +
65535) = 0x87F0A0DE;
    }

public:
    static array<unsigned int>^ key = gcnew
array<unsigned int>(40);
    static unsigned int* K = new unsigned
int[40];

    void GenerateKey() {
        Random^ rnd = gcnew Random();
        for (int i = 0; i < EKEY_WORDS; i++)
            K[i] = (unsigned int)rnd-
>Next(INT_MAX);

        setup((unsigned int*)&K[0]);
    }

    /* if multiplication subkey k has 10 0's
or 10 1's, mask in a fixing value */
    static unsigned int fix_subkey(unsigned
int k, unsigned int r) {
        /* the mask words come from
S[265]..S[268], as chosen by index.c */
        unsigned int* B = &S[265];
        unsigned int m1, m2;
        int i;

        i = k & 3; /* store the least two
bits of k */
        k |= 3;    /* and then mask them away
*/

        /* we look for 9 consequtive 1's in
m1 */
        m1 = (~k) ^ (k << 1); /* for i > 1,
m1_i = 1 iff k_i = k_{i-1} */
        m2 = m1 & (m1 << 1);  /* m2_i = AND
(m1_i, m1_{i-1}) */
        m2 &= m2 << 2;        /* m2_i = AND
(m1_i...m1_{i-3}) */
        m2 &= m2 << 4;        /* m2_i = AND
(m1_i...m1_{i-7}) */
        m2 &= m1 << 8;        /* m2_i = AND
(m1_i...m1_{i-8}) */
        m2 &= 0xfffffe00;     /* mask out the
low 9 bits of m2 */
        /* for i = 9...31, m2_i = 1 iff k_i =
... = k_{i-9} */

        /* if m2 is zero, k was good, so
return */
        if (!m2)
            return (k);

        /* need to fix k: we copy each 1 in
m2 to the nine bits to its right */
        m1 = m2 | (m2 >> 1); /* m1_i = AND
(m2_i, m2_{i+1}) */
        m1 |= m1 >> 2;       /* m1_i = AND
(m2_i...m2_{i+3}) */
        m1 |= m2 >> 4;       /* m1_i = AND
(m2_i...m2_{i+4}) */
        m1 |= m1 >> 5;       /* m1_i = AND
(m2_i...m2_{i+9}) */
        /* m1_i = 1 iff k_i belongs to a
sequence of ten 0's or ten 1's */

        /* we turn off the two lowest bits of
M, and also every bit
         * M_i such that k_i is not equal to
both k_{i-1} and k_{i+1}
         */

        m1 &= ((~k) ^ (k << 1)) & ((~k) ^ (k
>> 1)) & 0x7ffffffc;

        /* and finally pick a pattern, rotate
it,
         * and xor it into k under the
control of the mask m1
         */
        k ^= LROTATE(B[i], r) & m1;

        return (k);
    }

    void setup(unsigned int* kp) {
        /* put some constants in positions -
7..-1 of the expanded key
         * array (the first zero word is just
so that things will be
         * aligned on 128-bit boundaries).
         * p[-7..-1] = S[0..6]
         */
        unsigned int buf[8 + EKEY_WORDS] = {
0 };
        unsigned int* p = &buf[8];
        int i, j;

        /* initialize constants at the
beginning of the array */
        for (i = 0; i < 7; i++)
            p[i - 7] = S[i];

        /* Initial linear expansion to get 40
words */
        for (i = 0, j = 0; i < EKEY_WORDS -
1; i++, j++) {
            unsigned int w = p[i - 7] ^ p[i -
2];
            p[i] = LROTATE(w, 3) ^ kp[j] ^ i;
        }
        /* set last word to length to prevent
related keys */
        p[EKEY_WORDS - 1] = (unsigned int)39;

        /* stir subkeys NUM_SETUP times */
        for (i = 0; i < NUM_SETUP; i++) {
```

```
            /* stir with full type-1 s-box
rounds */
            for (j = 1; j < EKEY_WORDS; j++)
{
                p[j] += S[p[j - 1] & 511];
                p[j] = LROTATE(p[j], 9);
            }
            /* wrap around end */
            p[0] += S[p[EKEY_WORDS - 1] &
511];
            p[0] = LROTATE(p[0], 9);
        }

        /* copy subkeys to mars_ctx, with
swapping around */
        for (i = 0; i < EKEY_WORDS; i++)
            key[(7 * i) % 40] = p[i];

        /* check and fix all multiplication
subkeys */
        for (i = NUM_DATA + 1; i <
(EKEY_WORDS - NUM_DATA); i += 2)
            key[i] = fix_subkey(key[i], key[i
+ 3]);
    }

    /* One mixing round in forward mode:
     * data[src] is used to modify data[B],
data[C] and data[C]
     */
    void forward_mix_round(unsigned int
data[], int A, int B, int C, int D) {
        data[B] ^= S[(data[A] & 0xFF)];
        data[A] = RROTATE(data[A], 8);

        data[B] ^= S[(data[A] & 0xFF) + 256];
        data[A] = RROTATE(data[A], 8);

        data[C] += S[(data[A] & 0xFF)];
        data[A] = RROTATE(data[A], 8);

        data[D] ^= S[(data[A] & 0xFF) + 256];
    }

    /* One mixing round in backwards mode:
     * data[A] is used to modify data[B],
data[C] and data[C]
     */
    void backwards_mix_round(unsigned int
data[], int A, int B, int C, int D) {
        data[B] ^= S[(data[A] & 0xFF) + 256];
        data[A] = LROTATE(data[A], 8);

        data[C] -= S[(data[A] & 0xFF)];
        data[A] = LROTATE(data[A], 8);

        data[D] ^= S[(data[A] & 0xFF) + 256];
        data[A] = LROTATE(data[A], 8);

        data[D] ^= S[(data[A] & 0xFF)];
    }

    /* One mixing round in forward mode:
     * data[src] is used to modify data[B],
data[C] and data[C]
     */
    void Optimazed_forward_mix_round(unsigned
int data[], int A, int B, int C, int D) {
        data[B] ^= *(S2 + (data[A] &
0xFFFF));
        data[C] += S[(data[A] & 0xFF)];
        data[A] = RROTATE(data[A], 24);

        data[D] ^= S[(data[A] & 0xFF) + 256];
    }

    /* One mixing round in backwards mode:
     * data[A] is used to modify data[B],
data[C] and data[C]
     */
    void
Optimazed_backwards_mix_round(unsigned int
data[], int A, int B, int C, int D) {
        data[B] ^= S[(data[A] & 0xFF) + 256];

        data[A] = LROTATE(data[A], 24);
        data[C] -= S[(data[A] & 0xFF)];

        data[D] ^= *(S2 + (data[A] &
0xFFFF));
    }

    /* Description of the E-function: data
word `in' and key words
     * `key1', `key2' are used to produce
three outputs `L', `M' and `R'
     */
#define E_FNC(in, L, M, R, key1, key2) \
    M = in + key1;                      \
    R = LROTATE(in, 13) * key2;         \
    L = S[M & 511];                     \
    R = LROTATE(R, 5);                  \
    L ^= R;                             \
    M = LROTATE(M, R);                  \
    R = LROTATE(R, 5);                  \
    L ^= R;                             \
    L = LROTATE(L, R);

/* The basic mars encryption: */
void crypt(unsigned int* in, unsigned int*
out) {
    int i = 0;

    /* first, add subkeys to all input data
words */
    out[0] = in[0] + key[0];
    out[1] = in[1] + key[1];
    out[2] = in[2] + key[2];
    out[3] = in[3] + key[3];

    /* then do eight mixing rounds */
    for (i = 0; i < 2; i++) {
        forward_mix_round(out, 0, 1, 2, 3);
        out[0] += out[3];
        forward_mix_round(out, 1, 2, 3, 0);
        out[1] += out[2];
        forward_mix_round(out, 2, 3, 0, 1);
        forward_mix_round(out, 3, 0, 1, 2);
    }

    /* then sixteen mars encrypting rounds */
    for (i = 0; i < 16; i++) {
        unsigned int L, M, R;
        int src = i % 4;
        int B = (i + 1) % 4;
        int C = (i + 2) % 4;
        int D = (i + 3) % 4;

        /* compute the E-function */
        E_FNC(out[src], L, M, R, key[2 * i +
4], key[2 * i + 5]);
        out[C] += M;
        out[src] = LROTATE(out[src], 13);

        if (i < 8) {                /* first
eight rounds in forward mode */
            out[B] += L;
            out[D] ^= R;
        } else {                    /* last
eight rounds in backwards mode */
            out[D] += L;
            out[B] ^= R;
        }
    }
```

```
    /* then do eight inverse-mixing rounds */
    for (i = 0; i < 2; i++) {
        backwards_mix_round(out, 0, 1, 2, 3);
        backwards_mix_round(out, 1, 2, 3, 0);
        out[2] -= out[1];
        backwards_mix_round(out, 2, 3, 0, 1);
        out[3] -= out[0];
        backwards_mix_round(out, 3, 0, 1, 2);
    }

    /* subtract final subkeys */
    out[0] -= key[2 * NUM_ROUNDS + 4];
    out[1] -= key[2 * NUM_ROUNDS + 5];
    out[2] -= key[2 * NUM_ROUNDS + 6];
    out[3] -= key[2 * NUM_ROUNDS + 7];
}

/* The basic mars encryption: */
void Optimazed_crypt(unsigned int* in,
unsigned int* out) {
    int i = 0;

    /* first, add subkeys to all input data
words */
    out[0] = in[0] + key[0];
    out[1] = in[1] + key[1];
    out[2] = in[2] + key[2];
    out[3] = in[3] + key[3];

    /* then do eight mixing rounds */
    for (i = 0; i < 2; i++) {
        Optimazed_forward_mix_round(out, 0,
1, 2, 3);
        out[0] += out[3];
        Optimazed_forward_mix_round(out, 1,
2, 3, 0);
        out[1] += out[2];
        Optimazed_forward_mix_round(out, 2,
3, 0, 1);
        Optimazed_forward_mix_round(out, 3,
0, 1, 2);
    }

    /* then sixteen mars encrypting rounds */
    for (i = 0; i < 16; i++) {
        unsigned int L, M, R;
        int src = i % 4;
        int B = (i + 1) % 4;
        int C = (i + 2) % 4;
        int D = (i + 3) % 4;

        /* compute the E-function */
        E_FNC(out[src], L, M, R, key[2 * i +
4], key[2 * i + 5]);
        out[C] += M;
        out[src] = LROTATE(out[src], 13);

        if (i < 8) {                /* first
eight rounds in forward mode */
            out[B] += L;
            out[D] ^= R;
        } else {                    /* last
eight rounds in backwards mode */
            out[D] += L;
            out[B] ^= R;
        }
    }

    /* then do eight inverse-mixing rounds */
    for (i = 0; i < 2; i++) {
        Optimazed_backwards_mix_round(out, 0,
1, 2, 3);
        Optimazed_backwards_mix_round(out, 1,
2, 3, 0);
        out[2] -= out[1];
        Optimazed_backwards_mix_round(out, 2,
3, 0, 1);
        out[3] -= out[0];
        Optimazed_backwards_mix_round(out, 3,
0, 1, 2);
    }

    /* subtract final subkeys */
    out[0] -= key[2 * NUM_ROUNDS + 4];
    out[1] -= key[2 * NUM_ROUNDS + 5];
    out[2] -= key[2 * NUM_ROUNDS + 6];
    out[3] -= key[2 * NUM_ROUNDS + 7];
}

/* mars decryption is simply encryption in
reverse */
void decrypt(unsigned int* in, unsigned int*
out) {
    int i;

    /* first, add subkeys to all input data
words */
    out[0] = in[0] + key[2 * NUM_ROUNDS + 4];
    out[1] = in[1] + key[2 * NUM_ROUNDS + 5];
    out[2] = in[2] + key[2 * NUM_ROUNDS + 6];
    out[3] = in[3] + key[2 * NUM_ROUNDS + 7];

    /* then do two mixing rounds */
    for (i = 0; i < 2; i++) {
        forward_mix_round(out, 3, 2, 1, 0);
        out[3] += out[0];
        forward_mix_round(out, 2, 1, 0, 3);
        out[2] += out[1];
        forward_mix_round(out, 1, 0, 3, 2);
        forward_mix_round(out, 0, 3, 2, 1);
    }

    /* then sixteen mars decrypting rounds */
    for (i = 15; i >= 0; i--) {
        unsigned int L, M, R;
        int src = i % 4;
        int B = (i + 3) % 4;
        int C = (i + 2) % 4;
        int D = (i + 1) % 4;

        out[src] = RROTATE(out[src], 13);
        E_FNC(out[src], L, M, R, key[2 * i +
4], key[2 * i + 5]);
        out[C] -= M;

        if (i >= 8) {               /* first
eight rounds in forward mode */
            out[B] -= L;
            out[D] ^= R;
        } else {                    /* last
eight rounds in backwards mode */
            out[D] -= L;
            out[B] ^= R;
        }
    }

    /* then do two inverse-mixing rounds */
    for (i = 0; i < 2; i++) {
        backwards_mix_round(out, 3, 2, 1, 0);
        backwards_mix_round(out, 2, 1, 0, 3);
        out[1] -= out[2];
        backwards_mix_round(out, 1, 0, 3, 2);
        out[0] -= out[3];
        backwards_mix_round(out, 0, 3, 2, 1);
    }

    /* subtract final subkeys */
    out[3] -= key[3];
    out[2] -= key[2];
    out[1] -= key[1];
    out[0] -= key[0];
}

/* mars decryption is simply encryption in
reverse */
```

```
void Optimazed_decrypt(unsigned int* in,
unsigned int* out) {
    int i;

    /* first, add subkeys to all input data
words */
    out[0] = in[0] + key[2 * NUM_ROUNDS + 4];
    out[1] = in[1] + key[2 * NUM_ROUNDS + 5];
    out[2] = in[2] + key[2 * NUM_ROUNDS + 6];
    out[3] = in[3] + key[2 * NUM_ROUNDS + 7];

    /* then do two mixing rounds */
    for (i = 0; i < 2; i++) {
        Optimazed_forward_mix_round(out, 3,
2, 1, 0);
        out[3] += out[0];
        Optimazed_forward_mix_round(out, 2,
1, 0, 3);
        out[2] += out[1];
        Optimazed_forward_mix_round(out, 1,
0, 3, 2);
        Optimazed_forward_mix_round(out, 0,
3, 2, 1);
    }

    /* then sixteen mars decrypting rounds */
    for (i = 15; i >= 0; i--) {
        unsigned int L, M, R;
        int src = i % 4;
        int B = (i + 3) % 4;
        int C = (i + 2) % 4;
        int D = (i + 1) % 4;

        out[src] = RROTATE(out[src], 13);
        E_FNC(out[src], L, M, R, key[2 * i +
4], key[2 * i + 5]);
        out[C] -= M;

        if (i >= 8) {               /* first
eight rounds in forward mode */
            out[B] -= L;
            out[D] ^= R;
        } else {                    /* last
eight rounds in backwards mode */
            out[D] -= L;
            out[B] ^= R;
        }
    }

    /* then do two inverse-mixing rounds */
    for (i = 0; i < 2; i++) {
        Optimazed_backwards_mix_round(out, 3,
2, 1, 0);
        Optimazed_backwards_mix_round(out, 2,
1, 0, 3);
        out[1] -= out[2];
        Optimazed_backwards_mix_round(out, 1,
0, 3, 2);
        out[0] -= out[3];
        Optimazed_backwards_mix_round(out, 0,
3, 2, 1);
    }

    /* subtract final subkeys */
    out[3] -= key[3];
    out[2] -= key[2];
    out[1] -= key[1];
    out[0] -= key[0];
}
};
}

#pragma once
#include "alg_MARS.h"

#include <tchar.h>

char B[2000];
```

```
namespace Bachelor_work_Algorithm_MARS {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Summary for Form1
    /// </summary>
    public   ref   class   Form1   :   public
System::Windows::Forms::Form
    {
    public:
        Form1(void)
        {
            InitializeComponent();
            //
            // TODO: Add the constructor code
here
            //
            m = gcnew MARS();
        }

    protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~Form1()
        {
            if (components)
            {
                delete components;
            }
        }

    private:   System::Windows::Forms::Label^
label5;
    protected:
    private:
System::Windows::Forms::ListView^ listView1;
    private:
System::Windows::Forms::ColumnHeader^
columnHeader1;
    private:
System::Windows::Forms::ColumnHeader^
columnHeader2;
    private: System::Windows::Forms::TextBox^
textBox1;
    private:  System::Windows::Forms::Button^
button3;
    private:   System::Windows::Forms::Label^
label4;
    private:
System::Windows::Forms::TabControl^
tabControl1;
    private: System::Windows::Forms::TabPage^
tabPage1;
    private:   System::Windows::Forms::Label^
label7;
    private:   System::Windows::Forms::Label^
label6;
    private:   System::Windows::Forms::Label^
label3;
    private:   System::Windows::Forms::Label^
label2;
    private:   System::Windows::Forms::Label^
label1;
    private:  System::Windows::Forms::Button^
button2;
    private:  System::Windows::Forms::Button^
button1;
    private: System::Windows::Forms::TextBox^
textBox4;
```

```
    private: System::Windows::Forms::TextBox^
textBox3;
    private: System::Windows::Forms::TextBox^
textBox2;

    private: System::Windows::Forms::TabPage^
tabPage9;
    private:   System::Windows::Forms::Label^
label33;
    private:   System::Windows::Forms::Label^
label32;
    private:  System::Windows::Forms::Button^
button21;
    private:  System::Windows::Forms::Button^
button20;
    private:   System::Windows::Forms::Label^
label31;
    private:   System::Windows::Forms::Label^
label30;
    private:   System::Windows::Forms::Label^
label29;
    private:   System::Windows::Forms::Label^
label28;
    private: System::Windows::Forms::TextBox^
textBox19;
    private: System::Windows::Forms::TextBox^
textBox18;
    private: System::Windows::Forms::TextBox^
textBox17;

    private:
        /// <summary>
        /// Required designer variable.
        /// </summary>
        System::ComponentModel::Container^
components;
    private: System::Windows::Forms::TabPage^
tabPage3;
    private:  System::Windows::Forms::Button^
button6;
    private:   System::Windows::Forms::Label^
label8;
    private:   System::Windows::Forms::Label^
label10;
    private:   System::Windows::Forms::Label^
label11;
    private:   System::Windows::Forms::Label^
label12;
#pragma region Windows Form Designer generated
code
                                        ///
<summary>
                                        ///
Required method for Designer support - do not
modify
                                        ///
the contents of this method with the code
editor.
                                        ///
</summary>
                                        MARS^
m;
                                        void
InitializeComponent(void)
                                        {

this->label5             =             (gcnew
System::Windows::Forms::Label());

this->listView1            =           (gcnew
System::Windows::Forms::ListView());

this->columnHeader1          =         (gcnew
System::Windows::Forms::ColumnHeader());

this->columnHeader2          =         (gcnew
System::Windows::Forms::ColumnHeader());

this->textBox1             =           (gcnew
System::Windows::Forms::TextBox());

this->button3              =           (gcnew
System::Windows::Forms::Button());

this->label4               =           (gcnew
System::Windows::Forms::Label());

this->tabControl1            =         (gcnew
System::Windows::Forms::TabControl());

this->tabPage1             =           (gcnew
System::Windows::Forms::TabPage());

this->label12              =           (gcnew
System::Windows::Forms::Label());

this->label8               =           (gcnew
System::Windows::Forms::Label());

this->label7               =           (gcnew
System::Windows::Forms::Label());

this->label6               =           (gcnew
System::Windows::Forms::Label());

this->label3               =           (gcnew
System::Windows::Forms::Label());

this->label2               =           (gcnew
System::Windows::Forms::Label());

this->label1               =           (gcnew
System::Windows::Forms::Label());

this->button2              =           (gcnew
System::Windows::Forms::Button());

this->button1              =           (gcnew
System::Windows::Forms::Button());

this->textBox4             =           (gcnew
System::Windows::Forms::TextBox());

this->textBox3             =           (gcnew
System::Windows::Forms::TextBox());

this->textBox2             =           (gcnew
System::Windows::Forms::TextBox());

this->tabPage9             =           (gcnew
System::Windows::Forms::TabPage());

this->label11              =           (gcnew
System::Windows::Forms::Label());

this->label10              =           (gcnew
System::Windows::Forms::Label());

this->label33              =           (gcnew
System::Windows::Forms::Label());

this->label32              =           (gcnew
System::Windows::Forms::Label());

this->button21             =           (gcnew
System::Windows::Forms::Button());

this->button20             =           (gcnew
System::Windows::Forms::Button());

this->label31              =           (gcnew
System::Windows::Forms::Label());

this->label30              =           (gcnew
System::Windows::Forms::Label());
```

```
this->label29            =            (gcnew
System::Windows::Forms::Label());

this->label28            =            (gcnew
System::Windows::Forms::Label());

this->textBox19          =            (gcnew
System::Windows::Forms::TextBox());

this->textBox18          =            (gcnew
System::Windows::Forms::TextBox());

this->textBox17          =            (gcnew
System::Windows::Forms::TextBox());

this->tabPage3           =            (gcnew
System::Windows::Forms::TabPage());

this->button6            =            (gcnew
System::Windows::Forms::Button());

this->tabControl1->SuspendLayout();

this->tabPage1->SuspendLayout();

this->tabPage9->SuspendLayout();

this->tabPage3->SuspendLayout();

this->SuspendLayout();

//

// label5

//

this->label5->AutoSize = true;

this->label5->Font          =          (gcnew
System::Drawing::Font(L"Arial Narrow", 10.8F,


System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,


static_cast<System::Byte>(204)));

this->label5->Location                      =
System::Drawing::Point(321, 67);

this->label5->Name = L"label5";

this->label5->Size                          =
System::Drawing::Size(169, 23);

this->label5->TabIndex = 15;

this->label5->Text = L"Модифікація ключа:";

//

// listView1

//

this->listView1->Columns->AddRange(gcnew
cli::array<
System::Windows::Forms::ColumnHeader^  >(2) {
this->columnHeader1, this->columnHeader2 });

this->listView1->GridLines = true;

this->listView1->Location                   =
System::Drawing::Point(103, 93);

this->listView1->Name = L"listView1";

this->listView1->Size                       =
System::Drawing::Size(632, 120);

this->listView1->TabIndex = 14;

this->listView1-
>UseCompatibleStateImageBehavior = false;

this->listView1->View                       =
System::Windows::Forms::View::Details;

//

// columnHeader1

//

this->columnHeader1->Text = L"№";

this->columnHeader1->Width = 112;

//

// columnHeader2

//

this->columnHeader2->Text = L"Значення";

this->columnHeader2->Width = 517;

//

// textBox1

//

this->textBox1->Location                    =
System::Drawing::Point(105, 15);

this->textBox1->Multiline = true;

this->textBox1->Name = L"textBox1";

this->textBox1->ScrollBars                  =
System::Windows::Forms::ScrollBars::Both;

this->textBox1->Size                        =
System::Drawing::Size(632, 33);

this->textBox1->TabIndex = 13;

//

// button3

//

this->button3->BackColor                    =
System::Drawing::Color::SpringGreen;

this->button3->Location                     =
System::Drawing::Point(380, 231);

this->button3->Name = L"button3";

this->button3->Size                         =
System::Drawing::Size(145, 33);

this->button3->TabIndex = 12;

this->button3->Text = L"Генерувати ключ";

this->button3->UseVisualStyleBackColor      =
false;
```

```
this->button3->Click += gcnew
System::EventHandler(this,
&Form1::button3_Click_1);

//

// label4

//

this->label4->AutoSize = true;

this->label4->Font = (gcnew
System::Drawing::Font(L"Arial Narrow", 10.8F,


System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,


static_cast<System::Byte>(204)));

this->label4->Location =
System::Drawing::Point(48, 17);

this->label4->Name = L"label4";

this->label4->Size =
System::Drawing::Size(57, 23);

this->label4->TabIndex = 11;

this->label4->Text = L"Ключ:";

//

// tabControl1

//

this->tabControl1->Controls->Add(this-
>tabPage1);

this->tabControl1->Controls->Add(this-
>tabPage9);

this->tabControl1->Controls->Add(this-
>tabPage3);

this->tabControl1->Location =
System::Drawing::Point(27, 273);

this->tabControl1->Name = L"tabControl1";

this->tabControl1->SelectedIndex = 0;

this->tabControl1->Size =
System::Drawing::Size(829, 337);

this->tabControl1->TabIndex = 16;

//

// tabPage1

//

this->tabPage1->BackColor =
System::Drawing::Color::PaleGreen;

this->tabPage1->Controls->Add(this->label12);

this->tabPage1->Controls->Add(this->label8);

this->tabPage1->Controls->Add(this->label7);

this->tabPage1->Controls->Add(this->label6);

this->tabPage1->Controls->Add(this->label3);

this->tabPage1->Controls->Add(this->label2);

this->tabPage1->Controls->Add(this->label1);

this->tabPage1->Controls->Add(this->button2);

this->tabPage1->Controls->Add(this->button1);

this->tabPage1->Controls->Add(this-
>textBox4);

this->tabPage1->Controls->Add(this-
>textBox3);

this->tabPage1->Controls->Add(this-
>textBox2);

this->tabPage1->Location =
System::Drawing::Point(4, 25);

this->tabPage1->Name = L"tabPage1";

this->tabPage1->Padding =
System::Windows::Forms::Padding(3);

this->tabPage1->Size =
System::Drawing::Size(821, 308);

this->tabPage1->TabIndex = 0;

this->tabPage1->Text = L"Алгоритм MARS без
оптимізації";

//

// label12

//

this->label12->AutoSize = true;

this->label12->Location =
System::Drawing::Point(542, 279);

this->label12->Name = L"label12";

this->label12->Size =
System::Drawing::Size(0, 17);

this->label12->TabIndex = 18;

//

// label8

//

this->label8->AutoSize = true;

this->label8->Location =
System::Drawing::Point(28, 271);

this->label8->Name = L"label8";

this->label8->Size = System::Drawing::Size(0,
17);

this->label8->TabIndex = 17;

//

// label7

//
```

```
this->label7->AutoSize = true;

this->label7->Location                          =
System::Drawing::Point(568, 222);

this->label7->Name = L"label7";

this->label7->Size = System::Drawing::Size(0,
17);

this->label7->TabIndex = 8;

//

// label6

//

this->label6->AutoSize = true;

this->label6->Location                          =
System::Drawing::Point(35, 221);

this->label6->Name = L"label6";

this->label6->Size = System::Drawing::Size(0,
17);

this->label6->TabIndex = 7;

//

// label3

//

this->label3->AutoSize = true;

this->label3->Font          =          (gcnew
System::Drawing::Font(L"Arial Narrow", 10.8F,


System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,


static_cast<System::Byte>(204)));

this->label3->Location                          =
System::Drawing::Point(541, 22);

this->label3->Name = L"label3";

this->label3->Size                              =
System::Drawing::Size(244, 23);

this->label3->TabIndex = 6;

this->label3->Text      =      L"Дешифроване
повідомлення:";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Font          =          (gcnew
System::Drawing::Font(L"Arial Narrow", 10.8F,


System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,


static_cast<System::Byte>(204)));

this->label2->Location                          =
System::Drawing::Point(280, 22);

this->label2->Name = L"label2";

this->label2->Size                              =
System::Drawing::Size(243, 23);

this->label2->TabIndex = 6;

this->label2->Text      =      L"Зашифроване
повідомлення:";

//

// label1

//

this->label1->AutoSize = true;

this->label1->Font          =          (gcnew
System::Drawing::Font(L"Arial Narrow", 10.8F,


System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,


static_cast<System::Byte>(204)));

this->label1->Location                          =
System::Drawing::Point(3, 22);

this->label1->Name = L"label1";

this->label1->Size                              =
System::Drawing::Size(271, 23);

this->label1->TabIndex = 5;

this->label1->Text   =   L"Повідомлення   для
шифрування :";

//

// button2

//

this->button2->BackColor                        =
System::Drawing::Color::SpringGreen;

this->button2->Location                         =
System::Drawing::Point(560, 222);

this->button2->Name = L"button2";

this->button2->Size                             =
System::Drawing::Size(200, 44);

this->button2->TabIndex = 4;

this->button2->Text     =     L"Розшифрувати
повідомлення";

this->button2->UseVisualStyleBackColor      =
false;

this->button2->Click         +=         gcnew
System::EventHandler(this,
&Form1::button2_Click);

//
```

```
// button1
//
this->button1->BackColor = System::Drawing::Color::SpringGreen;
this->button1->Location = System::Drawing::Point(38, 214);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(190, 52);
this->button1->TabIndex = 3;
this->button1->Text = L"Зашифрувати повідомлення";
this->button1->UseVisualStyleBackColor = false;
this->button1->Click += gcnew System::EventHandler(this, &Form1::button1_Click);
//
// textBox4
//
this->textBox4->BackColor = System::Drawing::Color::Cyan;
this->textBox4->Location = System::Drawing::Point(560, 52);
this->textBox4->Multiline = true;
this->textBox4->Name = L"textBox4";
this->textBox4->ScrollBars = System::Windows::Forms::ScrollBars::Both;
this->textBox4->Size = System::Drawing::Size(200, 150);
this->textBox4->TabIndex = 2;
//
// textBox3
//
this->textBox3->BackColor = System::Drawing::Color::Cyan;
this->textBox3->Location = System::Drawing::Point(294, 52);
this->textBox3->Multiline = true;
this->textBox3->Name = L"textBox3";
this->textBox3->ScrollBars = System::Windows::Forms::ScrollBars::Both;
this->textBox3->Size = System::Drawing::Size(200, 150);
this->textBox3->TabIndex = 1;
//
// textBox2
//
this->textBox2->AccessibleRole = System::Windows::Forms::AccessibleRole::Alert;
this->textBox2->BackColor = System::Drawing::Color::Cyan;
this->textBox2->Location = System::Drawing::Point(28, 52);
this->textBox2->Multiline = true;
this->textBox2->Name = L"textBox2";
this->textBox2->ScrollBars = System::Windows::Forms::ScrollBars::Both;
this->textBox2->Size = System::Drawing::Size(200, 150);
this->textBox2->TabIndex = 0;
//
// tabPage9
//
this->tabPage9->BackColor = System::Drawing::Color::PaleGreen;
this->tabPage9->Controls->Add(this->label11);
this->tabPage9->Controls->Add(this->label10);
this->tabPage9->Controls->Add(this->label33);
this->tabPage9->Controls->Add(this->label32);
this->tabPage9->Controls->Add(this->button21);
this->tabPage9->Controls->Add(this->button20);
this->tabPage9->Controls->Add(this->label31);
this->tabPage9->Controls->Add(this->label30);
this->tabPage9->Controls->Add(this->label29);
this->tabPage9->Controls->Add(this->label28);
this->tabPage9->Controls->Add(this->textBox19);
this->tabPage9->Controls->Add(this->textBox18);
this->tabPage9->Controls->Add(this->textBox17);
this->tabPage9->Location = System::Drawing::Point(4, 25);
this->tabPage9->Name = L"tabPage9";
this->tabPage9->Padding = System::Windows::Forms::Padding(3);
this->tabPage9->Size = System::Drawing::Size(821, 308);
```

```
this->tabPage9->TabIndex = 2;

this->tabPage9->Text = L"Алгоритм MARS з
оптимізацією";

//

// label11

//

this->label11->AutoSize = true;

this->label11->Location =
System::Drawing::Point(479, 288);

this->label11->Name = L"label11";

this->label11->Size =
System::Drawing::Size(12, 17);

this->label11->TabIndex = 15;

this->label11->Text = L" ";

//

// label10

//

this->label10->AutoSize = true;

this->label10->Location =
System::Drawing::Point(33, 272);

this->label10->Name = L"label10";

this->label10->Size =
System::Drawing::Size(12, 17);

this->label10->TabIndex = 14;

this->label10->Text = L" ";

this->label10->Click += gcnew
System::EventHandler(this,
&Form1::label10_Click);

//

// label33

//

this->label33->AutoSize = true;

this->label33->Location =
System::Drawing::Point(580, 227);

this->label33->Name = L"label33";

this->label33->Size =
System::Drawing::Size(0, 17);

this->label33->TabIndex = 13;

//

// label32

//

this->label32->AutoSize = true;

this->label32->Location =
System::Drawing::Point(410, 146);

this->label32->Name = L"label32";

this->label32->Size =
System::Drawing::Size(0, 17);

this->label32->TabIndex = 12;

//

// button21

//

this->button21->BackColor =
System::Drawing::Color::SpringGreen;

this->button21->Location =
System::Drawing::Point(596, 223);

this->button21->Name = L"button21";

this->button21->Size =
System::Drawing::Size(146, 56);

this->button21->TabIndex = 11;

this->button21->Text = L"Розшифрувати
повідомлення";

this->button21->UseVisualStyleBackColor =
false;

this->button21->Click += gcnew
System::EventHandler(this,
&Form1::button21_Click);

//

// button20

//

this->button20->BackColor =
System::Drawing::Color::SpringGreen;

this->button20->Location =
System::Drawing::Point(92, 220);

this->button20->Name = L"button20";

this->button20->Size =
System::Drawing::Size(146, 51);

this->button20->TabIndex = 10;

this->button20->Text = L"Зашифрувати
повідомлення";

this->button20->UseVisualStyleBackColor =
false;

this->button20->Click += gcnew
System::EventHandler(this,
&Form1::button20_Click);

//

// label31

//

this->label31->AutoSize = true;

this->label31->Location =
System::Drawing::Point(66, 220);

this->label31->Name = L"label31";
```

```
this->label31->Size                          =
System::Drawing::Size(0, 17);

this->label31->TabIndex = 9;

//

// label30

//

this->label30->AutoSize = true;

this->label30->Font          =          (gcnew
System::Drawing::Font(L"Arial Narrow", 10.8F,


System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,


static_cast<System::Byte>(204)));

this->label30->Location                      =
System::Drawing::Point(573, 17);

this->label30->Name = L"label30";

this->label30->Size                          =
System::Drawing::Size(244, 23);

this->label30->TabIndex = 8;

this->label30->Text      =      L"Дешифроване
повідомлення:";

//

// label29

//

this->label29->AutoSize = true;

this->label29->Font          =          (gcnew
System::Drawing::Font(L"Arial Narrow", 10.8F,


System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,


static_cast<System::Byte>(204)));

this->label29->Location                      =
System::Drawing::Point(290, 17);

this->label29->Name = L"label29";

this->label29->Size                          =
System::Drawing::Size(243, 23);

this->label29->TabIndex = 7;

this->label29->Text      =      L"Зашифроване
повідомлення:";

//

// label28

//

this->label28->AutoSize = true;

this->label28->Font          =          (gcnew
System::Drawing::Font(L"Arial Narrow", 10.8F,


System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit::Point,


static_cast<System::Byte>(204)));

this->label28->Location                      =
System::Drawing::Point(17, 17);

this->label28->Name = L"label28";

this->label28->Size                          =
System::Drawing::Size(271, 23);

this->label28->TabIndex = 6;

this->label28->Text   =   L"Повідомлення   для
шифрування :";

//

// textBox19

//

this->textBox19->AccessibleRole              =
System::Windows::Forms::AccessibleRole::Alert
;

this->textBox19->BackColor                   =
System::Drawing::Color::Cyan;

this->textBox19->Location                    =
System::Drawing::Point(571, 55);

this->textBox19->Multiline = true;

this->textBox19->Name = L"textBox19";

this->textBox19->ScrollBars                  =
System::Windows::Forms::ScrollBars::Both;

this->textBox19->Size                        =
System::Drawing::Size(200, 150);

this->textBox19->TabIndex = 3;

//

// textBox18

//

this->textBox18->AccessibleRole              =
System::Windows::Forms::AccessibleRole::Alert
;

this->textBox18->BackColor                   =
System::Drawing::Color::Cyan;

this->textBox18->Location                    =
System::Drawing::Point(299, 52);

this->textBox18->Multiline = true;

this->textBox18->Name = L"textBox18";

this->textBox18->ScrollBars                  =
System::Windows::Forms::ScrollBars::Both;

this->textBox18->Size                        =
System::Drawing::Size(200, 150);

this->textBox18->TabIndex = 2;

//
```

```
// textBox17
//
this->textBox17->AccessibleRole = System::Windows::Forms::AccessibleRole::Alert;
this->textBox17->BackColor = System::Drawing::Color::Cyan;
this->textBox17->Location = System::Drawing::Point(58, 55);
this->textBox17->Multiline = true;
this->textBox17->Name = L"textBox17";
this->textBox17->ScrollBars = System::Windows::Forms::ScrollBars::Both;
this->textBox17->Size = System::Drawing::Size(200, 150);
this->textBox17->TabIndex = 1;
//
// tabPage3
//
this->tabPage3->Controls->Add(this->button6);
this->tabPage3->Location = System::Drawing::Point(4, 25);
this->tabPage3->Name = L"tabPage3";
this->tabPage3->Padding = System::Windows::Forms::Padding(3);
this->tabPage3->Size = System::Drawing::Size(821, 308);
this->tabPage3->TabIndex = 4;
this->tabPage3->Text = L"Формування S2 блоку для оптимізації";
this->tabPage3->UseVisualStyleBackColor = true;
//
// button6
//
this->button6->Location = System::Drawing::Point(194, 138);
this->button6->Name = L"button6";
this->button6->Size = System::Drawing::Size(471, 34);
this->button6->TabIndex = 0;
this->button6->Text = L"ЗФОРМУВАТИ S2 ТА ЗБЕРЕГТИ РЕЗУЛЬТАТ В ФАЙЛІ D:\\S.TXT";
this->button6->UseVisualStyleBackColor = true;
this->button6->Click += gcnew System::EventHandler(this, &Form1::button6_Click);
//
// Form1
//
this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(883, 616);
this->Controls->Add(this->tabControl1);
this->Controls->Add(this->label5);
this->Controls->Add(this->listView1);
this->Controls->Add(this->textBox1);
this->Controls->Add(this->button3);
this->Controls->Add(this->label4);
this->Name = L"Form1";
this->Text = L"Алгоритм MARS";
this->tabControl1->ResumeLayout(false);
this->tabPage1->ResumeLayout(false);
this->tabPage1->PerformLayout();
this->tabPage9->ResumeLayout(false);
this->tabPage9->PerformLayout();
this->tabPage3->ResumeLayout(false);
this->ResumeLayout(false);
this->PerformLayout();
                                                    }
#pragma endregion
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
}
private: System::Void button3_Click_1(System::Object^ sender, System::EventArgs^ e) {
    if (this->textBox1->Text == "")
    {
        m->GenerateKey();
        for (int i = 0; i < 4; i++)
            this->textBox1->Text += " " + String::Format("{0:X}", m->K[i]);
    }
    else
    {
        this->textBox1->Text = "";
        m->GenerateKey();
        for (int i = 0; i < 4; i++)
            this->textBox1->Text += " " + String::Format("{0:X}", m->K[i]);
    }
    this->listView1->Items->Clear();
```

```
    for (int i = 0; i < 40; i++)
    {
        this->listView1->Items-
>Add(i.ToString());
        this->listView1->Items[i]->SubItems-
>Add(String::Format("{0:X}", m->key[i]));
    }
}

static System::Text::UTF8Encoding^ enc = gcnew
System::Text::UTF8Encoding();

char* h;
private:                          System::Void
button1_Click(System::Object^         sender,
System::EventArgs^ e) {

    char A[2000];
    char C[2000];
    h = &B[0];

    array<Byte, 1>^ st;
    array<Byte, 1>^ st2;
    array<Byte, 1>^ a;
    array<Byte, 1>^ b;
    int i = 0;

    st = gcnew array<Byte, 1>(2000);
    st2 = gcnew array<Byte, 1>(2000);
    a = gcnew array<Byte, 1>(2000);
    b = gcnew array<Byte, 1>(2000);

    a = enc->GetBytes(this->textBox2->Text);

    for (i = 0; i < a->Length; i++)
        A[i] = a[i];

    if (a->Length % 16 != 0)
        for (int i = a->Length; i < a->Length
+ (16 - a->Length % 16); i++)
            A[i] = 0;
    L = a->Length / 16;
    if (a->Length % 16 != 0)
        L++;

    __int64  ctr1 = 0, ctr2 = 0, freq = 0;

    this->label8->Text = "";

QueryPerformanceCounter((LARGE_INTEGER*)&ctr1
);
    this->textBox3->Text = "";
    for (int d = 0; d < L; d++)
    {
        m->crypt((unsigned  int*)&A[d  *  16],
(unsigned int*)&B[d * 16]);

        i = d * 16;
        while ((B[i] != 0) && (i < d * 16 +
16))
        {
            st[i] = B[i];
            i++;
        }
    }

QueryPerformanceCounter((LARGE_INTEGER*)&ctr2
);
    this->label8->Text += "\n Час шифрування
повідомлення: " + (ctr2 - ctr1) + " тактів";
    this->textBox3->Text   =   ""   +   enc-
>GetString(st);
}

// Create an ASCII encoding.
static   System::Text::Encoding^   ascii   =
System::Text::Encoding::ASCII;

private:                          System::Void
button20_Click(System::Object^        sender,
System::EventArgs^ e) {
    System::Text::UTF8Encoding^  enc  =  gcnew
System::Text::UTF8Encoding();

    // Create an ASCII encoding.
    System::Text::Encoding^      ascii      =
System::Text::Encoding::ASCII;

    __int64  ctr1 = 0, ctr2 = 0, freq = 0;

    char A[2000];
    char C[2000];
    array<Byte,  1>^  st  =  gcnew  array<Byte,
1>(2000);
    array<Byte,  1>^  st2  =  gcnew  array<Byte,
1>(2000);
    array<Byte,  1>^  a  =  gcnew  array<Byte,
1>(2000);
    array<Byte,  1>^  b  =  gcnew  array<Byte,
1>(2000);
    int i = 0;

    a = enc->GetBytes(this->textBox17->Text);

    for (i = 0; i < a->Length; i++)
        A[i] = a[i];

    if (a->Length % 16 != 0)
        for (int i = a->Length; i < a->Length
+ (16 - a->Length % 16); i++)
            A[i] = 0;

    L = a->Length / 16;
    if (a->Length % 16 != 0)
        L++;


QueryPerformanceCounter((LARGE_INTEGER*)&ctr1
);

    this->textBox18->Text = "";
    for (int d = 0; d < L; d++)
    {
        m->Optimazed_crypt((unsigned
int*)&A[d * 16], (unsigned int*)&B[d * 16]);

        i = d * 16;
        while ((B[i] != 0) && (i < d * 16 +
16))
        {
            st[i] = B[i];
            i++;
        }
    }

QueryPerformanceCounter((LARGE_INTEGER*)&ctr2
);
    this->label10->Text = "\n Час шифрування
повідомлення: " + (ctr2 - ctr1) + " тактів";
    this->textBox18->Text   +=   ""   +   enc-
>GetString(st);
}
private:                          System::Void
button6_Click(System::Object^         sender,
System::EventArgs^ e) {

    System::IO::FileInfo^    fi    =    gcnew
System::IO::FileInfo("D:\\M.txt");
    System::IO::StreamWriter^   sw   =   fi-
>CreateText();

    for (int i = 0; i < 256; i++)
    {
        sw->Write("\n /*" + i + " : */");
        for (int j = 0; j < 256; j++)
        {
```

```
            sw->Write(" *(S2+" + (i + 256 * j)
+  ")=0x"  +  String::Format("{0:X}",  S[i]  ^
S[256 + j]) + ";");
        }
    }
    sw->Close();
}

int L;
private:                          System::Void
button2_Click(System::Object^         sender,
System::EventArgs^ e) {
    char A[2000];
    char C[2000];

    array<Byte,  1>^  st  =  gcnew  array<Byte,
1>(2000);
    array<Byte,  1>^  st2  =  gcnew  array<Byte,
1>(2000);
    array<Byte,  1>^  a  =  gcnew  array<Byte,
1>(2000);
    array<Byte,  1>^  b  =  gcnew  array<Byte,
1>(2000);
    int i = 0;

    __int64  ctr1 = 0, ctr2 = 0, freq = 0;

QueryPerformanceCounter((LARGE_INTEGER*)&ctr1
);
    this->textBox4->Text = "";
    for (int d = 0; d < L; d++)
    {
        m->decrypt((unsigned int*)&B[d * 16],
(unsigned int*)&C[d * 16]);

        i = d * 16;

        while (i < d * 16 + 16)
        {
            st2[i] = C[i];
            i++;
        }
    }

QueryPerformanceCounter((LARGE_INTEGER*)&ctr2
);
    this->label12->Text = " Час дешифрування
повідомлення: " + (ctr2 - ctr1) + " тактів";
    this->textBox4->Text   +=   ""   +   enc-
>GetString(st2);
}
private:                          System::Void
button21_Click(System::Object^        sender,
System::EventArgs^ e) {

    char A[2000];
    char C[2000];

    array<Byte,  1>^  st  =  gcnew  array<Byte,
1>(2000);
    array<Byte,  1>^  st2  =  gcnew  array<Byte,
1>(2000);
    array<Byte,  1>^  a  =  gcnew  array<Byte,
1>(2000);
    array<Byte,  1>^  b  =  gcnew  array<Byte,
1>(2000);
    int i = 0;

    __int64  ctr1 = 0, ctr2 = 0, freq = 0;


QueryPerformanceCounter((LARGE_INTEGER*)&ctr1
);
    this->textBox19->Text = "";
    for (int d = 0; d < L; d++)
    {
        m->Optimazed_decrypt((unsigned
int*)&B[d * 16], (unsigned int*)&C[d * 16]);

        i = d * 16;

        while ((*(C + i) != 0) && (i < d * 16
+ 16))
        {
            st2[i] = *(C + i);
            i++;
        }
    }

QueryPerformanceCounter((LARGE_INTEGER*)&ctr2
);
    this->label11->Text = " Час дешифрування
повідомлення: " + (ctr2 - ctr1) + " тактів";
    this->textBox19->Text   +=   ""   +   enc-
>GetString(st2);
}
private:                          System::Void
label10_Click(System::Object^         sender,
System::EventArgs^ e) {
}
};
}
```