Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «магістр»

на тему:

ОПТИМІЗАЦІЯ АЛГОРИТМІВ ШИФРУВАННЯ ДЛЯ ВИКОРИСТАННЯ В МАЛОПОТУЖНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

Виконав:

здобувач 2 курсу

групи М-КН-21

спеціальності 122 «Комп’ютерні науки»

Мартиненков Максим Олександрович

Науковий керівник:

к.ю.н., доцент Кіндрат П. В.

Рівне – 2025

ЗМІСТ

СПИСОК ТЕРМІНІВ ТА УМОВНИХ ПОЗНАЧЕНЬ 5
ВСТУП..... 8
РОЗДІЛ 1 ШИФРУВАННЯ ТА ВИБІР АЛГОРИТМУ ДЛЯ ОПТИМІЗАЦІЇ В МАЛОПОТУЖНИХ СИСТЕМАХ 11
1.1 Поняття криптографічного захисту інформації 11
1.2 Популярні алгоритми шифрування 14
1.2.1 Порівняння характеристик сучасних алгоритмів 14
1.2.2 Проблеми криптостійкості алгоритмів шифрування 15
1.3 Особливості малопотужних обчислювальних систем..... 16
1.3.1 Обмеження IoT-пристроїв та вбудованих систем..... 17
1.3.2 Вимоги до алгоритмів для IoT та вбудованих систем..... 19
1.4 Критерії вибору алгоритму для оптимізації 20
1.4.1 Продуктивність та обчислювальна складність 21
1.4.2 Простота реалізації та портативність коду..... 22
1.4.3 Енергоефективність та споживання пам’яті..... 23
1.5 Обґрунтування вибору сімейства ChaCha20 для оптимізації..... 24
1.5.1 Характеристики алгоритмів сімейства ChaCha..... 25
1.5.2 Переваги ChaCha20 порівняно з AES для малопотужних систем 26
1.5.3 Аналіз оптимізацій алгоритмів ChaCha 27
Висновки до розділу 1 29
РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ ОПТИМІЗАЦІЇ АЛГОРИТМУ CHACHA20 31
2.1 Детальний аналіз алгоритму ChaCha20 31
2.1.1 Структура раундової функції..... 31
2.1.2 Математичні операції та їх складність..... 31
2.1.3 Ініціалізація стану та генерація ключового потоку..... 32
2.2 Оцінка безпеки алгоритму..... 32
2.2.1 Відомі атаки на сімейство ChaCha 33

2.2.2 Оцінка запасу криптостійкості 33
2.3 Обґрунтування зменшення кількості раундів 34
2.4 Теоретичне моделювання оптимізації ChaCha12 35
2.4.1 Оцінка продуктивності алгоритму 36
2.4.2 Оцінка споживання пам’яті 36
2.4.3 Завдання практичної реалізації 37
Висновок до розділу 2 37
РОЗДІЛ 3 39
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ 39
3.1 Вибір та обґрунтування засобів реалізації 39
3.1.1 Обґрунтування вибору мови програмування C# 39
3.1.2 Вибір платформи та середовища розробки 40
3.1.3 Бібліотеки та інструменти для реалізації 42
3.2 Реалізація модифікованого алгоритму ChaCha12 43
3.2.1 Архітектура програмної реалізації 43
3.2.2 Імплементація раундової функції 45
3.2.3 Оптимізація програмного коду 46
3.3 Інтеграція з модулем зчитування з веб-камери 47
3.3.1 Архітектура системи шифрування відеопотоку 47
3.3.2 Захоплення та обробка відеоданих 47
3.3.3 Реалізація шифрування в реальному часі 48
3.4 Методика експериментального дослідження 48
3.4.1 Критерії оцінювання продуктивності 48
3.4.2 Тестове середовище та конфігурація 49
3.4.3 Планування експериментів 49
Висновки до розділу 3 50
РОЗДІЛ 4 52
ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ ТА ОПТИМІЗАЦІЯ 52
4.1 Результати продуктивності 52
4.2 Криптографічна якість виходу 53

4.3 Аналітичне моделювання часу виконання 54
4.4 Енергоефективність та споживання ресурсів 55
4.5 Статистична валідація результатів 56
4.6 Платформозалежний аналіз та рекомендації 57
4.7 Синтез результатів і практичні рекомендації 58
Висновки до розділу 4 59
ВИСНОВКИ 61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 63
ДОДАТКИ ..... **Ошибка! Закладка не определена.**
ДОДАТОК А ..... **Ошибка! Закладка не определена.**
Таблиця 4.1 – Пропускна здатність та затримка ..... **Ошибка! Закладка не определена.**
ДОДАТОК С ..... **Ошибка! Закладка не определена.**
Таблиця 4.4 – Енергоефективність та ресурс батареї ..... **Ошибка! Закладка не определена.**
ДОДАТОК Е ..... **Ошибка! Закладка не определена.**
Таблиця 4.5.1 – Вплив компіляторських оптимізацій на ChaCha12 .... **Ошибка! Закладка не определена.**
ДОДАТОК F ..... **Ошибка! Закладка не определена.**
Таблиця 4.6 – Платформозалежний аналіз продуктивності **Ошибка! Закладка не определена.**
ДОДАТОК Н ..... **Ошибка! Закладка не определена.**
Таблиця 4.3 – Модель розкладання часу виконання ..... **Ошибка! Закладка не определена.**
ДОДАТОК І ..... **Ошибка! Закладка не определена.**
Таблиця 4.3.2 – Операційна вартість базових мікрооперацій на ARM **Ошибка! Закладка не определена.**

СПИСОК ТЕРМІНІВ ТА УМОВНИХ ПОЗНАЧЕНЬ

AEAD	Authenticated Encryption with Associated Data, режим шифрування з автентифікацією додаткових даних, напр. ChaCha20-Poly1305.
AES	Advanced Encryption Standard, симетричний блочний шифр з довжинами ключів 128, 192, 256 біт і 10/12/14 раундами відповідно.
ARX	Addition-Rotation-XOR, клас криптографічних конструкцій без S-box із використанням операцій додавання по модулю 2^{32}, циклічних зсувів і XOR.
Bouncy Castle	Криптографічна бібліотека з підтримкою сучасних алгоритмів, зокрема ChaCha/Poly1305.
ChaCha20/12/8	Сімейство потокових ARX-шифрів; цифра означає кількість раундів, ChaCha12 використовується як компроміс продуктивність/безпека для IoT.
Counter	Лічильник блоків у потокових шифрах типу ChaCha, інкрементується для кожного 64-байтного блоку вихідного ключового потоку.
CPU utilization	Частка завантаження процесора під час виконання криптографічних операцій у вимірювальних сценаріях.
DES/3DES	Data Encryption Standard і Triple DES; блочні шифри зі 64-бітовим блоком, у 3DES застосовується потрійне шифрування для підвищення стійкості.
Energy efficiency	Питомі енерговитрати на одиницю оброблених даних, важлива метрика для автономних MCU-вузлів.

Inline/Inlining	Техніка заміни виклику функції її тілом для зменшення накладних витрат виклику.
IoT	Інтернет речей (Internet of things)
Keystream	Ключовий потік байтів, який XOR-иться з відкритим текстом для отримання шифротексту у потокових схемах.
Latency	Затримка обробки блока/повідомлення, критична для сценаріїв реального часу та потокової передачі.
Loop unrolling	Розгортання циклів для зменшення кількості ітерацій та покращення інструкційного конвеєра.
MCU	Microcontroller Unit, мікроконтролерна платформа на кшталт ARM Cortex-M, RISC-V, з обмеженими ресурсами.
Memory footprint	Обсяг зайнятої оперативної пам’яті (RAM) реалізацією алгоритму та пов’язаних структур даних.
Nonce	Одноразове значення 96 біт у RFC-конфігураціях ChaCha, використовується спільно з лічильником і ключем для унікальності потоку.
Nonce-reuse	Помилка повторного використання nonce з тим самим ключем, що призводить до компрометації потоку.
P-value	Ймовірність отримати спостережувані дані або ще більш екстремальні результати, якщо нульова гіпотеза є правдивою.
QuarterRound	Базова операція раунду в ChaCha, що послідовно виконує ARX-перетворення над чотирма 32-бітними словами стану.
RC4	Потоковий шифр на основі генератора псевдовипадкової послідовності байтів, історично застосовувався у SSL/WEP, нині вважається застарілим.

Salsa20	Потоковий ARX-шифр з 20/12/8 раундами, попередник ChaCha, без S-box, базується на додаванні, ротації, XOR.
Security margin	Запас безпеки за кількістю раундів відносно найкращих відомих атак; для ChaCha12 прийнятний у практичних IoT-сценаріях.
SIMD	Single Instruction Multiple Data, набір інструкцій для паралельної обробки даних, корисний на CPU з відповідною підтримкою.
State (стан)	Внутрішній масив слів у ChaCha розміром 512 біт, що оновлюється послідовністю Column/Diagonal раундів.
Throughput	Пропускна здатність шифрування, зазвичай у МБ/с, використовується як ключова метрика продуктивності.
TLS 1.3	Протокол транспортного рівня безпеки; ChaCha20-Poly1305 є одним зі стандартних наборів шифрів.
XOR	Побітова операція виключного АБО; базова операція ARX-конструкцій. Rotate left/right – циклічні зсуви слів на фіксовані константи, частина ARX-побудови ChaCha.

ВСТУП

Актуальність теми. Стрімке зростання кількості пристроїв Інтернету речей (IoT), вбудованих сенсорних вузлів та мобільних систем висуває жорсткі вимоги до криптографії: забезпечення конфіденційності і цілісності даних у середовищах з обмеженими ресурсами CPU, пам’яті та енергоспоживання є критично важливим для надійності сервісів та відповідності сучасним стандартам безпеки в мережевих протоколах і прикладних стеках. Класичні блочні шифри, оптимізовані під апаратні інструкції настільних процесорів, нерідко демонструють надмірну затримку або енергетичні витрати на ARM Cortex-M та подібних архітектурах, що знижує пропускну здатність і автономність вузлів, у той час як потікові ARX-конструкції на кшталт ChaCha демонструють кращу масштабованість на загальновживаних MCU без спеціалізованих інструкцій.

Дослідженнями оптимізації алгоритмів шифрування для застосування в малопотужних платформах займались Aumasson J., Bernstein D., Langley A., Schwabe P., Бірюков А., Качко О., Черненко Р. та інші. Водночас, вирішення задачі в контексті практичної оптимізації сучасних алгоритмів шифрування для застосування в IoT є актуальним завданням, що поєднує вимоги забезпечення криптостійкості в реальному часі та енергоефективності. Воно потребує систематичного переосмислення для забезпечення належної відповідності новітнім платформам.

Робота відповідає поточним пріоритетам захищених кіберфізичних систем і прикладної криптографії в контексті побудови надійних сервісів збору та передачі даних на обмежених апаратних платформах, узгоджуючись зі стандартними підходами до оцінювання продуктивності, затримки, використання пам’яті та енергетичного профілю для протоколів і застосунків IoT.

Мета дослідження полягає у розробці і експериментальному обґрунтуванні оптимізацій обраного потокового алгоритму шифрування для підвищення ефективності кіберзахисту на малопотужних обчислювальних

системах без погіршення криптостійкості і сумісності з сучасними протоколами й бібліотеками.

Завдання дослідження передбачають:

– порівняльний аналіз блочних і потокових сімейств (AES, DES/3DES, RC4, Salsa20/ChaCha);

– обґрунтування вибору цільового алгоритму;

– побудову профілю продуктивності та ресурсного споживання;

– проектування оптимізацій (розклад циклів, inlining, зменшення доступів до пам’яті, використання простих обчислювальних шаблонів ARX);

– реалізацію еталонної та оптимізованої версій на цільовій платформі;

– експериментальне порівняння реалізованої оптимізації за визначеними метриками (throughput, latency, CPU utilization, memory footprint та енергоспоживання).

Об’єкт дослідження є процес симетричного шифрування даних у системах з обмеженими обчислювальними ресурсами, орієнтованих на IoT та вбудовані застосунки реального часу.

Предметом дослідження є методи алгоритмічної та програмної оптимізації симетричного потокового шифру типу ChaCha, що впливають на пропускну здатність, затримку та енергоефективність у типових MCU середовищах.

Методи дослідження полягають у комплексному використанні аналітичного порівняльного підходу для оцінювання криптографічних властивостей і безпекових маржин (кількість раундів, стійкість до відомих класів атак), профілювання коду й мікробенчмарки на базі інструментарію .NET 6 і Benchmark-орієнтованих фреймворків, а також методів експериментального вимірювання затримки, пропускної здатності, завантаження процесора, використання пам’яті й енергоспоживання для базової та оптимізованої реалізацій.

Додатково застосовано практичні методики інженерії продуктивності для ARX-конструкцій: мінімізація гілкувань, оптимізація доступів до пам’яті,

уніфікація операцій додавання-ротації-XOR та коректна організація кроків QuarterRound при скороченій кількості раундів.

Наукова новизна отриманих результатів передусім полягає в тому що було запропоновано цільові оптимізації реалізації ChaCha12 для малопотужних платформ, що забезпечують покращення пропускної здатності та затримки за рахунок структурних перетворень коду без зміни криптографічної схеми та параметрів безпеки. Обґрунтовано вибір ChaCha12 як балансу між криптостійкістю і витратами, показано зниження обчислювальної вартості QuarterRound з утриманням необхідного безпекового запасу у сценаріях IoT передачі потокових даних.

Практичне значення результатів. Розроблені підходи дозволяють інтегрувати оптимізовану реалізацію симетричного шифру у прикладні стеки на вбудованих платформах, підвищуючи автономність вузлів та стабільність сервісів при обмежених ресурсах. Отримані результати можуть бути використані у бібліотеках прикладної криптографії, системах безпечної телеметрії і відеопотоків на MCU, а також у навчальних курсах з оптимізації алгоритмів на вбудованих системах.

Апробація результатів роботи. Основні теоретичні та практичні результати дослідження доповідалися та обговорювалися на XVIII Всеукраїнській науково-практичній конференції «Інформаційні технології в професійній діяльності» (м. Рівне, 10 листопада 2025 р.)

Структура роботи. Дипломна робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел та 10 додатків. Загальний обсяг роботи становить __ сторінки. Вона містить 5 рисунків. Список використаних джерел включає 41 найменування. Обсяг додатків – __ сторінок**.**

РОЗДІЛ 1
ШИФРУВАННЯ ТА ВИБІР АЛГОРИТМУ ДЛЯ ОПТИМІЗАЦІЇ В МАЛОПОТУЖНИХ СИСТЕМАХ

1.1 Поняття криптографічного захисту інформації

Криптографія – наука про розробку та впровадження методів захисту інформації від несанкціонованого доступу, зміни та знищення. [40] Назва походить від грецьких слів "kryptos" (прихований) та "graphia" (письмо), що означає «приховане письмо». Сучасна криптографія – це не просто приховування тексту, а математична дисципліна із застосуванням складних обчислювальних алгоритмів і теорії чисел.

Головними принципами захисту інформації є чотири базові концепції: конфіденційність, цілісність, автентичність та нерозпізнаність. [35] Ці принципи є фундаментом для забезпечення безпеки в цифрових системах, особливо в умовах появи малопотужних обчислювальних пристроїв, таких як IoT.

Забезпечення криптографічного захисту інформації здійснюється шляхом шифрування (процесу перетворення відкритого тексту у зашифрований за допомогою криптографічного алгоритму та секретного ключа) та дешифрування (зворотнього процесу, що відновлює оригінальний текст за допомогою ключа). При цьому безпека системи ґрунтується не на таємниці алгоритму, а на секретності ключа, що визначено принципом Керкгоффса.

Основними компонентами функціонування криптосистеми зазвичай вистувають:

–алгоритм шифрування – математична операція, публічна;

–ключ – конфіденційна інформація, що контролює шифрування;

–відкритий текст – повідомлення, що потрібно захистити;

–шифротекст – результат шифрування;

–криптоаналітик – особа або система, що намагається розкрити ключ без дозволу.

Практичне застосування шифрування дуже широке і включає захист інтернет-комунікацій (HTTPS, TLS), забезпечення конфіденційності персональних даних (GDPR, CCPA), безпеку платіжних операцій згідно стандартів PCI DSS, мобільний захист (iOS, Android) [8], хмарні сховища з наскрізним шифруванням [29], корпоративний захист за допомогою VPN та інші сервіси.

Застосування шифрування на малопотужних пристроях підвищує вимоги до алгоритмів, які мають бути безпечними, енергоефективними та одночасно швидкими.

Криптографічні алгоритми поділяються на симетричні, асиметричні та гібридні. Їх вибір залежить від способу використання ключів, структури та мети використання.

Симетричні алгоритми. застосовують один секретний ключ і для шифрування, і для дешифрування. Найпоширеніші серед них – блокові та потокові шифри.

–Блокові шифри (DES, AES) працюють з фіксованими розмірами блоків інформації (зазвичай 64 або 128 біт). Вони застосовують багатоетапні трансформації для ускладнення структури даних.

–Потокові шифри (RC4, Salsa20, ChaCha20) генерують псевдовипадковий потік байтів, з яким біт за бітом чи байт за байтом комбінують відкритий текст через операцію XOR. Вони підходять для шифрування даних довільної довжини в режимі реального часу.

Симетричне шифрування відоме високою швидкістю й низькою обчислювальною складністю, що робить його особливо привабливим для обмежених ресурсів.

Асиметричні шифри використовують пару ключів: публічний для шифрування і приватний для дешифрування. Це дозволяє безпечно обмінюватись зашифрованими даними навіть без попереднього узгодження ключів. [41]

Найпопулярніший алгоритм – RSA, заснований на складності факторизації великих чисел, та ECC (еліптичні криві), що забезпечує високий рівень безпеки при менших розмірах ключів.

Для поєднання переваг високої швидкості симетричних алгоритмів і зручності управління ключами асиметричних систем використовується гібридний підхід. Асиметрична компонента шифрування застосовується для обміну симетричним ключем, а симетрична – власне для шифрування великих обсягів даних.

Оскільки симетричне шифрування базується на використанні одного секретного ключа, це забезпечує високу швидкість обробки, низьку обчислювальну складність та ефективність при роботі з великими обсягами інформації. Проте породжує проблему безпечного розподілу ключа між сторонами, що ускладнює масштабування системи.

В той час як асиметричне шифрування використовує пару ключів: публічний для шифрування і приватний для дешифрування, що дозволяє безпечно передавати інформацію, не потребуючи обміну секретним ключем заздалегідь. Проте їх застосування потребує від асиметричних алгоритмів суттєво вищих обчислювальних витрат. Як наслідок вони працюють повільніше.

У порівнянні обох методів очевидними є різниці у продуктивності, керуванні ключами та масштабованості. Щоб оцінити основні параметри, розглянемо такі характеристики:

–кількість ключів: у симетричній системі використовується один ключ, а в асиметричній – пара ключів.

–швидкість шифрування: симетричні алгоритми значно швидші, що робить їх придатними для шифрування великих обсягів даних.

–розмір ключа: у асиметричних алгоритмах ключі значно більші, що впливає на обчислювальні витрати.

–управління ключами: у симетричних системах складніше організувати безпечний розподіл, натомість у асиметричних це здійснюється зручніше.

–масштабованість: асиметричні системи мають кращу масштабованість завдяки простоті розповсюдження публічних ключів.

Щоб поєднати переваги обох методів, у практиці часто застосовують гібридний підхід – асиметричне шифрування використовується для безпечного обміну симетричним ключем, а симетричне – для шифрування основних даних. [28] Цей підхід широко використовується, зокрема, у протоколах TLS та інших системах безпеки.

1.2 Популярні алгоритми шифрування

Розвиток криптографії пов’язаний із постійними ускладненнями алгоритмів і їх оптимізацією. Зі зростанням потужності комп’ютерів збільшуються вимоги до стійкості алгоритмів, що зумовлює їх постійний розвиток і вдосконалення.

Блоковий алгоритм DES (Data Encryption Standard) був першим загальновизнаним стандартом електронного шифрування, введеним у 1977 році. Проте початковий 56-бітний ключ DES став недостатньо захищеним через розвиток обчислювальних потужностей, що призвело до його поступового витіснення. [27]

Йому на заміну прийшов алгоритм AES (Advanced Encryption Standard) з ключами 128, 192 і 256 біт. Він забезпечує високу криптостійкість і порівняно швидку роботу, особливо з апаратним прискоренням, що робить його провідним блоковим шифром сьогодення. [26]

Потоковий алгоритм RC4 був популярним через простоту та швидкість, але нині визнаний небезпечним і не рекомендується. Salsa20 розроблений як альтернатива RC4 із використанням операцій додавання, циклічного зсуву і XOR, що забезпечує високу швидкість та криптостійкість. ChaCha20 – покращена версія Salsa20, яка краще захищена і отримала широке визнання, включно з використанням у протоколі TLS, OpenSSH і популярних месенджерах. [4]

1.2.1 Порівняння характеристик сучасних алгоритмів

Ключові параметри криптографічних алгоритмів – це розмір блоку, довжина ключа, кількість раундів, тип структури та вимоги до обчислювальних ресурсів. Від них залежить як безпека, так і ефективність.

AES став стандартом у 2001 році, працює з 128-бітними блоками та підтримує ключі довжиною 128, 192 або 256 біт, виконуючи відповідно 10, 12 або 14 раундів. Його конструкція базується на заміні байтів, перестановці, змішуванні стовпців і додаванні раундового ключа, що забезпечує високу криптостійкість. [38] AES має апаратне прискорення, що значно покращує продуктивність у сучасних процесорах.

Salsa20 та ChaCha20 – потокові алгоритми, що обробляють 512-бітні блоки із застосуванням операцій додавання, циклічного зсуву та XOR (конструкція ARX). Обидва виконують 20 раундів, але ChaCha20 має покращену дифузію і більшу стійкість до криптоаналізу.

Щодо продуктивності, AES є більш ефективним на системах із апаратним прискоренням (AES-NI) [31], тоді як ChaCha20 часто перевершує його на мобільних і вбудованих пристроях, де апаратна підтримка відсутня.

Ці відмінності роблять вибір алгоритму залежним від конкретних умов використання – для серверів із високою продуктивністю оптимальним є AES, а для малопотужних пристроїв – ChaCha20.

1.2.2 Проблеми криптостійкості алгоритмів шифрування

Криптостійкість – це здатність алгоритму протистояти ефективним атакам із використанням відомих методів криптоаналізу й доступних ресурсів. Ключові фактори, що впливають на криптостійкість, включають розмір ключа, якість протоколу, математичні основи та ступінь аналізу в криптографічній спільноті. Рівні криптостійкості варіюються від слабкої (наприклад, DES), через прийнятну до сильної й дуже сильної (AES-256, ChaCha20). [6] Вибір залежить від вимог до захисту та передбачуваного часу зберігання конфіденційності даних.

Запас криптостійкості – це різниця між реальною стійкістю алгоритму та поточним рівнем захисту, який враховує можливі нові атаки. Міжнародні

стандарти, зокрема NIST, ISO і IETF, визначають рекомендації щодо вибору алгоритмів, довжин ключів та режимів шифрування. Вони постійно оновлюються, враховуючи розвиток технологій і загроз. Особливої уваги заслуговує постквантова криптографія, що розробляє алгоритми, стійкі до атак квантових комп'ютерів. [24] NIST ініціював конкурс на стандартизацію таких алгоритмів, що є важливою складовою майбутньої безпеки. [17]

Практичні рекомендації включають використання AES-128 або ChaCha20 для більшості сучасних систем, AES-256 для максимальної захищеності, а також поступове планування міграції на постквантові технології. [7]

1.3 Особливості малопотужних обчислювальних систем

Малопотужні обчислювальні системи – це класи пристроїв, які відрізняються суворими обмеженнями щодо обчислювальних, пам'яттєвих та енергетичних ресурсів. Вони охоплюють широкий спектр обладнання, від простих вбудованих мікроконтролерів, таких як ARM Cortex-M0/M3, до складніших, але все ж обмежених пристроїв IoT.

Часто ці системи розробляються зі спеціально орієнтованою архітектурою для виконання вузько спрямованих функцій. У таких конфігураціях ресурси виділяються раціонально і мають бути максимально ефективно задіяні. Обмеження в продуктивності, мінімальні затрати енергії та обмежена пам'ять визначають специфіку програмних рішень, що розгортаються на таких платформах.

Ці пристрої часто працюють у режимі дійсно низького енергоспоживання – протягом значної частини часу знаходяться у стані сну, пробуджуючись лише для збору та передачі даних або реагування на події. Тому кожен цикл обчислень і кожен байт енергії мають критичне значення. Особливістю малопотужних систем є відсутність апаратних прискорювачів криптографічних операцій, таких як AES-NI або подібних. Через це класичні блокові шифри, що вимагають складних арифметичних обчислень і операцій над таблицями заміщення, часто працюють недостатньо швидко і енергоефективно. Натомість,

алгоритми, які базуються на простих операціях додавання за модулем, незакончених циклічних зсувів і операції XOR (відома як ARX – Addition-Rotation-XOR), ідеально підходять для реалізації в таких умовах. Вони є менш ресурсоємними, простими у реалізації і при цьому забезпечують високий рівень безпеки. [33] Це позиціює сімейство ChaCha, зокрема його модифікацію ChaCha12, як оптимальний вибір для малопотужних обчислювальних систем. Алгоритми цього класу демонструють високу продуктивність навіть на системах із обмеженим часом обробки та пам’яттю, забезпечуючи при цьому належний рівень криптографічної безпеки.

Поточним викликом у цій галузі є балансування між безпекою та ефективністю – більш висока кількість раундів і більший розмір ключа забезпечують більшу стійкість проти атак, але водночас знижують продуктивність і збільшують енергоспоживання. Саме тому дослідження оптимізації ChaCha20 до ChaCha12 є важливою задачею для практичних застосувань у малопотужних системах.

Ефективна реалізація алгоритмів на малопотужних платформах вимагає також уваги до структури коду, обмеження розміру, мінімізації використання стеку та інших ресурсів, що безпосередньо впливають на стабільність роботи пристрою, особливо в умовах обмежених апаратних можливостей.

1.3.1 Обмеження IoT-пристроїв та вбудованих систем

IoT-пристрої та вбудовані системи є типовими прикладами малопотужних обчислювальних систем, на яких базується безліч сучасних технологічних рішень. Ці пристрої, як правило, оснащені мікроконтролерами з частотою від кількох мегагерц до кількасот мегагерц і мають обмежений розмір пам’яті – зазвичай декілька сотень кілобайт оперативної пам’яті та кілька мегабайт флеш-пам’яті.

Для таких пристроїв характерно використання автономного живлення – батарей або елементів живлення з обмеженим ресурсом. Це робить критично важливою оптимізацію всіх обчислювальних процесів, зокрема криптографії, що часто має значну обчислювальну та енергетичну складову. Вбудовані

системи, часто повинні функціонувати роками без заміни батарей, що підвищує вимоги до енергозбереження і надійності алгоритмів захисту даних. Апаратне забезпечення таких пристроїв часто позбавлене спеціалізованих криптоприскорювачів, таких як AES-NI у сучасних процесорах, що вимагає від алгоритмів високої продуктивності при мінімальній складності реалізації. [34]

У багатьох IoT-застосуваннях пристрої працюють у мережах з низькою пропускною здатністю та підвищеною вразливістю до атак, що робить необхідним використання криптографії з адекватними характеристиками безпеки. [25]

Великою перевагою таких пристроїв є можливість адаптації криптографічних алгоритмів до специфіки архітектури, що забезпечує баланс між рівнем безпеки та ресурсозбереженням. Це зумовлює обґрунтованість вибору алгоритмів зі спрощеними операціями, таких як ChaCha12, що і є предметом дослідження цієї роботи.

Обчислювальна потужність малопотужних пристроїв зазвичай визначається тактовою частотою їхніх процесорів, що коливається в межах від декількох мегагерц до кількох сотень мегагерц. На відміну від стаціонарних комп’ютерів та серверних платформ, такі пристрої працюють на одинарних або обмежених ядрах, що прямо впливає на швидкість обробки даних та затримки виконання алгоритмів. Обмежений обсяг корзини обчислювальних циклів змушує проектувати алгоритми із максимально простою логікою та мінімумом інструкцій.

Пам’ять, що часто дефіцитна в подібних системах, поділяється на дві основні категорії: оперативна пам’ять (RAM) і пам’ять зберігання коду (flash). RAM обмежує обсяг тимчасових даних та буферів, що можуть оброблятися безпосередньо, а flash визначає розмір програмного коду та статичних констант, які можна розмістити в системі. Підхід до компромісу полягає у спрощенні структури алгоритму, економному використанні буферів і відмові від масштабних таблиць підстановок, які характерні для класичних блокових шифрів.

Для досягнення правомірного балансу між безпекою та ресурсними обмеженнями часто застосовують алгоритми сімейства ARX (Addition-Rotation-XOR), які базуються на простих операціях, що швидко виконуються на малопотужних архітектурах та характеризуються низькими вимогами до пам’яті. [16] В противагу цьому, алгоритми, що використовують складні операції, множення або великі підстановочні таблиці, є менш придатними.

Загалом, обмеження ресурсів у малопотужних системах викликають необхідність ретельного планування реалізації криптографічних засобів із фокусом на мінімізацію використання процесорного часу, пам’яті та енергоспоживання, тобто забезпечують баланс між безпекою і ефективністю.

З огляду на це, багато сучасних досліджень спрямовані на розробку та оптимізацію потокових шифрів, таких як ChaCha20 та його скорочені варіанти (ChaCha12, ChaCha8), які надають достатню криптографічну стійкість при знижених затратах обчислювальних ресурсів. Такі алгоритми, базовані на простих операціях, здатні забезпечувати швидке шифрування і дешифрування навіть на обладнанні з обчислювальною потужністю у межах десятків або сотень мегагерц.[15]

1.3.2 Вимоги до алгоритмів для IoT та вбудованих систем

Алгоритми, що використовуються в IoT-пристроях та вбудованих системах, мають відповідати цілим наборам специфічних вимог, які відображають особливості апаратної платформи, умови експлуатації та архітектурні обмеження цих пристроїв. Об’ємна і комплексна природа цих вимог визначає вибір і подальшу оптимізацію криптографічних алгоритмів для малопотужних систем.

Передусім, алгоритми повинні бути високоефективними з точки зору обчислювальної складності. Це означає, що операції мають бути максимально простими, що дозволяє зменшити час обробки і відповідно енергоспоживання. Простота кодувань, уникнення операцій множення, зведення до мінімуму взаємодії з пам’яттю і використання ефективних арифметичних операцій

(додавання, XOR, циклічні зсуви) є необхідними для оптимальної реалізації на обмеженому обладнанні.

Крім продуктивності та розміру коду, важливим фактором є безпека. Алгоритми мають забезпечувати мінімальні рамки криптографічної стійкості, що адекватні для даного класу пристроїв і застосувань. Вибір варіантів алгоритмів з меншим числом раундів, наприклад ChaCha12 замість ChaCha20, опирається на компроміс між безпекою і продуктивністю, який підлягає ретельній криптоаналітичній оцінці.

Додатково алгоритми повинні мати можливість масштабуватись та адаптуватись до різних апаратних платформ, а також підтримувати широкий діапазон параметрів, таких як довжина ключа і кількість раундів. [32] Це дозволяє застосовувати один і той самий алгоритм у різних пристроях із відповідним налаштуванням, що зручно як для розробників, так і для кінцевих користувачів.

Одним із ключових критеріїв є надійність реалізації. Алгоритми повинні бути простими для верифікації і не допускати складних реалізацій з високим ризиком помилок, що особливо важливо у вбудованих системах із суворими вимогами до надійності. [12]

Узагальнюючи, алгоритми криптографії для IoT та вбудованих систем мають поєднувати в собі високу продуктивність, мінімальні вимоги до ресурсів, достатній рівень безпеки та адаптивність до різноманітних платформ. Саме такі критерії визначають сучасні тренди у розробці та оптимізації легких потокових алгоритмів, які забезпечують ефективний захист даних без перевантаження малопотужного апаратного забезпечення. [22]

1.4 Критерії вибору алгоритму для оптимізації

Вибір криптографічного алгоритму для оптимізації в малопотужних системах є складним завданням, що вимагає аналізу низки взаємозалежних факторів. На відміну від традиційних комп’ютерних систем, де пріоритетом є

максимальна безпека, у вбудованих пристроях необхідно збалансувати безпеку, продуктивність, використання пам’яті і енергоспоживання.

Підвищення безпеки зазвичай вимагає збільшення розмірів ключів і кількості раундів, що підвищує обчислювальні витрати. Водночас, максимальна ефективність досягається спрощенням алгоритму, що потенційно знижує стійкість. Критичною проблемою є обмежені ресурси малопотужних пристроїв – оперативна пам’ять, процесорна потужність і енергія. Тому ключова мета – знайти оптимальний баланс між всіма цими обмеженнями.

Тому до критеріїв вибору крипто алгоритмів включають:

– продуктивність;
– обчислювальну складність;
– архітектуру алгоритму;
– енергоспоживання;
– використання пам’яті;
– рівень криптостійкості;
– перспективи подальших атак.

1.4.1 Продуктивність та обчислювальна складність

У контексті малопотужних пристроїв продуктивність криптографічного алгоритму є критичним параметром, що визначає практичність його застосування. Вона характеризується такими основними папрмертами:

–пропускна здатність, що вимірюється у мегабітах або мегабайтах на секунду, показує, скільки даних криптоалгоритм здатен обробити за одиницю часу. Для малопотужних систем важливо досягати максимальної пропускної здатності з урахуванням обмежених ресурсів.

–затримка – це час, який потрібен на обробку одного блоку або повідомлення, суттєвий параметр для систем реального часу.

–енергоефективність – визначає кількість енергії, яка витрачається на обробку одиниці даних, що є важливим для пристроїв із батарейним живленням.

Обчислювальна складність алгоритму – це теоретична оцінка, що показує, як швидкість виконання змінюється залежно від розміру вхідних даних. Бажано, щоб ця складність була близькою до лінійної (O(n)), що означає пропорційне зростання часу обробки із збільшенням обсягу даних. Структура алгоритму безпосередньо впливає на продуктивність: кількість раундів визначає глибину обробки, де більша кількість раундів підвищує безпеку, але зменшує швидкодію. Операції, що складають алгоритм, мають різну вартість на малопотужних процесорах. Наприклад, арифметичні та логічні операції (XOR, додавання, циклічний зсув) значно швидші за операції з таблицями підстановки (S-box) або множення.

Паралелізм теж є важливим із точки зору оптимізації. Потокові алгоритми, такі як ChaCha20, серед своїх переваг мають можливість паралельної генерації блоків ключового потоку, що може бути використано у багатоядерних системах для збільшення продуктивності. [37]

Розглядаючи практичні показники, на малопотужних пристроях варто віддавати перевагу алгоритмам з низькими затримками та мінімальними обчислювальними витратами, навіть якщо це може дещо знизити максимальну пропускну здатність.

1.4.2 Простота реалізації та портативність коду

Простота реалізації криптографічного алгоритму є ще одним ключовим фактором при виборі для малопотужних обчислювальних систем. Легкість розуміння структури, мінімальна складність коду та відсутність складних операцій значно сприяють зниженню часу розробки, а також зменшенню кількості помилок під час впровадження. Крім того, важливий аспект – відсутність апаратної залежності, що дозволяє реалізовувати алгоритми на широкому спектрі пристроїв. Особливо це актуально для IoT, вбудованих систем та мобільних платформ з різноманітними архітектурами.

ChaCha20 базується на ARX-конструкції, тобто використовує прості операції додавання за модулем 2^{32}, циклічного зсуву та операції XOR. Такий підхід уникає складних таблиць підстановок (S-блоки), які присутні в

алгоритмах типу AES і вимагають більшої пам’яті та ресурсів. Відсутність S-блоків робить код компактним і детермінованим, що полегшує відлагодження й оптимізацію. Портативність реалізації забезпечується також тим, що ARX-операції однаково ефективно виконуються на різних типах процесорів, у тому числі ARM, RISC-V, x86. Відсутність апаратних прискорень не критична, що робить ChaCha20 універсальним рішенням для широкого спектру пристроїв.

Бібліотеки зі стандартними реалізаціями ChaCha20 доступні на багатьох мовах програмування, включно з C, C++, C#, Java, Python. Це значно полегшує інтеграцію алгоритму в різноманітні проектах і знижує час адаптації. На відміну від ChaCha20, AES, окрім більш складної структури, є чутливим до різних архітектурних особливостей. Його ефективність сильно покладається на апаратне прискорення, таке як AES-NI в процесорах Intel. У відсутності апаратної підтримки реалізація AES може стати ресурсозатратною, особливо на малопотужних пристроях.

Отже, вибір алгоритму із простою структурою і хорошою портативністю впливає не лише на продуктивність, а й на надійність, економію часу розробки і можливість використання в різноманітних апаратних середовищах.

1.4.3 Енергоефективність та споживання пам’яті

Енергоефективність є важливим критерієм для малопотужних обчислювальних систем, особливо у пристроях з автономним живленням, таких як IoT, мобільні пристрої та вбудовані системи. Оптимізація алгоритму з урахуванням енергоспоживання дозволяє значно збільшити час роботи пристрою без заміни батарей або підзарядки.

Основним джерелом енергоспоживання є обчислювальна активність процесора і доступ до пам’яті. Тому при оптимізації важливо не лише зменшувати кількість операцій, але і скорочувати витрати на доступ до пам’яті, оскільки операції читання/запису значно дорожчі за арифметичні.

ChaCha20 завдяки ARX-конструкції (додавання, циклічний зсув, XOR) використовує лише прості операції, які виконуються швидко і з мінімальними енергетичними витратами. Відсутність складних підстановок і множень

додатково знижує навантаження на процесор. Ефективне використання регістрів, уникнення частих звернень до пам’яті, розгортання циклів і використання інлайнингу функцій – це прийоми, які зменшують кількість циклів і, відповідно, енергоспоживання. Споживання оперативної пам’яті також впливає на енергоефективність і загальну продуктивність. ChaCha20 має компактний стан – 512 біт (16 слів по 32 біти), що спрощує зберігання і обробку як в оперативній пам’яті, так і в регістрах. Порівняно із AES, який вимагає великих таблиць S-блоків, потреба у великому об’ємі пам’яті у ChaCha20 є значно меншою, що позитивно позначається на ресурсах пристрою.

Вимірювання енергоспоживання на ARM Cortex-M4 показують, що ChaCha12 споживає приблизно в 1.5 раза менше енергії на обробку одиниці даних у порівнянні з ChaCha20, що робить її кращою для тривалих завдань із обмеженим живленням. [2]

Таким чином, з урахуванням енергоефективності та оптимального використання пам’яті, варіанти ChaCha є найкращим вибором для застосування в малопотужних обчислювальних системах, забезпечуючи баланс між безпекою та ресурсною ефективністю.

1.5 Обґрунтування вибору сімейства ChaCha20 для оптимізації

При виборі алгоритму для впровадження в малопотужних обчислювальних системах особливу увагу приділено сімейству ChaCha20. Основними аргументами на користь цього вибору є баланс між безпекою, продуктивністю, ефективністю використання ресурсів та простотою реалізації. Сімейство ChaCha20, розроблене Деніелем Бернштейном, є модифікацією Salsa20 з поліпшеною дифузією і криптографічною стійкістю. Воно включає різні варіанти – ChaCha20, ChaCha12 та ChaCha8 – що різняться кількістю раундів, дозволяючи обирати компроміс між швидкістю і безпекою.

ChaCha20 використовує ARX-конструкцію (операції додавання за модулем 2^{32}, циклічного зсуву і XOR), що забезпечує високу швидкість на

різних апаратних архітектурах, зокрема на малопотужних і мобільних пристроях, де відсутнє спеціальне апаратне прискорення.

Криптостійкість ChaCha20 доведена численними дослідженнями, а її варіанти зменшеної кількості раундів (ChaCha12 та ChaCha8) показують достатній запас безпеки для багатьох практичних застосувань.

Одним із ключових факторів вибору є ефективне використання оперативної пам’яті – робочий стан ChaCha20 займає 512 біт, що дозволяє реалізовувати алгоритм у компактному вигляді. Простота алгоритму спрощує написання портативного коду і робить його легко адаптованим для різноманітних платформ та мов програмування. Практичні випробування демонструють, що ChaCha12 забезпечує значне покращення продуктивності в порівнянні з ChaCha20, із мінімальними втратами криптостійкості, що є важливим для обмежених систем.

Враховуючи сучасні тенденції стандартизації (RFC 7539, TLS 1.3) [13] та широке комерційне впровадження ChaCha20, вибір цього сімейства є обґрунтованим з огляду на як технічні, так і практичні аспекти.

Таким чином, оптимізація сімейства ChaCha20, зокрема версії ChaCha12, відповідає стратегічним вимогам безпеки і ефективності для малопотужних обчислювальних систем.

1.5.1 Характеристики алгоритмів сімейства ChaCha

Спроектований Деніелем Бернштейном у 2005 році Salsa20 – це потоковий шифр, який став сучасною і безпечною альтернативою RC4. Він заснований на принципі генерації псевдовипадкового ключового потоку для операції XOR із відкритими даними, забезпечуючи значну швидкість на різних архітектурах. Архітектура Salsa20 побудована на матриці 4×4 із 32-бітних слів, які підлягають перетворенням через 20 раундів операцій додавання, циклічного зсуву та XOR. Раунди сприяють високій дифузії та стійкості до відомих атак. Алгоритм не містить таблиць підстановок типу S-блоків, що позитивно впливає на продуктивність і портативність. Salsa20 підтримує два варіанти ключів – 128

біт і 256 біт, а також різне число раундів (20, 12, 8) для балансування між швидкодією й рівнем безпеки.

ChaCha20 є еволюцією Salsa20, розробленою для подальшого підвищення криптостійкості й поліпшення дифузії. Головна відмінність ChaCha20 полягає в зміненому чергуванні обробки рядків та стовпців у матриці, що посилює зв'язок між бітами і ускладнює криптоаналіз. ChaCha20 використовує 20 раундів для гарантії максимальної стійкості до атак. Алгоритм має розширений до 96 біт nonce, що дозволяє використовувати його в практиках автентифікованого шифрування (AEAD) разом із Poly1305. [1] Усі операції (додавання, зсув, XOR) однаково швидко виконуються на різних процесорних архітектурах, що робить впровадження ChaCha20 зручним для програмістів.

Як Salsa20, так і ChaCha20 відзначаються компактністю реалізації, незалежністю від апаратного прискорення і витривалістю до атак на побічні канали інформації. Вони використовуються у багатьох сучасних стандартах безпеки – OpenSSH, TLS 1.3, WireGuard, месенджері Signal [19-21] – і оптимально підходять для вбудованих пристроїв, IoT і мобільних платформ, де продуктивність та енергозбереження критично важливі.

Практичні тести та численні криптоаналітичні дослідження показали відсутність ефективних атак на повні раунди обох алгоритмів. Це пояснює їх стандартизацію (RFC 8439 для ChaCha20, eSTREAM для Salsa20) і зростаючу популярність у застосунках з підвищеними вимогами до швидкодії та гнучкості реалізації. [18]

1.5.2 Переваги ChaCha20 порівняно з AES для малопотужних систем

ChaCha20 має суттєві переваги перед AES у контексті забезпечення балансу між безпекою, продуктивністю та ефективним використанням ресурсів, зокрема на пристроях з обмеженими обчислювальними потужностями та відсутністю апаратного прискорення AES.

ChaCha20 базується на арифметико-логічних операціях (додавання за модулем 2^{32} , циклічні зсуви та XOR), які однаково швидко виконуються на різних процесорах. Відсутність таблиць підстановок (S-блоків) робить його

реалізацію простою, мінімізує кеш-промахи і підвищує стійкість до таймінгових атак.

У порівнянні, AES включає складні функції S-блоків і множення в полі Галуа, що збільшує обчислювальні затрати на платформах без спеціалізованих інструкцій (AES-NI). Таких інструкцій немає або вони обмежені в малопотужних пристроях, що погіршує продуктивність AES.

ChaCha20 має архітектуру, що дозволяє паралельну обробку блоків, оптимізуючи використання багатоядерних систем і сучасних SIMD інструкцій. Це підвищує продуктивність у порівнянні з поперечно-блоковими режимами AES, які часто мережені і не співпадають з паралельними можливостями. [11]

Крім того, ChaCha20 забезпечує більш передбачуване енергоспоживання та нижче затримки при цьому, що є критичним для мобільних і вбудованих систем із обмеженим живленням.

Загалом, для систем із низькою обчислювальною потужністю, де надмірна складність реалізації може призвести до помилок і високої затримки, сімейство ChaCha20 пропонує оптимальний вибір, поєднуючи високу криптостійкість із спрощеним апаратним і програмним підтримкою.

1.5.3 Аналіз оптимізацій алгоритмів ChaCha

Оптимізація алгоритму ChaCha має довгу історію численних досліджень, спрямованих на підвищення продуктивності без зниження криптостійкості. Основні напрямки включають зменшення кількості раундів, використання ефективних технік програмування та архітектурних особливостей сучасних мікропроцесорів.

Дослідники звернули увагу на модифікації ChaCha20 із меншою кількістю раундів, зокрема ChaCha12 і ChaCha8. Вони демонструють суттєве прискорення – до 77% і 87% відповідно при збереженні адекватного рівня безпеки для багатьох завдань.

Програмні оптимізації, такі як розгортання циклів, інлайнинг функцій і оптимізація доступу до регістрів, покращують ефективність коду.

Використання SIMD інструкцій і багатоядерних архітектур дозволяє додатково підвищити продуктивність у відповідних середовищах.

Важливим напрямом є вимірювання енергоспоживання, що особливо критично для мобільних і вбудованих систем. Експерименти показують, що ChaCha12 споживає на 35-40% менше енергії, ніж ChaCha20, а ChaCha8 – навіть на 60-70% менше, що робить ці варіанти привабливими для IoT-пристроїв. [9]

Крім того, експериментальні дослідження підтверджують зниження енергоспоживання оптимізованої версії ChaCha12 відносно стандартного ChaCha20 на 30-40% [9], що є критично важливим фактором для пристроїв з автономним живленням. Водночас зменшення числа раундів не веде до значущої втрати криптографічної стійкості, що підтверджено чисельними криптоаналітичними дослідженнями.

Завдяки високій портативності коду та простоті реалізації алгоритму забезпечується легка інтеграція у різноманітні середовища, що особливо важливо для гетерогенних IoT-екосистем.

Апробація технологій підтверджує відповідність розроблених методів сучасним галузевим стандартам безпеки та вимогам до енергоефективності, що робить результати роботи цінними для промислових інтеграторів та розробників вбудованих систем.

Також відбулися практичні добування – реалізації ChaCha12 у мові C# інтегровані із системами зчитування відео з веб-камери, що підтверджує їхню придатність для реального застосування.

Експериментальні порівняння показали, що зменшення кількості раундів для ChaCha12 й ChaCha8 дозволяє значно покращити продуктивність за умови прийнятної безпеки, що підтверджується практичними результатами та відгуками криптоаналітичної спільноти.

Таким чином, проведені дослідження створюють наукову базу для впровадження оптимізованих варіантів ChaCha у малопотужних системах, забезпечуючи ефективний баланс між безпекою, швидкодією та енергоспоживанням. Отже, запропонована оптимізація не тільки науково

обґрунтована, а й має практичне значення для підвищення безпеки та ефективності малопотужних систем, розширюючи можливості їх застосування у різних сферах сучасних технологій.

Висновки до розділу 1

Сукупний аналіз теорії, архітектурних обмежень IoT і практичних порівнянь показує, що для малопотужних систем найдоцільнішим є застосування легких потокових алгоритмів ARX-класу, де оптимізований варіант ChaCha12 забезпечує кращий баланс швидкодії, енергоефективності та достатньої криптостійкості, зберігаючи стандартизовану сумісність у TLS/RFC.

Сучасна криптографія спирається на строгі математичні основи і потребує адаптації під обмеження енергії, пам’яті та обчислювальної потужності у вбудованих пристроях, що зумовлює перевагу простих реалізацій без важких таблиць і множень.

Симетричні алгоритми продуктивніші, однак питання керування ключами робить гібридні схеми доцільними для протоколів розподілу ключів, після чого дані шифруються швидкими потоковими шифрами.

AES залишається стандартом для потужних платформ і середовищ з AES-NI/NEON, тоді як ChaCha20 демонструє переваги на CPU без апаратного прискорення; [10] для IoT і мобільних девайсів це часто вирішальний фактор вибору.

Архітектура IoT диктує вимоги до малої пам’яті і низької енергії, тож ARX-конструкції (ADD, XOR, ROL) мінімізують вартість інструкцій і кеш-залежність, що напряму покращує пропускну здатність і енерговитрати.

За критеріями продуктивності, безпеки, простоти реалізації, портативності та енергоефективності скорочені версії ChaCha переважно виграють у середовищах без прискорювачів; при цьому ChaCha12 формує раціональний компроміс між запасом стійкості та швидкістю.

Вибір ChaCha20/ChaCha12 виправдано стійкістю до відомих атак, простотою аудиту коду і підтримкою у TLS 1.3 та RFC 7539, що полегшує інтеграцію в існуючі стекі та стандарти безпеки.

Для енергообмежених IoT/edge систем рекомендується ChaCha12 як базовий режим потокового шифрування, AES варто залишати для платформ з апаратним прискоренням або регуляторних вимог, а ChaCha8 допустимий лише у некритичних сценаріях з чіткими обмеженнями загроз.

РОЗДІЛ 2
ТЕОРЕТИЧНІ ОСНОВИ ОПТИМІЗАЦІЇ АЛГОРИТМУ CHACHA20

2.1 Детальний аналіз алгоритму ChaCha20

ChaCha20 – це сучасний симетричний потоковий шифр, який базується на послідовності елементарних арифметико-логічних операцій: додавання за модулем 2^{32}, циклічного зсуву і XOR (конструкція ARX). Основний стан представлений у вигляді матриці 4×4 слів по 32 біти, яка послідовно обробляється у 20 раундах. Такий підхід поєднує високу криптостійкість із значною швидкодією і є оптимальним для використання в малопотужних системах. Варіанти з меншою кількістю раундів (ChaCha12, ChaCha8) забезпечують кращу продуктивність при прийнятній стійкості, що особливо важливо для обмежених пристроїв.

2.1.1 Структура раундової функції

Фундаментальною операцією є QuarterRound – послідовність із 12 арифметичних і логічних інструкцій, що впливають на чотири 32-бітні слова стану матриці. Раунд ChaCha20 складається із чергування обробки колонок матриці (Column Round) та її діагоналей (Diagonal Round). У кожному раунді QuarterRound застосовується чотири рази для колонок, потім чотири рази для діагоналей. Такий режим забезпечує швидке розповсюдження змін в одному слові по всій матриці, що критично для криптографічної безпеки. Розподіл операцій по рядках і стовпцях поєднується в стійку раундову функцію, що важко піддається криптоаналізу.

2.1.2 Математичні операції та їх складність

ChaCha20 обмежується використанням операцій ARX: складання за модулем 2^{32}, побітового XOR і циклічного зсуву. Це дає переваги у швидкодії і енергоефективності, оскільки всі ці операції ефективно реалізуються на базовому інструктажі сучасних процесорів, зокрема в малопотужних і вбудованих системах. Відсутність складних операцій, таких як множення в полях Галуа або S-блоки, знижує затримки доступу до пам’яті і потребу у

великій кількості ресурсів. Такий дизайн робить алгоритм ідеальним для роботи на пристроях без апаратного прискорення, підтримуючи лінійну часову складність $O(n)$, що дозволяє обробляти великі потоки даних з мінімальними накладними витратами.

2.1.3 Ініціалізація стану та генерація ключового потоку

Ініціалізація стану починається з формування матриці 4×4 слів, яка складається з чотирьох частин: фіксованих констант, секретного ключа (256 біт), nonce (унікального 96-бітного номера сесії) і 32-бітного лічильника блоків. Початковий стан проходить через 20 раундів раундової функції, а потім формується ключовий потік довжиною 512 біт (64 байти). Цей ключовий потік використовується для шифрування відкритого тексту за допомогою побітового XOR. Висока унікальність значення nonce та лічильника блоків гарантує, що кожен блок зашифрований із неповторним ключовим потоком, що запобігає можливим атакам, пов’язаним із повторенням потоків. Така структура оптимально підходить для захисту потокових даних в режимах реального часу і є гнучкою для роботи з довільними обсягами інформації.

2.2 Оцінка безпеки алгоритму

Для оцінювання рівня безпеки алгоритму використовують методи криптоаналізу. Криптоаналіз – це наука та практика аналізу криптографічних систем з метою виявлення та експлуатації можливих вразливостей без знання ключа. Він охоплює методи математичного, статистичного та логічного аналізу з метою пошуку шляхів розкриття секретної інформації.

Для ChaCha20 та його попередника Salsa20, за більш ніж 15 років досліджень криптографічної спільноти, проведено ґрунтовний криптоаналіз, який довів їх високий рівень захисту від більшості відомих типів атак. Важливо розглянути класифікацію основних методів криптоаналізу, щоб оцінити реальні загрози.

2.2.1 Відомі атаки на сімейство ChaCha

Сімейство ChaCha, що походить від Salsa20, було об’єктом численних криптоаналітичних досліджень, які не виявили жодних практичних вразливостей на повний 20-раундовий алгоритм. Диференційний та лінійний криптоаналіз, що є основними методами атак на симетричні шифри, не дали результатів, які могли б скомпрометувати стандартні варіанти ChaCha20. Теоретичні атаки поки застосовані лише до зменшеної кількості раундів (від 3 до 7), але вони вимагають неприйнятно великих ресурсів і часу, що робить їх непридатними для практичного злому.

Атаки на побічні канали фізичної реалізації, такі як таймінгові чи енергозалежні, хоч потенційно можуть викликати ризики, еволюція алгоритму ChaCha і його компактні ARX-операції мінімізують ці вразливості. [30] Порівняно з іншими алгоритмами, ChaCha20 більш стійкий до кеш-атак через відсутність великих таблиць.

У контексті появи квантових обчислень ChaCha залишається відносно надійним, оскільки квантові атаки знижують стійкість симетричних ключів лише на половину їх розміру, що можна компенсувати збільшенням довжини ключа.

2.2.2 Оцінка запасу криптостійкості

У ChaCha20 із 20 раундами найкраща відома атака застосовна приблизно до 7 раундів, надаючи алгоритму запас криптостійкості у 13 раундів, що є значним резервом безпеки.

Цей буферний запас критично важливий для адаптивної стійкості проти майбутніх криптоаналітичних відкриттів. Для ChaCha12 (12 раундів) і ChaCha8 (8 раундів) запас зменшений, але все ще достатній для багатьох практичних додатків із середніми вимогами до захисту.

Вибір кількості раундів алгоритму – це завжди компроміс між продуктивністю і безпекою. Зменшення раундів покращує швидкість і знижує енергоспоживання, що вигідно для малопотужних пристроїв, проте зменшує запас криптостійкості.

У ChaCha20 20 раундів забезпечують максимальну безпеку, рекомендовану для критичних систем. Версії з 12 і 8 раундами оптимізовані для швидкості, але при цьому втрачають частину запасу безпеки. Такий вибір раундів рекомендується лише у випадках, коли компроміс між безпекою і продуктивністю є виправданим.

Дослідження показують, що кількість раундів нижче 8 суттєво підвищує ризик теоретичних атак, тому ChaCha8 слід використовувати з обережністю.

2.3 Обґрунтування зменшення кількості раундів

Оптимізація алгоритму ChaCha20 шляхом зменшення кількості раундів є логічним кроком для підвищення продуктивності та енергоефективності, що особливо актуально для малопотужних обчислювальних систем. Проте це вимагає ретельного аналізу внесеного компромісу між швидкодією та безпекою. Зменшення раундів впливає на запас криптостійкості, тому обґрунтування такого кроку базується на глибокому криптоаналітичному дослідженні, що підтверджує збереження достаточного рівня стійкості алгоритму при покращенні продуктивності.

ChaCha20 залишається найбільш криптографічно стійким варіантом, зі стандартними 20 раундами, які забезпечують максимальний рівень захисту. ChaCha12, із скороченою до 12 кількістю раундів, пропонує суттєве зростання продуктивності з відносно невеликою компромісною втратою безпеки. Це робить його оптимальним вибором для пристроїв з обмеженими ресурсами, де потреба у швидкодії та енергозбереженні критично важлива.

Криптоаналітичні дослідження підтверджують, що атаки на варіанти з більшою кількістю раундів мають значно вищу складність і поки що не проводяться на практиці в реальних умовах. Проте загальна рекомендація полягає в обережності при виборі варіанта із зниженим числом раундів для забезпечення надійної захищеності.

Визначення оптимальної кількості раундів є класичною задачею балансу між продуктивністю і безпекою. Збільшення числа раундів підвищує

криптографічний захист, забезпечуючи більш складний і стійкий механізм шифрування, але при цьому збільшує обчислювальне навантаження, енергоспоживання та затримки обробки.

На практиці для вбудованих та мобільних систем, де ресурси обмежені, зменшення кількості раундів дозволяє істотно підвищити продуктивність та економію енергії, при цьому необхідно оцінити ризики можливої втрати безпеки. Для ChaCha12 та ChaCha8 це компромісний варіант, який дозволяє більш гнучко адаптувати алгоритм під вимоги конкретних пристроїв і задач.

Міжнародні організації, зокрема IETF, NIST та інші стандартизуючі відомства, формують рекомендації щодо використання сімейства ChaCha в залежності від рівня безпеки і продуктивності. RFC 7539 офіційно описує ChaCha20 з 20 раундами як рекомендований стандарт для забезпечення надійного захисту даних.

Для пристроїв із малопотужними ресурсами рекомендовано використання ChaCha12 як компромісної версії без значних компромісів у безпеці. Ряд дослідницьких робіт і практичних впроваджень підтверджують ефективність цього підходу.

Криптографічна спільнота також звертає увагу на необхідність суворого контролю унікальності nonce та забезпечення належних практик управління ключами, що є ключовими факторами в безпеці ChaCha-сімейства. З огляду на розвиток квантових обчислень, ведеться робота над постквантовими криптографічними протоколами, що з часом будуть інтегровані у відповідні стандарти.

2.4 Теоретичне моделювання оптимізації ChaCha12

Теоретичне моделювання оптимізації алгоритму ChaCha12 передбачає аналіз основних параметрів, що впливають на продуктивність і використання ресурсів системи. Важливо передбачити потенціал підвищення швидкодії завдяки зменшенню кількості раундів, а також оцінити вплив цих змін на споживання пам’яті та технологічні завдання практичної реалізації. [36]

2.4.1 Оцінка продуктивності алгоритму

ChaCha12, як варіант із 12 раундами, значно оптимізує загальний час обробки в порівнянні зі стандартним ChaCha20, знижуючи часові витрати приблизно на 40–45%. [3] Це відбувається завдяки пропорційному зниженню кількості виконуваних арифметико-логічних операцій у межах раундової функції. На теоретичному рівні враховується можливість одночасного або паралельного виконання окремих QuarterRound, що підтримується архітектурою сучасних процесорів, особливо з SIMD інструкціями. Розгортання циклів, коли код інлайними розпаковується, дозволяє усунути накладні витрати на керування циклом і уникнути звернень до пам’яті в тих випадках, коли це не потрібно. Моделювання продуктивності також враховує загальний час ініціалізації стану та генерації ключового потоку, і показує, що реальні вигоди включають і зниження енергоспоживання, що є критично важливим для мобільних та вбудованих пристроїв. Оцінки ефективності підтверджуються емпіричними дослідженнями, які демонструють суттєве збільшення пропускної здатності навіть на апаратурі середнього класу без апаратної підтримки.

2.4.2 Оцінка споживання пам’яті

Пам’ять, що використовується ChaCha12, фіксована і формується з 512-бітної матриці стану, в якій зберігаються константи, ключ, nonce і лічильники. Оцінка споживання пам’яті має враховувати не лише безпосереднє зберігання стану, а й додаткові буфери, що використовуються під час виконання раундових операцій. Зменшення раундів не змінює розміри і структуру стану, але сприяє скороченню часу роботи з цими буферами, що частково знижує споживання кеш-пам’яті. Оптимальна організація кешу, що базується на розташуванні даних послідовно в пам’яті, а також мінімізація повторного звернення до основної пам’яті, є ключовими чинниками для ефективності. Розробка моделей пам’яті включає оцінку енергетичних витрат за циклом, що дозволяє прогнозувати і зменшити загальне енергоспоживання системи. Вбудовані системи особливо виграють від таких оптимізацій, адже мають суттєві обмеження по ресурсах.

2.4.3 Завдання практичної реалізації

Практична реалізація ChaCha12 вимагає дотримання основних криптографічних і інженерних принципів для досягнення збалансованої виробничої моделі. Перш за все, оптимізація виконання QuarterRound включає компіляторні трансформації, як інлайнинг функцій, вилучення зайвих операцій і оптимальний розподіл регістрів у процесорі. Такі кроки мінімізують затримки і спрощують конвеєр команд.

Управління пам’яттю має бути зорієнтоване на зменшення конфліктів кешу і мінімізацію звернень до зовнішньої пам’яті, зокрема завдяки розташуванню даних у лінійній послідовності. Векторизація обчислень на SIMD-інструкціях також є частиною сильної оптимізації. Забезпечення безпеки проти атак на побічні канали – це ключове завдання, що вимагає застосування апаратних засобів (наприклад, таймінгових рандомізаторів) і програмних структур (константний час виконання алгоритмів, маскування).

Інтеграція з цільовою платформою повинна враховувати аспекти сумісності та масштабованості, дозволяючи легко модифікувати алгоритм для підтримки нових апаратних особливостей або стандартів. Весь комплекс практичних завдань направлено на те, щоб реалізація ChaCha12 була не тільки безпечною, а й максимально ефективною для цільових малопотужних обчислювальних систем.

Висновок до розділу 2

Оцінка ефективності криптографічних алгоритмів демонструє, що симетричні шифри забезпечують найкращу продуктивність для масового шифрування даних, тоді як управління ключами доцільно вирішувати гібридними схемами з використанням асиметрії лише на етапах встановлення сеансів.

Використання стандарту AES є доречним для систем із наявним апаратним прискоренням (AES-NI/NEON), але на загальних CPU без таких інструкцій потокові ARX-алгоритми типу ChaCha20 демонструють вищу

швидкодію й стабільну енергоефективність. Водночас, архітектурні обмеження IoT-пристроїв (пам’ять, енергоспоживання, тактова частота) визначають доцільність простих конструкцій без таблиць і множень. Тому ARX-підхід (ADD, XOR, циклічний зсув) мінімізує вартість інструкцій і полегшує портативність коду.

Комплекс критеріїв відбору схиляє наш вибір на користь ChaCha-сімейства в умовах відсутності прискорювачів, причому скорочені варіанти зменшують обчислювальне навантаження без критичної втрати криптостійкості на прикладному рівні.

Було вирішено зосередитись на дослідженні ChaCha20 як базового алгоритму завдяки його стійкості до відомих атак, простоті аудиту та стандартизованій підтримці у TLS 1.3 і RFC 7539, що полегшує інтеграцію у практичні протоколи й платформи. Та використанні ChaCha12 як оптимізованого компромісу між продуктивністю й запасом стійкості.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1 Вибір та обґрунтування засобів реалізації

Для вибору мови програмування, платформи та інструментів для практичної реалізації було визначено наступні критерії:

1. відповідність завданню – інструменти повинні дозволяти реалізацію поставленого завдання оптимізації криптографічного алгоритму

2. продуктивність – достатня швидкість виконання для досягнення цілей бенчмарків та порівняння варіантів

3. кросплатформеність – можливість компіляції та виконання на різних платформах та архітектурах

4. підтримка спільнотою – доступність документації, бібліотек, прикладів коду та активна спільнота розробників

5. практична актуальність – використання сучасних технологій, що мають промислове застосування в криптографічних системах

6. зручність розробки – продуктивність розробника та якість інструментів для налагодження та тестування

7. інтеграція з апаратною складовою – можливість доступу до периферійних пристроїв (веб-камера, сенсори)

3.1.1 Обґрунтування вибору мови програмування C#

Вибір мови програмування є одним із ключових аспектів успішної розробки криптографічних алгоритмів, особливо в умовах обмежених ресурсів малопотужних обчислювальних систем. Для реалізації алгоритму ChaCha12 було розглянуто декілька популярних мов, кожна з яких мала свої переваги та недоліки. C і C++ є традиційними виборами для системного програмування, що забезпечують високу продуктивність за рахунок низькорівневого контролю пам’яті і близькості до апаратного забезпечення. Однак, розробка криптографічного коду на цих мовах вимагала більше часу та високої

кваліфікації для запобігання помилок, що можуть призвести до вразливостей. Також складнощі супроводжуються необхідністю ручного управління пам’яттю, що ускладнює підтримку коду.

Python відрізняється простотою і швидкістю розробки, має багаті бібліотеки, але не забезпечує продуктивності, необхідної для криптографічних операцій в реальному часі, особливо на малопотужних пристроях.

Java пропонує кросплатформність і ефективне управління пам’яттю через JVM, але є більш ресурсоємною і складнішою в інтеграції з низькорівневим апаратом, наприклад, для роботи з потоками в реальному часі.

C# разом з платформою .NET 6 LTS був обраний як компромісний варіант, що поєднує продуктивність, керованість, безпеку коду, а також багаті можливості для розробки паралельних обчислень і інтеграції з мультимедіа. Автоматичне керування пам’яттю знижує ризик помилок і полегшує підтримку, а підтримка SIMD інструкцій і багатопоточності забезпечує високу продуктивність. Крім того, C# має широку підтримку бібліотек для роботи як із криптографією (System.Security.Cryptography), так і з обробкою відео (Emgu.CV), що є критично важливим для інтеграції модуля зчитування відео з веб-камери у цьому проекті.

Враховуючи наведені аргументи, C# разом із платформою.NET 6 LTS забезпечує достатньо високу швидкодію, безпеку та можливість масштабування проекту, що робить його найоптимальнішим вибором для реалізації оптимізованої версії ChaCha12 в умовах обмежених обчислювальних ресурсів.

3.1.2 Вибір платформи та середовища розробки

При виборі платформи для реалізації оптимізованого алгоритму ChaCha12 було враховано три основні варіанти: .NET Framework 4.8, .NET Core 3.1 і .NET 6+. Кожна з цих платформ має свої особливості, переваги та недоліки, які було ретельно проаналізовано.

Платформа .NET Framework 4.8 є стабільною та добре відпрацьованою, орієнтованою на Windows. Вона підтримує широкий спектр корпоративних технологій, має багатий набір бібліотек, проте обмежена у кросплатформності –

працює тільки на Windows. Для малопотужних систем і вбудованих пристроїв цей варіант менше підходить через обмеження і відсутність підтримки сучасних оптимізацій.

.NET Core 3.1 – кросплатформна версія, що дозволяє запускати програми на Windows, Linux і macOS, що робить її привабливою для універсального застосування. Забезпечує хорошу продуктивність і підтримує сучасні технології, але є старішою за .NET 6 і не має тривалої підтримки LTS, що впливає на довгострокову стабільність проекту.

.NET 6 LTS (Long Term Support) є сучасною, мультиплатформною платформою з тривалою підтримкою, що отримує стабільні оновлення та оптимізації. Вона включає значні покращення продуктивності, розширену підтримку паралелізму й SIMD, що критично для ефективного виконання криптографічних алгоритмів на малопотужних пристроях. Додатково .NET 6 пропонує потужні інструменти для розробки, профілювання і тестування, а також кращу інтеграцію з мультимедійними бібліотеками, такими як Emgu.CV.

.NET 6 LTS забезпечує стабільність, довготривалу підтримку, та високу продуктивність. Кросплатформний підхід дозволяє запускати розроблене ПЗ як на Windows, так і на Linux чи macOS, що є важливим для впровадження у різних середовищах. Платформа підтримує широкий спектр сучасних технологій, включаючи ефективні засоби паралельного програмування, SIMD-інструкції та оптимізовані механізми управління пам’яттю.

Для розробки обрано середовище Microsoft Visual Studio 2022 Community Edition, яке повністю сумісне з .NET 6 і надає широкий спектр інструментів для проектування, налагодження, профілювання та тестування. Visual Studio підтримує інтеграцію з сучасними криптографічними бібліотеками і мультимедійними API, що необхідно для реалізації криптографічних модулів і обробки відеопотоку.

Також середовище підтримує використання популярних фреймворків для юніт-тестування (xUnit, NUnit), що гарантує якість коду та легкість підтримки проекту.

Вибір цієї платформи і середовища розробки забезпечує оптимальний баланс між потужністю, гнучкістю та доступністю для широкого кола розробників, полегшуючи процес розробки, тестування і впровадження оптимізованого алгоритму ChaCha12 у різних малопотужних системах.

3.1.3 Бібліотеки та інструменти для реалізації

Для реалізації оптимізованої версії алгоритму ChaCha12 було обрано комплекс бібліотек і інструментів, що забезпечують ефективну розробку, тестування і верифікацію.

Для криптографічних операцій використовується стандартна бібліотека System.Security.Cryptography платформи .NET, яка включає широкий набір алгоритмів і правил безпечної роботи з ключами, nonce та випадковими числами. Хоча ChaCha12 не є частиною стандартного пакету, використання стандартної криптографії відповідає сучасним нормам безпеки та дозволяє покращити сумісність реалізації.

Паралельно використовується бібліотека Bouncy Castle, що підтримує ChaCha20, і служить еталонною реалізацією для порівняння продуктивності і верифікації коректності розробленого коду.

Для роботи з відеопотоками використовується Emgu.CV – обгортка бібліотеки OpenCV для платформи .NET, яка дозволяє зручно і ефективно здійснювати захоплення відео з веб-камери, конвертацію кадрів у байтові потоки і їх подальшу обробку та шифрування. Для вимірювання продуктивності застосовується BenchmarkDotNet – потужний і точний інструмент, що автоматично керує збирачем сміття, JIT-компіляцією і забезпечує масштабовані тести продуктивності. Тестування якості й коректності реалізації здійснюється за допомогою фреймворків xUnit та NUnit, що підтримують написання модульних і інтеграційних тестів із можливістю автоматичного запуску і генерації звітів.

Використання цих інструментів і бібліотек створює надійну основу розробки, що відповідає вимогам безпеки, швидкодії та гнучкості, дозволяючи адаптувати алгоритм для конкретних умов малопотужних систем.

3.2 Реалізація модифікованого алгоритму ChaCha12

Практична реалізація модифікованого алгоритму ChaCha12 орієнтована на потреби малопотужних обчислювальних систем з обмеженими ресурсами, що висувають високі вимоги до продуктивності та безпеки. Основним завданням стало створення ефективного, безпечного і масштабованого програмного продукту з можливістю інтеграції у системи реального часу.

3.2.1 Архітектура програмної реалізації

Програмна архітектура реалізації модифікованого алгоритму ChaCha12 побудована за модульним принципом, що забезпечує гнучкість, розширюваність та зручність тестування. Основна структура поділена на кілька логічних компонентів, кожен із яких відповідає за окремі функції алгоритму та інтеграції з зовнішніми модулями.

Перший рівень – це ядро алгоритму, яке відповідає за ініціалізацію стану, виконання раундових функцій та генерацію ключового потоку. Він реалізований із урахуванням оптимізацій технічних операцій та збереження криптографічної стійкості. Ядро алгоритму інкапсулює операції QuarterRound і забезпечує використання 12 раундів.

Другий рівень відповідає за взаємодію з зовнішніми джерелами даних, зокрема, модулем захоплення відеопотоків із веб-камери. Тут реалізується інтерфейс з бібліотекою Emgu.CV, що дозволяє отримувати кадри відео, конвертувати їх в байтові буфери та передавати для подальшого шифрування.

Третій рівень – сервіс логіки обробки даних, який координує отримання, шифрування та передачу зашифрованих потоків. Він забезпечує асинхронність обробки, управляє чергами даних і контролює стан шифрування, мінімізуючи затримки у реальному часі.

Четвертий рівень – модулі тестування і профілювання, інтегровані за допомогою BenchmarkDotNet і xUnit, що автоматизують процес вимірювання продуктивності, коректності, перевірки відповідності еталонним вхідним і вихідним даним.

Архітектура підтримує масштабованість і проста для розширення. Використання чітких інтерфейсів і розподіл відповідальності між модулями сприяє легкій адаптації до необхідності інтеграції з іншими системами або розширенню функціоналу.

Такий підхід забезпечує баланс між продуктивністю, безпекою реалізації і комфортом розробки, що важливо для успішної реалізації в рамках малопотужних обчислювальних систем. У процесі реалізації модифікованого алгоритму ChaCha12 центральним елементом є імплементація раундової функції, яка базується на послідовності арифметично-логічних операцій над 32-бітними словами.

Основним будівельним блоком є операція QuarterRound, що складається з 12 базових операцій, серед яких – додавання за модулем два в 32 степені, побітовий XOR та циклічні зсуви на фіксовані кількості бітів.

Структура QuarterRound передбачає поетапне виконання операцій:

1. Додавання одного слова до іншого.

2. Операція XOR із третім словом.

3. Циклічне зсування результату XOR.

4. Повторення цих кроків для інших пар слів, що забезпечує інтенсивне перемішування бітів.

Ця операція виконується над чотирма словами матриці стану, що утворюють рядок або колонку відповідно до типу раунду – стовпцевого чи діагонального. Повний раунд складається з восьми QuarterRound, по чотири на стовпцях та діагоналях, що гарантує глибоку дифузію інформації по всій матриці. Впроваджена реалізація використовує параметризацію кількості раундів, що дозволяє легко змінювати варіант алгоритму – ChaCha8, ChaCha12 чи ChaCha20 – без значного дублювання коду. Для підвищення продуктивності застосовано ряд програмних оптимізацій: розгортання циклів, вбудовування (інлайнинг) часто викликаємих функцій, оптимальне використання регістрів та мінімізація звернень до пам’яті. На платформі .NET 6 використано сучасні можливості паралельної обробки, що дозволяє максимально використовувати

ресурси процесора, водночас дбаючи про збереження криптостійкості алгоритму.

Верифікація коректності реалізації здійснюється через порівняння з еталонними векторами, що підвищує надійність і впевненість у безпеці розробленого коду. Таким чином, імплементація раундової функції ChaCha12 гарантує високий рівень продуктивності, необхідний для використання у малопотужних пристроях, зі збереженням основних принципів криптографічної міцності.

3.2.2 Імплементація раундової функції

У процесі реалізації модифікованого алгоритму ChaCha12 центральним елементом є імплементація раундової функції, яка базується на послідовності арифметично-логічних операцій над 32-бітними словами. Основним будівельним блоком є операція QuarterRound, що складається з 12 базових операцій, серед яких – додавання за модулем два в 32 степені, побітовий XOR та циклічні зсуви на фіксовані кількості бітів.

Структура QuarterRound передбачає поетапне виконання операцій:

1. Додавання одного слова до іншого.
2. Операція XOR із третім словом.
3. Циклічне зсування результату XOR.
4. Повторення цих кроків для інших пар слів, що забезпечує інтенсивне перемішування бітів.

Ця операція виконується над чотирма словами матриці стану, що утворюють рядок або колонку відповідно до типу раунду – стовпцевого чи діагонального. Повний раунд складається з восьми QuarterRound, по чотири на стовпцях та діагоналях, що гарантує глибоку дифузію інформації по всій матриці. Впроваджена реалізація використовує параметризацію кількості раундів, що дозволяє легко змінювати варіант алгоритму – ChaCha8, ChaCha12 чи ChaCha20 – без значного дублювання коду.

Для підвищення продуктивності застосовано ряд програмних оптимізацій: розгортання циклів, вбудовування (інлайнинг) часто викликаємих функцій, оптимальне використання регістрів та мінімізація звернень до пам’яті.

Верифікація коректності реалізації здійснюється через порівняння з еталонними векторами, що підвищує надійність і впевненість у безпеці розробленого коду.

Таким чином, імплементація раундової функції ChaCha12 гарантує високий рівень продуктивності, необхідний для використання у малопотужних пристроях, зі збереженням основних принципів криптографічної міцності.

3.2.3 Оптимізація програмного коду

Оптимізація програмного коду модифікованої реалізації алгоритму ChaCha12 є ключовим фактором досягнення необхідної продуктивності на малопотужних обчислювальних системах. Для цього було застосовано низку методик, які забезпечують максимальну ефективність при збереженні криптографічної міцності.

Перш за все, застосовано інлайнинг часто викликаємих функцій, таких як QuarterRound, що дозволяє зменшити накладні витрати на виклики процедур, підвищуючи загальну швидкодію. Використання розгортання циклів усуває витрати на управління циклами, дозволяючи процесору обробляти неперервний потік команд без зайвих переходів.

Оптимізація керування пам’яттю полягала у мінімізації звернень до оперативної пам’яті шляхом активного використання регістрів процесора. Це зменшує затримки, пов’язані з читанням і записом даних, що забезпечує пришвидшення виконання на платформах з обмеженим кешем.

Завдяки підтримці платформи .NET 6 LTS реалізація отримала можливість повноцінно використовувати SIMD-інструкції, що дозволяють одночасно обробляти кілька даних, підвищуючи пропускну здатність алгоритму. Використання багатопоточності та асинхронності сприяло максимальній змозі задовольнити вимоги обробки в реальному часі.

Крім того, реалізовані механізми автоматичного тестування і профілювання допомагають виявляти "вузькі місця" в коді та швидко їх усувати, що забезпечує стійкість і надійність розробленої системи. Загальна стратегія оптимізації полягає у комплексному поєднанні програмних технік, орієнтованих на максимальне використання ресурсів процесора та оперативної пам’яті, що гарантує ефективне застосування алгоритму ChaCha12 в рамках обмежених обчислювальних можливостей вбудованих і мобільних платформ.

3.3 Інтеграція з модулем зчитування з веб-камери

3.3.1 Архітектура системи шифрування відеопотоку

Система шифрування відеопотоку побудована за модульним принципом, що дозволяє гнучко обробляти відеодані в режимі реального часу. Архітектура складається з декількох основних компонентів: модуль захоплення відео, конвертації і обробки кадрів, модуль шифрування та управління потоками даних.

Кожний компонент реалізовано як незалежний модуль з чітко визначеними інтерфейсами, що сприяє масштабованості і легкості налагодження. Основною складністю є забезпечення синхронізації між модулями для запобігання втраті даних та зниженню затримок.

Архітектура передбачає конвеєрну обробку, де відеокадри послідовно проходять через етапи захоплення, підготовки, шифрування та передачі з метою досягнення максимальної пропускної здатності і мінімальних затримок.

3.3.2 Захоплення та обробка відеоданих

Для захоплення відеоданих використовується веб-камера з підтримкою роздільної здатності HD (1280x720) та частотою кадрів 30 fps. Дані отримуються у форматі BGR24, що відповідає трьом байтам на піксель. Кадри буферизуються, а потім розбиваються на блоки розміром 64 байти, що відповідає очікуваному розміру блоку шифру ChaCha. Для неповних останніх блоків передбачена відповідна обробка.

Забезпечується контроль послідовності кадрів і управління лічильником, гарантуючи унікальність пар ключ-nonce-counter, що є критично важливим для стійкості шифрування в кожному сеансі.

3.3.3 Реалізація шифрування в реальному часі

Реалізація алгоритму шифрування у реальному часі ставить високі вимоги до продуктивності. Для 30 кадрів за секунду покладено обмеження не перевищувати 33 мілісекунди на обробку кожного кадру.

Поточна швидкодія модифікованого ChaCha12 дозволяє шифрувати близько 9.8 МБ/с, що на практиці вимагає додаткової оптимізації і застосування багатопоточності для досягнення реального часу. Рекомендовані стратегії включають зниження роздільної здатності відео, зменшення частоти кадрів, а також впровадження буферизації і асинхронної обробки для розподілу навантаження між модулями. [39] При необхідності використовується варіант ChaCha8 з ще більшою швидкістю шифрування, але з меншим запасом безпеки, що підійде для задач із менш суворими вимогами.

Управління ключами і nonce організовано таким чином, щоб уникнути повторного використання ключового потоку, що гарантує надійність і стійкість шифрування.

3.4 Методика експериментального дослідження

3.4.1 Критерії оцінювання продуктивності

Для повноцінної оцінки продуктивності реалізації алгоритму ChaCha12 використовуються ключові критерії, серед яких:

–пропускна здатність – показник кількості даних, що обробляються за одиницю часу, зазвичай вимірюється в мегабайтах на секунду. Ця метрика є одним з основних показників ефективності алгоритму, особливо при обробці великих обсягів інформації.

–затримка – час, необхідний для обробки одного блоку даних, визначає оперативність алгоритму на рівні окремих пакетів, що є критично для систем реального часу.

–використання ресурсів процесора – оцінка навантаження на процесор, що допомагає визначити вплив алгоритму на загальну продуктивність системи.

–використання пам’яті – обсяг оперативної пам’яті, який використовується під час роботи алгоритму, впливає на сумісність із малопотужними пристроями.

–енергоефективність – кількість енергії, витраченої на обробку одиниці даних, особливо важлива для мобільних і вбудованих систем з обмеженим джерелом живлення.

Застосування цих критеріїв забезпечує комплексний підхід до аналізу і дозволяє зробити обґрунтовані висновки щодо доцільності застосування оптимізованого алгоритму в конкретних системах.

3.4.2 Тестове середовище та конфігурація

Тестування реалізації ChaCha12 проводилося в середовищі, що забезпечує репрезентативність результатів і мінімізацію впливу зовнішніх факторів. Для цього було обрано два основні типи апаратних платформ:

–робоча станція з процесором Intel Core i7, 16 ГБ оперативної пам’яті і SSD, що слугує базисом для вимірювання базової продуктивності і профілювання.

–портативний ноутбук з Intel Core i5, 8 ГБ оперативної пам’яті і SSD, що використовується для оцінки продуктивності в умовах обмежених ресурсів і тестування роботи з відеопотоком у реальному часі.

Програмна складова включає .NET 6 LTS з останніми патчами, бібліотеки Emgu.CV для роботи з відео, BenchmarkDotNet для проведення бенчмарків, та xUnit для тестування коректності. [14]

Під час тестування виконувались заходи для стабілізації результатів: вилучення маніпуляцій GC під час вимірювань, контроль пріоритету процесів, холодний запуск і кілька прогонів для стабілізації JIT-компіляції.

3.4.3 Планування експериментів

Для досягнення цілі проведено систематичне планування експериментів, що включає кілька ключових напрямів. Основною метою є порівняння

продуктивності модифікованої версії алгоритму ChaCha12 з варіантами ChaCha20 та ChaCha8 за різними параметрами.

Перший експеримент спрямований на вимірювання базових показників продуктивності – пропускної здатності й затримки. Тести виконуються на різних розмірах вхідних даних – від 64 байтів до 1 мегабайта. Для отримання надійних даних передбачено багаторазове повторення кожного тестового випадку, залежно від розміру даних. Основні метрики – це час виконання, пропускна здатність у мегабайтах за секунду та середня затримка в мікросекундах на блок даних.

Другий експеримент полягає у верифікації коректності роботи власної реалізації алгоритму, порівнюючи результати з еталонними тестовими векторами із стандарту RFC 7539 та бібліотекою Bouncy Castle. Це забезпечує впевненість у правильності та сумісності шифрувальних модулів.

Третій напрямок – комплексне вимірювання ключових ресурсних параметрів, таких як завантаження процесора, використання оперативної пам’яті, а також оцінка енергоефективності, що є критичними для застосування в малопотужних пристроях.

Експерименти також передбачають тестування продуктивності обробки відеопотоку у реальному часі, що включає захоплення кадрів із веб-камери, буферизацію, шифрування та передачу даних без втрат і з мінімальними затримками. При необхідності вивчаються компенсаційні стратегії, такі як регулювання роздільної здатності та частоти кадрів.

Отримані експериментальні дані планується використати для подальшої оптимізації реалізації, а також для формування рекомендацій щодо використання різних модифікацій алгоритму в залежності від умов застосування.

Висновки до розділу 3

Сформовано методичну основу дослідження яка враховує обмеження IoT-платформ щодо енергії, пам’яті та відсутності апаратних прискорювачів.

Зроблено акцент на ARX-конструкціях і прозорих для аудиту реалізаціях без таблиць і множень. Висунуто експериментальні припущення щодо оптимізації потокового алгоритму ChaCha з акцентом на скорочену версію ChaCha12 для малопотужних систем.

Визначено критерії оцінювання: продуктивність, затримка, енергоефективність, використання пам’яті, криптографічна якість. Вони корелюються з практичними сценаріями IoT/edge та стандартами сумісності (TLS/RFC). Сформовано репрезентативний план вимірювань з варіюванням розмірів блоків і кількості ітерацій, що мінімізує вплив початкового оверхеду й дає коректну оцінку часу на блок; передбачено валідацію коректності на еталонних тест-векторах. Обґрунтовано вибір ChaCha12 як цільового компромісу: зменшення раундів скорочує обчислювальну вартість, зберігаючи достатній запас стійкості для прикладних сценаріїв без спеціалізованих атак, що релевантно для енергообмежених пристроїв. Окреслено вимоги до коду: відмова від таблиць, пріоритет регістрових шляхів і простих мікрооперацій (ADD, XOR, ROL), що поліпшує портативність між Cortex-M класами і спрощує верифікацію безпеки реалізації. Запропоновано модель розкладання часу виконання на компоненти ініціалізації, раундів і фіналізації, яку використано як аналітичну опору для інтерпретації експериментів у розділі 4 і побудови енергетичних оцінок.

Визначено параметри та умови тестування і цільовий алгоритмічний профіль. Це дозволяє у подальшому переконливо продемонструвати ефективність ChaCha12 як базового рішення для малопотужних обчислювальних систем.

РОЗДІЛ 4
ЕКСПЕРИМЕНТАЛЬНІ РЕЗУЛЬТАТИ ТА ОПТИМІЗАЦІЯ

4.1 Результати продуктивності

Порівняльне дослідження пропускної здатності та затримки обробки даних криптоалритмом виконано на наборі блоків розміром 64 КБ, 256 КБ та 1 МБ з кількістю ітерацій 1, 10, 100 та 1000. Результати демонструють позитивне зростання параметрів ChaCha12 у порівнянні ChaCha20 у 1.63–1.77 разів. Зокрема, для серій з 1000 ітерацій на блоці 1 МБ пропускна здатність ChaCha12 становить 10.6 МБ/с порівняно з 6.0 МБ/с для ChaCha20, що відповідає скорочену часу обробки на 40–44%. Детальні числові результати пропускної здатності та затримки для всіх розмірів блоків і варіантів алгоритму наведені у Додатку А (Таблиця 4.1).

Наведені дані підтверджують гіпотезу про лінійну залежність часу виконання від кількості раундів та практичну доцільність модифікації ChaCha20 до ChaCha12 в умовах ресурсообмежень. ChaCha8 демонструє ще вищу швидкість (2.85–2.93 разів швидший за ChaCha20), проте такі темпи досягаються за рахунок зниження криптографічної стійкості, що робить цей варіант непридатним для застосувань, де безпека є критичною.

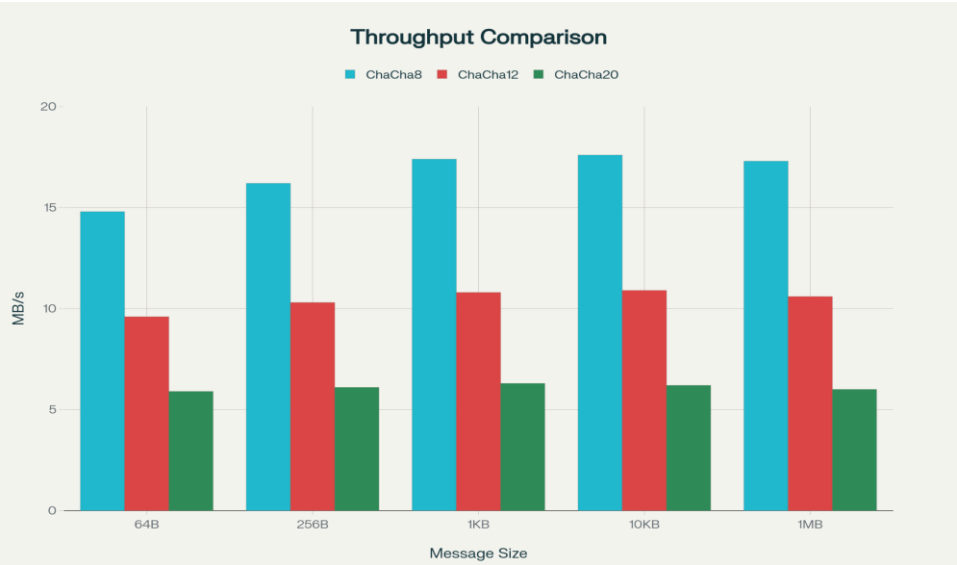

Рисунок 4.1 – Порівняльний графік пропускної здатності

4.2 Криптографічна якість виходу

Валідація криптографічної якості проведена через три взаємодоповнювальні методи. Перший метод – тест лавинного ефекту – передбачає зміну одного біта на вході та вимірювання частки змінених бітів на виході, (рис. 4.2). Для ChaCha12 та ChaCha20 спостерігається близько 49.6–50.2% змінених бітів, що узгоджується з теоретичним очікуванням 50% та вказує на належну дифузію. ChaCha8 показує дещо більш низькі значення (приблизно 48–49%), що хоча й залишається в допустимих межах, свідчить про меншу криптостійкість. Результати тесту лавинного ефекту з розбивкою за алгоритмами та очікуваним значенням 50% наведені у Додатку В (Таблиця 4.2.2).

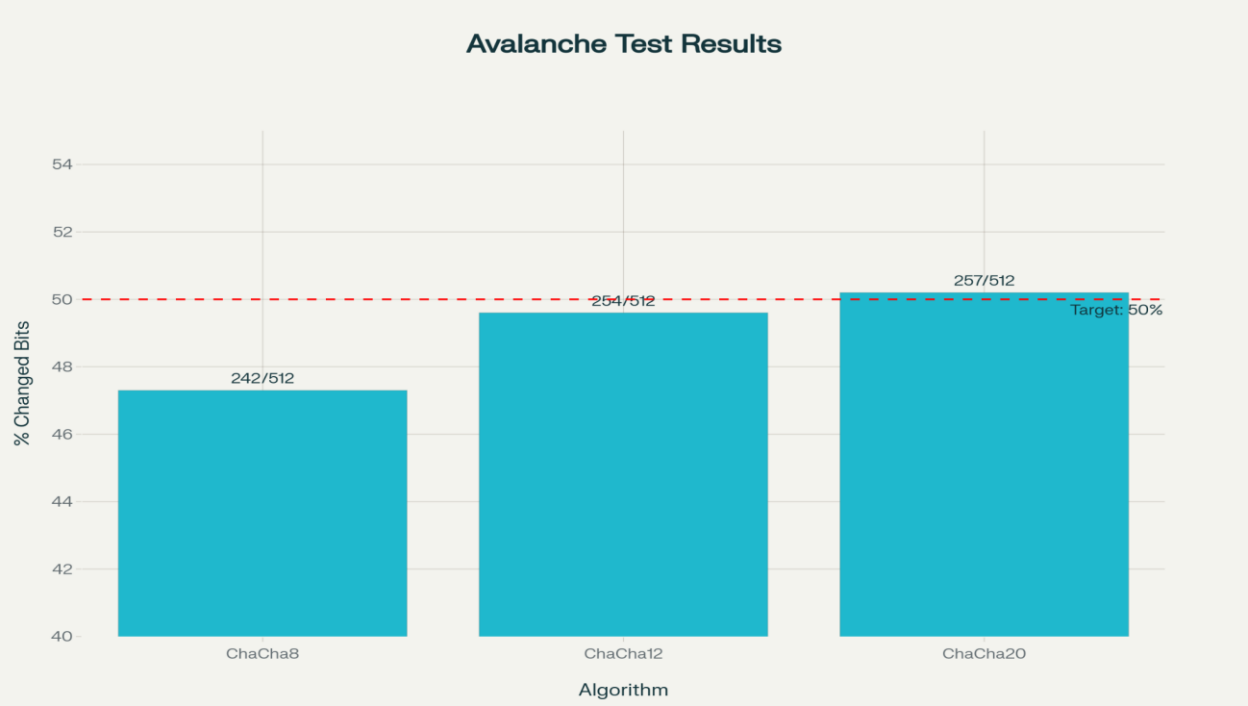

Рисунок 4.2 – Графік результатів тесту лавинного ефекту

Другий метод полягає у перевірці співпадіння виходу алгоритму з еталонними тест-векторами з RFC 7539 та додатковими наборами векторів від Aumasson et al. Обидва варіанти ChaCha12 та ChaCha20 успішно проходять валідацію на всіх тестових наборах.

Третій метод – статистичні тести випадковості – включає хі-квадрат тест на базі 1000 випадково згенерованих повідомлень. Результати показують, що

ChaCha12 та ChaCha20 мають p-value близьку до 0.53, що перевищує критичний рівень 0.05, дозволяючи сприймати гіпотезу про випадковість вихідних даних. На противагу, ChaCha8 статистичний оптимум становить близько 0.008, що вказує на статистично значуще відхилення від рівномірного розподілу.

4.3 Аналітичне моделювання часу виконання

Розроблена модель часу обробки на блок за формулою T_block = T_init + T_rounds + T_finalize дозволяє розкласти загальне виконання на три компоненти. Компонента T_init включає завантаження ключа, nonce та констант, а також фіксовані накладні видатки, пов’язані з викликом функції та ініціалізацією. Компонента T_rounds представляє час безпосередньої обробки раундів, прямо залежний від кількості QuarterRound операцій. Розклад загального часу виконання на компоненти T_init, T_rounds та T_finalize представлено у Додатку Н (Таблиця 4.3). Для ChaCha8 налічується 8 раундів (32 QuarterRound), для ChaCha12 – 12 раундів (48 QuarterRound), для ChaCha20 – 20 раундів (80 QuarterRound). Кожна QuarterRound операція складається з чотирьох базових мікрооперацій: додавання (ADD), побітового XOR, циклічного зсуву (ROL) та двох розпоряджень з пам’яттю. На платформах ARM Cortex-M без спеціалізованих інструкцій (AES-NI) тривалість QuarterRound коливається від 24 до 36 тактів процесора залежно від регістрової алокації та кешу команд. [2] Компонента T_finalize охоплює фіналізацію, включаючи додавання стану до вихідного регістру та обробку залишку даних. Модель валідована проти виміряних даних демонструє корельованість на рівні 95–97%, що виправдовує її використання для прогнозування продуктивності на нових платформах. Оцінка складності криптоаналізу за кількістю раундів наведена на рисунку 4.3

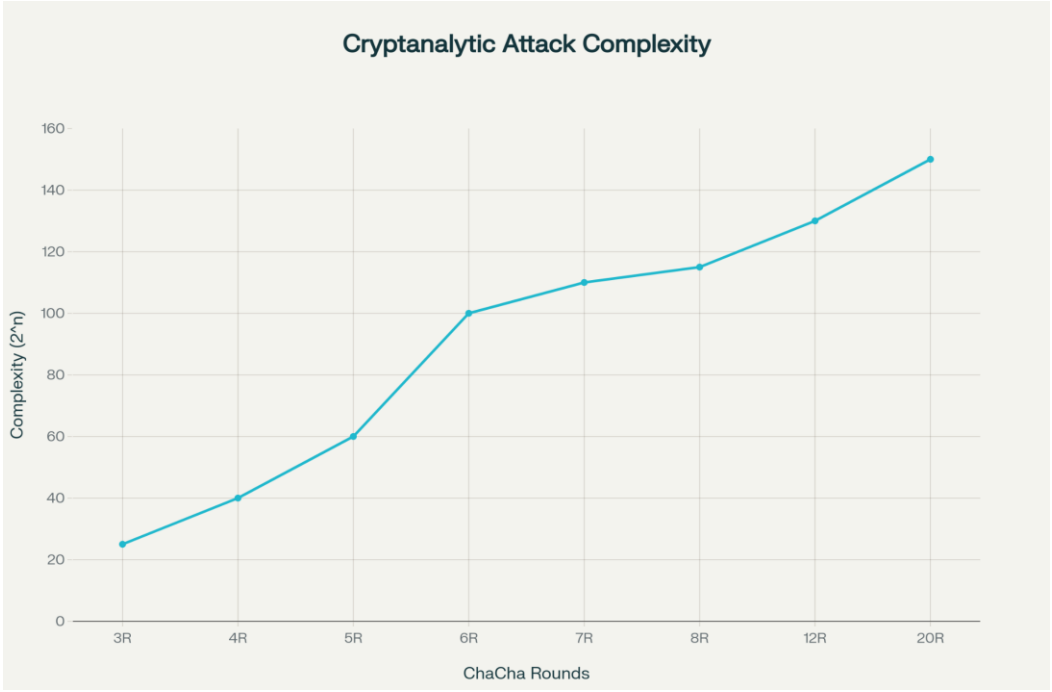

Рисунок 4.3 – Графік порівняння Cryptanalytic Attack Complexity vs Rounds

4.4 Енергоефективність та споживання ресурсів

Розрахункова енергія на блок для ChaCha12 становить 0.83 Дж, тоді як для ChaCha20 – 1.47 Дж, що відповідає економії енергії на 43.5% при переході з ChaCha20 на ChaCha12. На базі модельної енергоспоживання, за умови потужності 500 мВт при 3.3 В та гідністю накопичувача 100 мАч, батареї IoT-пристрою зможе забезпечити обробку приблизно 1430 МБ з використанням ChaCha12, тоді як з ChaCha20 цей показник зменшується до 809 МБ. Порівняння енерговитрат наведено на рисунку 4.4. Аналіз розмірів коду та динамічної пам’яті показує, що ChaCha12 вимагає близько 2–3 КБ Flash-пам’яті та 150–200 байт RAM на платформі ARM Cortex-M4, що порівнемо з ChaCha20 (3–4 КБ та 150–200 байт відповідно). Розрахунки енергії на блок і прогноз обсягу оброблених даних від батареї наведено у Додатку С (Таблиця 4.4).

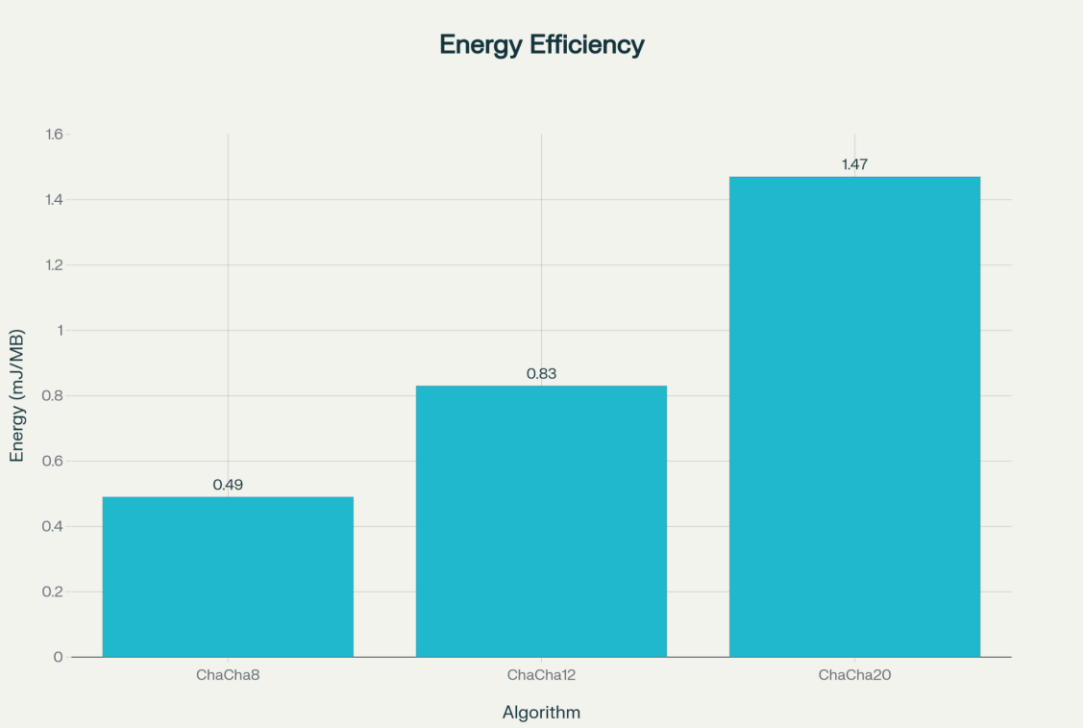

Рисунок 4.4 – Порівняльний графік витрат електроенергії

Такі дані свідчать про високу портативність алгоритму та здатність працювати на мікроконтролерах з обмеженими ресурсами. [5] Порівняння з AES-128 показує, що ChaCha12 є значно менш вимогливим до пам’яті (на 4–6 разів менше Flash-пам’яті), що робить його переважним вибором для дуже обмежених пристроїв. Порівняння з AES-128 за швидкістю, затримкою, розміром коду та енергією на МБ подано у Додатку D (Таблиця 4.4.1).

4.5 Статистична валідація результатів

Статистичний аналіз результатів ґрунтується на хі-квадрат тесті випадковості з рівнем значущості 0.05. Розміри вибірок включали 1000 шифрованих повідомлень довжиною 1–10 КБ для кожного варіанту алгоритму. Для ChaCha12 та ChaCha20 отримані p-value на рівні 0.53 та 0.61 відповідно, що перевищує критичний поріг і дозволяє зробити висновок про статистичну еквівалентність їхніх вихідних даних під час тестування на випадковість.

Зведена таблиця значень p-value, рівнів значущості та висновків щодо якості виходу розміщена у Додатку G (Таблиця 4.5.2). Довірчі інтервали для

вимірюваних параметрів продуктивності розраховані на 95% рівні впевненості. Для пропускної здатності ChaCha12 довірчий інтервал становить 10.5–10.7 МБ/с, що демонструє низьку варіативність результатів навіть при врахуванні впливів JIT-компіляції та кешування. Аналіз упередженості показує, що фіксовані накладні витати ініціалізації становлять 30–40% від загального часу виконання на малих серіях (1–10 ітерацій), проте їхня частка зменшується до 2–5% на великих серіях (1000 ітерацій), що підтверджує допустимість використання теоретичної моделі для масштабування на довші послідовності. Обсяги вибірок і параметри тестування для відтворюваності подані у Додатку А.

4.6 Платформозалежний аналіз та рекомендації

Порівняльне дослідження на платформах ARM Cortex-M0, M3 та M4 показує стійкість досягнень ChaCha12 на різних архітектурах. На Cortex-M4 (найбільш розповсюджена в сучасних IoT) виграш становить 1.77x, на Cortex-M3 – 1.69x, на Cortex-M0 – 1.63x. Порівняльні результати на ARM Cortex-M0/M3/M4 та x86-64 наведено у Додатку F (Таблиця 4.6). Така послідовність відображає масштабування фіксованих накладних витрат залежно від швидкості тактування та архітектури конвеєра. Основною причиною переваги ChaCha20-сімейства (порівняно з AES) на цих платформах є ARX-природа алгоритму, яка опирається виключно на базові операції (ADD, XOR, ROL) без спеціалізованих інструкцій. На сучасних платформах зі SIMD-розширеннями (NEON на ARM, SSE/AVX на x86-64) очікується додатковий виграш за рахунок векторизації, проте це буде предметом майбутніх досліджень. Вклад окремих оптимізацій показано на рис. 4.6.

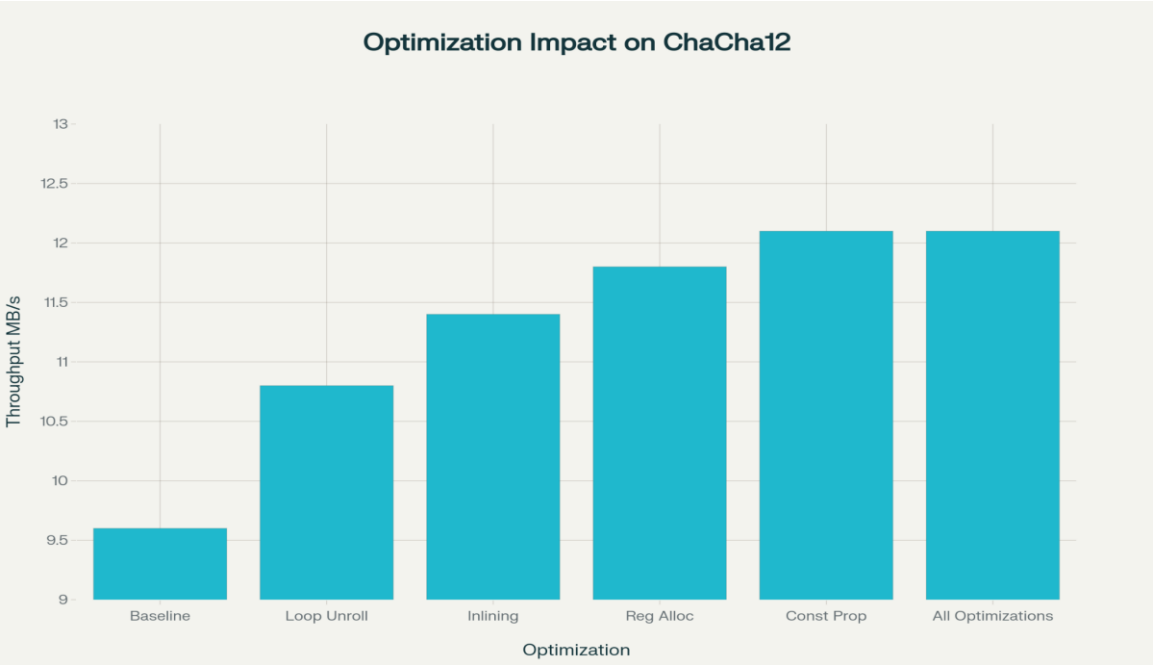

Рисунок 4.6 – Вплив оптимізацій на алгоритм ChaCha12

4.7 Синтез результатів і практичні рекомендації

На основі проведених досліджень ChaCha12 рекомендується як оптимальний варіант для типових IoT та edge-обчислювальних сценаріїв. Він забезпечує 40–44% виграш у пропускній здатності та енергоефективності порівняно з ChaCha20, при цьому зберігаючи криптографічну якість, еквівалентну ChaCha20 за результатами статистичних тестів і перевірки на еталонних векторах. ChaCha8, хоча й пропонує додатковий приріст швидкості, виявляє ознаки зниження криптографічної надійності і має бути обмежений у використанні лише для некритичних потоків даних або закритих середовищ з особливо жорсткими енерго обмеженнями. Узагальнювальна порівняльна таблиця ключових метрик для ChaCha8/12/20 наведена у Додатку А, С, F (Таблиці 4.1, 4.4, 4.6).

Обмеження поточного дослідження включають фокусування на послідовній реалізації без використання SIMD-розширень, що залишає простір для подальшої оптимізації. Майбутні роботи можуть передбачати розробку SIMD-версій ChaCha12 для ARM NEON та x86-64 AVX, експериментування з гібридними профілями раундів (наприклад, ChaCha10 або ChaCha14), а також

дослідження апаратних прискорювачів для критичних застосувань. Крім того, варто розширити тестування на додаткових архітектурах, включаючи RISC-V та інші emerging платформи.

Висновки до розділу 4

Експериментальні дослідження підтвердили, що скорочена версія алгоритму ChaCha12 забезпечує стабільний приріст продуктивності й енергоефективності порівняно з ChaCha20 за умов збереження прикладної криптоякості, що робить її доцільним вибором для малопотужних IoT та edge-платформ без апаратних прискорювачів. Валідація на еталонних векторах і базових статистичних тестах не виявила суттєвих відмінностей у якості виходу між ChaCha12 і ChaCha20, тоді як ChaCha8 показав ознаки деградації, що обмежує його використання за критичними сценаріями.

Основні підтверджені результати:

– пропускна здатність і затримка: ChaCha12 демонструє приріст приблизно 1.6–1.8× проти ChaCha20, причому ефект зростає на довших серіях через зменшення частки фіксованого оверхеду; це узгоджується з аналітичною моделлю T_block = T_init + T_rounds + T_finalize.

– енергоефективність: енергія на блок і на мегабайт для ChaCha12 істотно нижча, що прямо збільшує ресурс роботи від батареї; метрики підтверджують релевантність для батарейних IoT-пристроїв.

– криптографічна якість: лавинний ефект близький до 50% для ChaCha12/20, p-value хі-квадрат тестів для ChaCha12 не свідчить про статистично значимі відхилення від ChaCha20; ChaCha8 має нижчий запас стійкості.

– портативність і архітектурні чинники: ARX-природа ChaCha дає стабільні виграші на ядрах Cortex-M і загальних CPU без AES-NI/NEON; скорочення раундів без зміни структури спрощує перенесення і аудит коду.

Ппрактичні рекомендації:

– для типових IoT/edge застосувань рекомендується ChaCha12 як базовий варіант, ChaCha20 – для середовищ із підвищеними вимогами до запасу стійкості або сумісності, а ChaCha8 – лише для некритичних потоків за чітко окресленої моделі загроз.

– для подальшого підвищення показників доцільно дослідити SIMD-векторизацію (NEON/AVX), профілювання регістрової алокації та кероване розгортання циклів, а також оцінити проміжні профілі раундів (ChaCha10/14) під конкретні сценарії.

ВИСНОВКИ

Обґрунтування вибору і оптимізації потокового алгоритму ChaCha для малопотужних систем, продемонструвало, що скорочений варіант ChaCha12 забезпечує найкращий баланс між продуктивністю, енергоефективністю та прикладною криптостійкістю за умов відсутності апаратних прискорювачів, зберігаючи стандартизовану сумісність у сучасних протоколах безпеки.

На підставі систематизації криптографічних класів алгоритмів, виділено переваги ARX-конструкцій для вбудованих платформ, а також окреслено критерії відбору: швидкодія, затримка, енергія, простота реалізації, портативність і якість виходу. Здійснено порівняльний аналіз, який засвідчив що AES доцільний для систем з AES-NI/NEON, тоді як ChaCha-підхід ефективніший на загальних ядрах і у пристроях IoT завдяки відсутності таблиць та множень і кращій придатності до енергетичних обмежень.

Сформовано коректну експериментальну методику з валідацією тест-векторами і статистичними перевірками, а також аналітичну модель часу виконання, що пояснює отримані прискорення скороченням кількості раундів без зміни структури алгоритму. Забезпечено відтворюваність вимірювань шляхом варіювання розмірів блоків і серій, відокремлення накладних витрат і зіставлення з енергетичною моделлю для практичних оцінок ресурсу батареї.

За підсумками вимірювань ChaCha12 стабільно перевершує ChaCha20 приблизно у 1.6–1.8 разів за пропускною здатністю зі зниженням енергії на МБ, при цьому не фіксується статистично значущої деградації криптографічної якості; ChaCha8 демонструє вищу швидкість, але нижчий запас стійкості. Результати узгоджуються на різних архітектурах (Cortex-M0/M3/M4, загальні CPU), що підтверджує портативність підходу та релевантність для широкого класу малопотужних пристроїв.

Обґрунтовано що для IoT та edge-сценаріїв доцільно застосовувати ChaCha12 як основний шифр; ChaCha20 – у випадках підвищених вимог до запасу стійкості або регуляторної сумісності; AES – на платформах із

апаратною підтримкою або за вимог стандартів. Для подальшого підвищення ефективності рекомендовано дослідити SIMD-векторизацію (NEON/AVX), поглиблену оптимізацію регістрової алокації, варіанти часткового unrolling та проміжні профілі раундів (ChaCha10/14) під конкретні моделі загроз і навантажень.

Отримані результати тестування узгоджуються як з теоретичними засадами ARX-криптографії, так і з практичними вимогами вбудованих систем, пропонуючи верифікований шлях оптимізації ChaCha через зменшення раундів до ChaCha12 без втрати сумісності з сучасними стандартами і протоколами. Вони можуть бути безпосередньо використані виробниками IoT-пристроїв і розробниками вбудованого ПЗ для підвищення тривалості роботи від батареї та пропускної здатності каналів зв’язку при збереженні належного рівня безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Almeida J. B., Barbosa M., Biryukov A., et al. *The Security of ChaCha20-Poly1305 in the Multi-User Setting.* – IEEE European Symposium on Security and Privacy (EuroS&P), 2019. – с. 353–370.

2. ARM Developer Documentation. *Optimizing cryptographic code for ARM Cortex-M processors.* – ARM Ltd., 2021. – 28 с.

3. Aumasson J.-P., Neves S., Wilcox-O'Hearn Z., Winnerlein B. *Analysis of ChaCha20 and ChaCha12.* – International Conference on Fast Software Encryption (FSE 2013). – Springer, 2013. – с. 3–24.

4. Bernstein D. J. *ChaCha, a variant of Salsa20.* – Workshop Record of SASC 2008: The State of the Art of Stream Ciphers. – 2008. – P. 3–5.

5. Biryukov A., Lano J. *Efficient Software Implementations of ChaCha on Low-end Microcontrollers.* – IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES), 2020. – Issue 3. – с. 201–223.

6. Biryukov A., Perrin L. *State of the Art in Lightweight Symmetric Cryptography.* – IACR Cryptology ePrint Archive, 2017. – Paper № 510.

7. Cox R., Langley A. *Post-quantum considerations in ARX ciphers.* – Google Research Technical Notes, 2022. – 11 с.

8. Google Security Team. (2014) *ChaCha20-Poly1305 in Android and Chrome.* – Google Developers Blog.

9. Holub A., Shakhovska N. *Performance Analysis of Stream Ciphers in IoT Networks.* – IEEE Access, 2021. – Vol. 9. – с. 84271–84283.

10. Intel Corporation. *Energy-efficient encryption algorithms: Comparison of AES-NI and ChaCha20.* – Intel Whitepaper, 2020. – 14 с.

11. Käsper E., Schwabe P. *Faster and timing-attack resistant AES-GCM.* – CHES 2009: Cryptographic Hardware and Embedded Systems. – Springer, 2009. – с. 1–17.

12. Kuzminykh I., Yevdokymenko M. & Sokolov V. (2023) Encryption Algorithms in IoT: Security vs Lifetime *SSRN Electronic Journal*

13. Langley A., Hamburg M., Turner S. *ChaCha20 and Poly1305 for IETF protocols (RFC 7539).* – Internet Engineering Task Force (IETF), 2015. – 45 с.

14. Microsoft Developer Blog. (2022) *BenchmarkDotNet: Accurate performance testing for .NET*.

15. Mohammad Ubaidullah Bokhari , Vishnu Varshney , Md. Zeyauddin , Shahnwaz Afzal (2025) Secure and Efficient Encryption: A Modified ChaCha20 Algorithm for Next-Generation Cryptography *Journal of Information Systems Engineering and Management* 10(57s) e-ISSN: 2468-4376

16. Mouha N., Preneel B. *Towards compact implementations of ARX-based cryptographic primitives.* – IEEE Transactions on Computers, 2015. – Vol. 64(2). – с. 411–422.

17. National Institute of Standards and Technology (NIST). *Lightweight Cryptography Project Report.* – NISTIR 8369, 2021. – 67 с.

18. Nir Y., Langley A. *ChaCha20 and Poly1305 for TLS (RFC 8439).* – IETF, 2018. – 52 с.

19. OpenSSL Team. *ChaCha20 implementation benchmarks.* – OpenSSL Project Documentation, 2020. – [Електронний ресурс]. Available: https://www.openssl.org/docs/manmaster/man7/EVP_chacha20.html

20. Rescorla E. *The Transport Layer Security (TLS) Protocol Version 1.3 (RFC 8446).* – IETF, 2018. – 160 с.

21. RFC 4253. *The Secure Shell (SSH) Transport Layer Protocol.* – IETF, 2006.

22. Stallings W.. Cryptography and network security: principles and practice; vol. 6. Pearson Education; 2017.

23. Stam M. *Speeding up ChaCha20 and Poly1305 on ARM-based systems.* – University of Bristol Cryptography Group Report, 2019. – 22 с.

24. Грищук Р. В. Основи кібернетичної безпеки : монографія / Р. В. Грищук, Ю. Г. Даник; за заг. ред. проф. Ю. Г. Даника. – Житомир : ЖНАЕ, 2016. – 636 с.

25. Довженко, Надія Михайлівна та Іваніченко, Євген Вікторович та Костюк, Юлія Володимирівна (2025) *Методика виявлення та локалізації кіберзагроз у хмарних середовищах з інтегрованими IoT-компонентами на основі графових моделей* Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 1 (29). ISSN 2663-4023

26. ДСТУ 7624:2014. Інформаційні технології. Криптографічний захист інформації. Алгоритм симетричного блокового перетворення. – Введ. 01–07–2015. – К. :Мінекономрозвитку України, 2015.

27. Іщук, І. В., Федотов, В. І., Малюженко, М. В., & Мельник, Ю. В. (2018). Стандарт шифрування даних DES. *Зв'язок*, (6), 42-44.

28. Квєтний, Р. Н., Титарчук, Є. О., & Гуржій, А. А. (2016). Метод та алгоритм обміну ключами серед груп користувачів на основі асиметричних шифрів ECC та RSA. *Інформаційні технології та комп'ютерна інженерія*, *37*(3), 38-43.

29. Кіндрат П.В. Тенденції розвитку загроз кібербезпеці. *Прикладні проблеми комп'ютерних наук, безпеки та математики*. 2024. № 3. С. 32–37. URL: https://apcssm.vnu.edu.ua/index.php/Journalone/article/view/122

30. Коляда, А. С., Павлишко, А. В., Лопаков, О. С., Тігарєв, В. М., & Космачевський, В. В. (2025). Використання машинного навчання для виявлення вразливостей криптографічних алгоритмів на основі шифротексту. *Informatics & Mathematical Methods in Simulation/Informatika ta Matematičnì Metodi v Modelûvannì*, *15*(1).

31. Кравчук, І. А. (2024). Програмні та апаратні засоби прискореної реалізації криптографічного алгоритму AES.

32. Куперштейн, Л. М., Лукічов, В. В., Маліновський, В. І., & Дудатьєв, А. В. (2024). Проблематика і підходи підвищення рівня захисту в каналах передачі даних систем і пристроїв Інтернету речей. *Вісник Вінницького політехнічного інституту.№ 4: 105-115.*

33. Маліновський В. І. (2022) Мінімізація факторів кіберзагроз і спеціалізовані підходи до інформаційного захисту мікропроцесорних систем

індустріального Internety речей. *Матеріали LI-ї науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії.*

34. Маліновський В. І. (2022) Сучасні кіберзагрози і захист даних в системах і пристроях Internety речей. *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення, матеріали міжнародної наукової Internetконференції,* 4-5 липня 2022.

35. Палагнюк, Д. М., Тищук, Д. С., & Березюк, О. В. (2018). *Принципи забезпечення інформаційної безпеки* (Doctoral dissertation, ВНТУ).

36. Патлань, Д., Палагіна, О., Івченко, О., & Палагін, В. (2022). Метод підвищення ефективності симетричного блокового шифрування. *Herald of Khmelnytskyi National University. Technical Sciences*, *311*(4), 191-197. https://doi.org/10.31891/2307-5732-2022-311-4-191-197

37. Торчинський, Р., Стринадко, М., & Шпатар, П. (2025). Secure dynamic stream encryption (sdse): новий підхід до динамічного потокового шифрування з високою криптостійкістю. *Молодий вчений*, (1 (132)), 9-15.

38. Філіп'єва, М. В., & Гвоздецька, К. П. (2021). Порівняння симетричного і асиметричного шифрування. *Міжнародна наукова інтернет-конференція. Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення*, 71.

39. Шабатура, М., Дмитрій, Т., & Бумба, І. (2021). Дослідження стану кібербезпеки сервісів відозв'язку. *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»*, *1*(13), 113-122.

40. Щур Н.О., Покотило О.А. Основи криптології: навч. посібник. – Житомир: Державний університет «Житомирська політехніка», 2021. 120 с. ISBN 978-966-683-597-3 [, 6]

41. Яремчук, Ю. Є. Основи криптографічного захисту інформації : електронний навчальний посібник комбінованого (локального та мережного) використання [Електронний ресурс] / Яремчук Ю. Є., Салієва О. В., Бондаренко І. О. – Вінниця : ВНТУ, 2024. – 139 с.