НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

«На правах рукопису»
УДК 004.056.5:629.7.052

«До захисту допущено»
Завідувач кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2026 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Математичні методи криптографічного захисту інформації»

зі спеціальності: 113 Прикладна математика
на тему: **«Розробка малоресурсного блокового шифру тимчасової стійкості БПЛА»**

Виконав:
студент IV курсу, групи ФІ-23ФІ-23
Коробкіна Софія Сергіївна _____

Керівник: Ковальчук
д.т.н., проф. каф. ММЗІ
Ковальчук Людмила Василівна _____

Рецензент:
доцент кафедри ММАД, к.ф-м.н.
Терещенко Іван Миколайович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ — 2026

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти — перший (бакалаврський)
Спеціальність — 113 Прикладна математика,
ОПП «Математичні методи криптографічного захисту інформації»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2026 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Коробкіна Софія Сергіївна

1. Тема роботи: «*Розробка малоресурсного блокового шифру тимчасової стійкості БПЛА*», науковий керівник дисертації: д.т.н., проф. каф. ММЗІ Ковальчук Людмила Василівна,

затверджені наказом по університету №__ від «__» _____ 2026 р.

2. Термін подання студентом роботи: «__» _____ 2026 р.

3. Об'єкт дослідження: процес формування та застосування S-блоків

4. Предмет дослідження: є алгоритм генерації S-блоку, для шифру PRESENT

5. Перелік завдань:

— провести огляд опублікованих джерел за тематикою дослідження

— провести аналіз та порівняння малоресурсних алгоритмів

— розробити алгоритм генерування S-блоків

— проаналізувати S-блок, згенерований алгоритмом

6. Дата видачі завдання: 10 вересня 2025 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2025 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-жовтень 2025 р.	Виконано
3	Ознайомлення з малоресурсними алгоритмами шифрування	Жовтень-грудень 2025 р.	Виконано
4	Аналіз малоресурсних алгоритмів, та їх порівняння	Січень-лютий 2026 р.	Виконано
5	Формування вимог для алгоримту	Березень-квітень 2026 р.	Виконано
6	Написання алгоритму, та перевірка	Квітень-травень 2026 р.	Виконано
7	Оформлення дипломної роботи	Травень-червень 2026 р.	Виконано

Студент _____ Софія КОРОБКІНА

Керівник _____ Людмила КОВАЛЬЧУК

РЕФЕРАТ

Кваліфікаційна робота містить: 34 стор., 1 рисунки, 2 таблиць, 6 джерел.

Мета дослідження є дослідження малоресурсного блокового шифру тимчасової стікості в БПЛА, що надійно захищає інформацію в умовах своєї роботи.

Об'єктом дослідження є процес захисту інформації, що передається та оброблється в БПЛА

Предметом дослідження є алгоритми побудови малоресурсних блокових шифрів тимчасової стійкості.

У ході виконання дослідження було розглянути проблематику сучасної малоресурсної криптографії, чому вона виникла та як розвивається. Також розглянуто два алгоритми шифрування потоковий ENOCORO та блоковий шифр. Після чого розроблений алгоритм генерації S-блоку, для модифікованої версії алгоритму PRESENT

Наприкінці анотації великими літерами зазначаються ключові слова. Ось так:

МАЛОРЕСУРСНА КРИПТОГРАФІЯ, СИМЕТРИЧНА КРИПТОГРАФІЯ, МАЛОРЕСУРСНІ АЛГОРИТМИ, РОЗРОБКА АЛГОРИТМУ

ABSTRACT

The qualification work contains: 34 pages, 1 figures, 2 tables, 6 sources.

The purpose of the study is to study a low-resource block cipher of temporal consistency in a UAV, which reliably protects information in its operating conditions.

The object of the study is the process of protecting information transmitted and processed in a UAV

The subject of the study is algorithms for constructing low-resource block ciphers of temporal consistency.

During the study, the issues of modern low-resource cryptography were considered, why it arose and how it is developing. Two encryption algorithms, the stream ENOCORO and the block cipher, were also considered. After that, an S-block generation algorithm was developed, for a modified version of the PRESENT algorithm

Keywords are indicated in capital letters at the end of the annotation. Here it is:

LOW-RESOURCE CRYPTOGRAPHY, SYMMETRIC CRYPTOGRAPHY, LOW-RESOURCE ALGORITHMS, ALGORITHM DEVELOPMENT

ЗМІСТ

ВСТУП

Актуальність дослідження. Сучасні пристострої часто працюють при обмежених обчислювальних обмеженнях, тому для них існує окремий підрозділ криптографії, малоресурсна криптографія. І деякі алгоритми є залежать від S-блоків, тому приходить ідея видозмінити S-блок та розробити алгоритм, котрий буде їх перебирати.

Метою дослідження є розробка алгоритму генерації S-блоків для малоресурсного блокового шифру PRESENT, щоби підвищити його стійкість та прискорити його при використанні при обмежених ресурів.

1) провести огляд опублікованих джерел за тематикою дослідження;
2) проаналізувати двох представників малоресурсних алгоритмів
3) Формування вимог до 8-бітового блоку
4) Розробити алгоитм, що перевіряє ці вимоги
5) Розробити алгоритм генерації S-блоків

Об'єктом дослідження є процес захисту інформації, що передається та оброблється в БПЛА

Предметом дослідження є алгоритми побудови малоресурсних блокових шифрів тимчасової стійкості.

При розв'язанні поставлених завдань використовувались такі *методи дослідження*: і тут коротенький перелік наприклад, але не обмежуючись: методи комбінаторного аналізу, методи прикладної алгебри, методи комп'ютерного моделювання комп'ютерного

Наукова новизна отриманих результатів полягає, що запровоновано алгоритм генерації S-блоків для малоресурсного блокового шифру PRESENT, що дозволяє генерувати та перевіряти нелійні перестановки, а це в свою чергу підвищує стійкість алгоритму та підвущує швидкість роботи.

Практичне значення результатів полягає в тому що цей алгоритм можна використовувати для генерації S-блоків, а це підвищує стійкість

алгоритму шифрування. Така ідея може допомогти підвищити криптографічну стійкість, без втрати обчислювальних або енергетичних витрат, а це є важливим для пристоїв в умовах обмежених ресурсів.

1 МАЛОРЕСУРСНІ АЛГОРИТМИ ШИФРУВАННЯ

Перед тим як перейти до самої теми, треба розібратися що таке малоресурсна криптографія, і чому вона взагалі була створена.

1.1 Теоритичні основи малоресурсної криптографії

В наш час з кожним днем зростає кількість пристроїв, вони виконують різні завдання пов'язані з даними. Особливо на що варто звернути, це той факт що в різних пристоїв різні здачі, якщо це якісь персональні комп'ютери або можливо сервери, то нам важливо, щоби всі дані були надійно захищені та можемо втрачати на це великий ресурс і час. В ту саму чергу є інші задачі і пристої, де в нас обмежені ресурси, важлива швидка передача, перевірка та розшифрування, але можна пожертвувати якимось якостями в моменті. Саме вирішенням як опитимізувати традиційні методи, і займається малоресурсна криптографія. А тепер давайте визначимо цей термін. Малоресурсна криптографія— це підгалузь криптографії, головною задачею цієї галузі є розв'язок задач по адаптації шифрів для пристоїв з обмеженими ресурсами. При порівнянні класичної та малоресусної можемо знайти одну суттєву різницю, що показує докорінні відміності. В класичній криптографії головною задачею є максимальний рівень криптографічної стікості. Тому питання кількості пам'яті або витарата часу не є гострим питанням. А от в малоресурсній в залежності від використання йти на поступки та знаходити те, що можна дати в умовну жерту заради швидкості або наприклад заради малого використання пам'яті. В цілому розвитком саме цьої галузі криптографії ми можемо завдячити активним розвитком пристоїв, котрим треба підключення до інтернету, такі пристої зазвичай не оснащують дорогими й якісними мікроконтролерами, через

що використання не адаптованих шифрів може призвести наприклад до перегріву і поламки пристої, або довгої роботи, що в обох випадках не влаштовує користувачів. Малоресурсну криптографію, не можливо виділити як окрему галузь, як мінімум через те що математичні механізми, що використовуються такі самі, нам так само важливо забезпечувати конфіденційність, цілісність та автентичність даних, проте є деякі обмеження. Так само тут використовуються блокові і потокові шифри, в залежності від потреб і задач. При цьому всьому треба зазначити одну важливу річ, не завжди обмеження накладаються саме через невеликі ресурси пристою з яким він взаємодіє, а також з тим, з якими пристороями він взаємодіє. Що можна виділити про такі алгоритми це те, що в нас є задача зробити максимально малу кількість складних операцій. Щоби більш точно описати, що саме мається на увазі, то активно використовуються операції XOR, циклічний зсув, прості підстановки і тому подібне. Що відкриває нам можливість зменшити споживання енергії та спростити реалізаію. Задача створення малоресурсного шифру по своїй природі, є досить суперечливою задачею через те, що нам треба поєднати досить не поєднувані речі, що суперечать один одному. Нашою задачею є високий рівень безпеки, але при цьому і мінімальне використання ресурсів. Проте в практичних задачах приходиться шукати баланс між рівнем безпеки, продуктивності, а також тим наскільки складна буде реалізація. В цілому так як головною задачею криптографії є захищеність. Які б обмеження на реалізацію ми не накладали, то алгоритм нікому не буде потрібен, якщо він вразливий до атак. Зазвичай перевіряють стійкість до таких атак як повний перебір ключів, різнецевий та лінійний криптоаналіз, та інші відомі нам популярні атаки. Які б не стояли обмеження, немає сенсу в алгоритмі, який можна легко взаламати. Але щоби визначити малоресурсний алгоритм ефективним або навпаки неефективним, треба знову зазначити, що вся малересурсна криптографія будується на пошуку компромісу між тими ресурсами, що ми маємо а також продуктивністю.

Але як визначити продуктивність в межах таких алгоритмів. Тут нам треба декілька термінів. Споживання енергії, а також потужність, затримка і пропускна здатність. Також в деяких джерелах вимоги до ресурсів, можуть їх називати витратами. Логічно буде розібратися, що впливає на продуктивність. Для початку розглянемо споживання та потужність. Велика кількість пристоїв обмежена саме цим, через те, що в частина пристоїв працюють від батарейок. А вони в свою чергу мають обмежений ресурс та заряд, також може бути проблема через те, що батарейки складно або неможливо зарядити або складно замінити. В свою чергу затримка є критичним фаткором для програм та пристоїв, що виконують задачі в реальному часі. [2], [4]

1.2 Вимоги до малоресурсних криптографічних алгоитмів

Як в будь якому алгоритмі треба чітко визначити та зрозуміти вимоги до цих алгоритмів, деякі речі, які є важливими для нас вже було розглянуто в попередньому розділі. Але давайте розглянемо це більш формалізовано, а також краще можемо дослідити різницю між звичайною криптографією та малоресурсною.

Першим на що варто звертати увагу при розробці малоресурсного шифру, то це те яка в ньому апаратна складність, так як це фактор і впливає, чи буде взагалі пристрій справлятися з роботою та як буде її виконувати, для цього використовують показники, що і вимірюють її щоби визначити наскільки він є тяжким для пристороїв. Розглядаючі далі, то ми повинні звертати увагу на те скільки пам'яті використовує алгоритм, так як для звичайної роботи ми можемо виділити багато ресурса, то тут треба ефективно працювати і використовувати мінімум пам'яті. Рухаючись далі, то нам треба звернути увагу на низьке енергоспоживання. Зазвичай це фактор не є головним, і його не опрацювують так само завзято, як стійкість наприклад, проте в даному випадку деякі пристрої втрачають свої позитивні характеристики, якщо

їх треба часто заряджати, бо треба витрачати на цей час і ресурси. Щоби знизити енергоспоживання, зазвичай це робиться за допомогою зменшення кількості операцій, які виконуються, також заради цього використовують логічні перетворення. Через це переважна кількість таких алгоритмів відмовляються від використання складних арифметичних операцій. Згадаємо пропускну здатність, котру було описано в минулому підрозділі. А також не можна забути про останню вимогу. Це гнучкість та маштабованість, якщо в звичайних задачах перед нами не стає проблема застосування алгоритму для різних пристоїв та задач, то у випадку з малоресусними алгоритмами, дуже корисним стає саме ця функія. Так як один і той самий алгоритм, для присто, де ресурс обмежений, так і для більш сильних пристоїв, для взаємодії, тому для деяких алгоритмів є задачею саме безпека на обмежений час. [3]

1.3 Які бувають малоресусні алгоритми

Є декілька груп за якими ми можемо поділити малоресурсні алгоритми, це як і в звичайній криптографії потокові, блокові. Саме через те що таких груп декілька, то їх треба розглянути поближче.

Першою групою, котру ми розглянемо будуть потокові шифри, основна їх ідея полягає в тому, вони шифрують усі дані одним потоком, звідки і йде назва. Такий підхід допомогає нам знизити затримки, пов'язані з обробкою даних, а також зменшити виокристання пам'яті. Зазвичай їх використовують в роботі з передачею даних в реальному часі. Проте в такому випадку виникають проблеми так як потокові шифри є чутливими до повторного використання гамма. Також такі алгоритми є дуже чутливими до синхронізації, через невеликий зсус в потоці може статися помилка при дешифруванні. А також аналіз безпеки таких шифрів є складніше, аніж блокових шифрів. Прикладами потокових шифрів є ENOCORO, TRIVIUM, GRAIN.

Іншим видом шифрів є блокові шифри. Такі алгоритми виконують

шифрування фіксованими блоками даними. Головна ідея це застосовувати декілька разів раундові перетворення, якими можуть бути підстановки або перестановки бітів. Такі алгоритми є найбільш поширеними в в цій галузі, так як вони дозволяють балансувати між безпекою та ефективністю. Прикладами таких алгоритмів є PRESENT, GIFT, SIMON. [5]

Висновки до розділу 1

Як тут можна було прочитати, бачимо, що було розібрано що таке малоресурсні алгоритми шифрування, для чого вони використовуються. Що є важливим для таких алгоритмів, та які вони бувають. Головне, що треба зрозуміти, що кожна задача є унікальною, тому різні алгоритми знаходять те чим можна пожертвувати саме в їх випадку, в деяких випадках це навіть стійкість, бо є випадки коли дані треба зберігати від зловмисника пару годин або днів. Разом з цим шифри ділять на два типи, в залежності від реалізації, кожен з них має свої переваги та недоліки, особливо важливо саме задача та конкретний алгоритм. Тому в наступній главі пропоную розглянути один потоковий і блоковий шифр більш детально.

2 ОГЛЯД ДЕЯКИХ СУЧАСНИХ МАЛОРЕСУРСНИХ АЛГОРИТМІВ

В першому розділі ми розгляну що таке в цілому малоресурсні алгоритми, а тепер розглянемо два приклади.

2.1 Потоковий шифр ENOCORO

ENOCORO це малоресурсний потоковий алгоритм, котрий розроблений компанією я Hitachi, а також використовується в стандарті ISO/IEC 29192-3:2012. Головна задача котра стояла перед розробниками цього шифру було запезпечити високий рівень безпеки, але разом з цим мінізувати апаратні та енергетичні витрати. Через це його використовують у сенсерниз мережах та вбудованих контролерах.

Структура алгоритму виглядає настпуним чином. В нас є два головних блок, які виконує алгоритм— це блок генерації гами та блок, що накладає гаму на інформацію. Розглянемо це більш формально:

В блоці генерації гами відбувається два перетворення, для зручності будемо називати їх наступним чином функція переходу та функцію виходу. Для зручності введемо наступні позначення:

$K = V_{128} = (V_8)^{16}$— множина ключів алгоритму

$IV = V_{64} = (V_8)^8$— множина всіх допустимих ініціалізаційних векторів, які можуть подаватися на вхід алгоритму

$S = S_A \times S_B$— множина внутрішніх станів алгоритму

$S_B = V_{256}$— Множина значень стану

$OUT = V_8$—множина вихідних значень

Також треба зазначити, що елементи $k \in K$ та $i \in IV$ будуть позначитися наступним чином:

$k = (k_0, \ldots, k_{15})$ і $i = (i_0, \ldots, i_7)$, де $k_l, i_j, l \in 0, \ldots, 15, j \in 0, \ldots, 7$

Тепер можна описати роботу генераії гами, наступними функціями

1. Функція виходу $Out : S \to OUT$, ця функція на кожному такті своєї роботи видає відрізок гами розміром в 1 байт за таким правилом:

Нехай $s = (a,b) \in S$— це поточний внутрішній стан, в якому $a = (a_0, a_1) \in S_A$, $b = (b_0, b_1, \ldots, b_{31}) \in S_B$, $a_i, b_i \in V_8 i \in \{0,1\}, i \in \{0,\ldots,31\}$, тоді вихідне значенні залежить тільки від a, яке ми визначаємо за формулою $o = OUT(a) = a_1$

2. Функцію переходу $Next : S \to S$, котрий оновлює внутрішній стан за правилом: якщо $s = (a,b) \in S$— поточний внутрішній стан, в якому $a = (a_0, a_1) \in S_A$, $b = (b_0, b_1, \ldots, b_{31}) \in S_B$, $a_i, b_i \in V_8 i \in \{0,1\}, i \in \{0,\ldots,31\}$, тоді наступний стан s' визначаємо а формулою

$$s' = Next(s) = Next(a,b) = (a',b'),$$

де $a' = \alpha(a,b) = (a'_0, a'_1) = (s_8(b_{16}), s_8(b_{29})) \oplus L(s_8(b_2) \oplus a_0, s_8(b_7) \oplus a_1)$ L це матриця 2x2 над полем F_8, і дії до виконуються також в цьому полі. В свою чергу b ми визначаємо таким чином:

$$b'_i = \begin{cases} b_{i-1}, \text{якщо} i \notin \{0,3,8,17\} \\ b_{31} \oplus a_0, \text{якщо} i = 0 \\ b_2 \oplus b_6, \text{якщо} i = 3 \\ b_7 \oplus b_{15}, \text{якщо} i \end{cases}$$

А тепер перейдемо до блоку, де ми накладаємо гаму на інформацію. Тут ми просто працюємо по формулі

$$c_i = m_i \oplus \gamma_i, i \in \overline{1,\ldots,k}$$

А розшифрування проводиться так само, але навпаки

$$m_i = c_i \oplus \gamma, i \in \overline{1,\ldots,k}$$

Проте перед виконанням щифрування інформації, нам треба виконати ініціалізації внутрішнього стану. Спочатку ми вводимо початкові значення, а після проводимо 96 тактів функції Next.

Цей алгоритм був стандартизований, тому назвати його поганим не

можна. Проте в цьому алгоритмі є деякі проблеми, через які він не підходить для деяких завдань. Так як алгоритм є потоковим, він дуже залежить від того наскільки правильно вводять початкові значення. Щоби була згенерована однакова гамма функція, так як від неї і залежить шифрування та розшифрування. Разом з цим йде проблематика синхронізації, так як якщо в якийсь момент була втрачена цілісність, або виникла помилка передачі даних, то виникає гамма може згенерувати по різному, тому і шифруваня та розшифрування може відбутися не правильно. А також варто зазничити що через велику кількість раундів він працює занадто довго. Тому давайте розгялнемо інший шифр. [6]

2.2 Блоковий шифр PRESENT

Блоковий алгоритм PRESENT був опублікований в 2007 році групою авторів таких як A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. Цей шифр складається з 31-го повного раунду, та вкороченно 32-го раунду. Довжина блоку 64 біти, а ключ 80 або 128 бітів. Щоби не описувати шифрування є наступна діаграма.

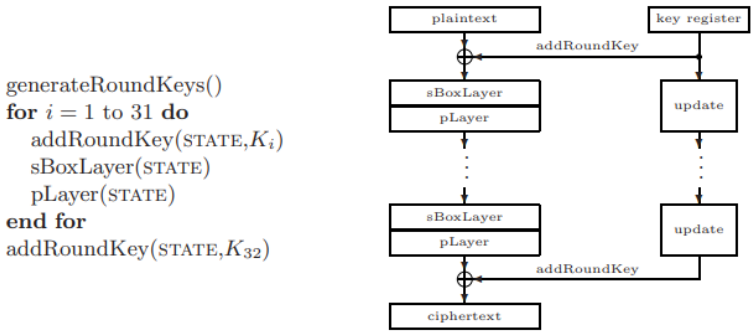

Fig. 1. A top-level algorithmic description of PRESENT

Рисунок 2.1 – Структура алгоритму

Джерело: [1]

І більш формальний опис алгоритму. Перетворення одного раунду

виглядає наступним чином: Позначимо $x \in V_{64}$ як вхід, і $k \in V_{64}$ як ключ раунду. Тепер визначимо функцію раунду ось так:

$$F(x,k) = P(S(x \oplus k)),$$

в свою чергу P та S задані таким чином:

Перетворення $S : V_{64} \to V_{64}$ є підстановкою, і водночас для $u = (u_{15}, \ldots, u_0) \in V_{64}, u_i \in V_4, i = \overline{0, 15}$, дійсним є настпуне

$$S(u) = S(u_{15}, \ldots, u_0) = (s(u_{15}), \ldots, s(u_0)),$$

і перетворення s задається наступною таблицею: А перетворення

Таблиця 2.1 – Назва таблиці

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

$P : V_{64} \to V_{64}$ є бітовою перестановкою, і водночас для $u = (u_{63}, \ldots, u_0) \in V_{64}$, дійсним є настпуне:

$$P(u) = P(u_{63}, \ldots, u_0) = (u_{p(63)}, \ldots, u_{p(0)}),$$

і перетворення $p : 0, 1, \ldots, 63 \to 0,1, \ldots, 63$ є перестановка, яка визначається за настпуним правилом

$$p(i) = \begin{cases} 16 \cdot i mod 63, \text{ якщо } i \neq 63 \\ 63, \text{ якщо } i = 63 \end{cases}$$

Після всіх цій дій треба розглянути ключовий розклад. Нехай в нас є заданий ключ шифрування:

$$K = (k_{79}, \ldots, k_0) \in V_{80}$$

Тепер ключ для першого раунду $K_1 = (k_{63}, \ldots, k_0) \in V_{64}$, визначається як

$$K_1 = (k_{63}, \ldots, k_0) = (k_{79}, \ldots, k_{16}),$$

А далі до вектрора $K = (k_{79}, \ldots, k_0)$ ми застосовуємо наступні перетворення:

Перше, до вектора $K = (k_{79}, \ldots, k_0) \in V_{80}$ виконуємо зсув на 61 позицію вліво

Друге, перші 4 значення вектора $(k_{79}, \ldots, k_{76})$ замінюються на $S(k_{79}, \ldots, k_{76})$

Трете, виконуємо XOR бітів (k_{19}, k_{15}) та round_couter, що дорівнює номеру раунда, що подається у вигляді 5-бітового вектору.

Ці три дії ми повторюємо для того щоби зформувався ключ на кожному раунді.

Підсумовуючи треба зазначити, що в PRESENT є ще скорочена версія, в якій зменшується кількість раундів, проте таке застосування є не надійним, проте для деяких застосувань це має сенс. А от стандартний PRESENT є ефективним алгоритмом, так як в нього низькі апаратні витрати. Але він використовує 64-бітний блок, та велика кількість раундів роблять його не зручним для використання в сучасних пристоях, так як він задовго працює, а зі зменшенням кількості раундів зменшується стійкість. [1]

Висновки до розділу 2

В цьому розділи ми розглянули, двох представників малоресурсних алгоритмів. Потоковий ENOCORO не зручний через його вразливіть до точної передачі даних, а в умовах поганої передачі даних, повністю може зламатися шифрування або розшифрування. В свою чергу PRESENT набагато приємніший в цьому плані, але тут викикає інша проблема, в якій час роботи алгоритма та довжина блоку роблять його теж

проблематичним для використання

3 РОЗРОБКА АЛГОРИТМУ, ЩО ПЕРЕВІРЯЄ НАЯВНІСТЬ В S-БЛОКУ ВИЗНАЧЕНИХ РІЗНИЦЕВИХ ВЛАСТИВОСТЕЙ

3.1 Формування вимог до 8-бітового, які є аналогічними до S-блоку алгоритму PRESENT

В цьому розділи ми використаємо ідею L-PRESENT, також використаємо ідею зменшення кількості раундів, проте в попередньому розділі вже було зазначено, що це не працює, тому буде використана ідея збільшення кількості пам'яті, шляхом змінини S-блоку з 4-бітового на 8-бітовий. Але перед цим треба сфомулювати до цього S-блоку.

Першою вимогою є те, що ця функція перестановки повинна бути бієктивною. Запишемо формально цю вимогу

$$\forall x_1, x_2 \in \{0,1\}^8, x_1 \neq x_2 \Rightarrow S(x_1) \neq S(x_2)$$

Друга вимога накладається на диференціальні характеристики. Тобто щоби в нас під час підбору такої перестановки не було небажаних диференціальних переходів для однобітних різниць. Нехай в нас є α та β, які визначаються наступним чином: $\alpha \in A$— множина всіх 8-бітних векторів ваги один, отже

$$A = \{1, 2, 4, 8, 16, 32, 64, 128\}$$

$\beta \in B$— аналогічна множина

$$B = \{1, 2, 4, 8, 16, 32, 64, 128\}$$

Після чого обчислюємо таблицю за наступною формулою:

$$D(\alpha,\beta) = \frac{1}{2^8}\sum_{x=0}^{255}\delta(S(x\oplus\alpha)\oplus S(x),\beta),$$

де

$$\delta = \begin{cases} 1, x = y \\ 0, x \neq y \end{cases}$$

І це значення D повинно задовільняти нерівність $D \leqq \frac{4}{256} = 2^{-6}$.

3.2 Розробка алгоритм, який перевіряє вимоги

Перед тим як написати алгорим, який буде генерувати потрібні нам S-блоки треба розробити алгоритм, який зможе перевірити його належність до за вимогами, які наведені в пореденьому розділі. Мова для написання такого алгоритму була обрана Python, через його зручність та велику кількість бібліотек. Ось як виглядає цей алгоритм:

```
def delta(a, b):
    if a!=b:
        return 0
    else:
        return 1

def ddeeee(alfa:int, beta:int, sblok:list, v):

    sum=0
    for x in v:
        x_alfa=x^alfa
        s_sx=sblok[x_alfa]^sblok[x]
        sum+=delta(s_sx, beta)
    sum=sum/pow(2, 8)
```

```
    return sum
```

Тут можна побачити, що особливої проблеми в написанні такої програми немає проблеми. І вона працює швидко, можна переходити далі.

3.3 Розробити алгоритм генерації S-блоку із заданими властивостями

Так як в попередньому розділі ми вже створили перевірку для S-блоків, то тепер можна і почати генерацію,

```
from multiprocessing import Process, cpu_count, Value
import time
import random


def listic():
    s = list(range(256))
    random.shuffle(s)
    return s

def poisk(alfa, beta, v):

    sblok = listic()

    for a in alfa:
        for b in beta:
            if ddeeee(a, b, sblok, v):
                return False

    with open("sbloks.txt", "a") as f:
        print(sblok, file=f)
```

```
    return True
def funk(start, limit, counter, alfa, beta, v):

    local = 0

    while time.time() - start < limit:

        poisk(alfa, beta, v)
        local += 1

        if local % 1000 == 0:
            with counter.get_lock():
                counter.value += 1000
            local = 0
```

Цей код хоч і працює правильно, але при роботі в декілька діб, хоч і з перервами, не зміг знайти нічого. Перша функція listic виконую генерацію перестановки. Наступна функція poisk визиває listic, а потім перевіряє її, і якщо така перестановка знаходиться, то вона записується у окремий файл. Цю функцію визивається далі обмежену кількість разів, спочатку я пробувала запускати цю функцію обмежену кількість разів, наприклад 2^{20}, проте результатів це не давало, і програму треба було запускати декілька разів, тому потім прийшла ідея запустити роботу не на кількість варіантів, які треба перебрати, в просто на час. Варто зазначити, що ця програма працює максимульну кількість потоків, це пришвидшило роботу, але не допомогло знайти перестановку.

Ці всі дії не призвели до жодного результату, тому треба було переписати алгоритм, бо цей не допоможе нам. Тому прийшла інша ідея в голову,

використати формулу такого вигляду:

$$s(x) = A((Bx)^{-1}),$$

де A та B є лінійними оборотними перетвореннями над F_{2^8}. В свою чергу обернення виконується, як відображення в полі $F_{2^8} = F_2(\alpha)$, де α це корінь незвідного многочлена $m(t) = t^8 + t^4 + t^3 + t + 1 \in F_2[X]$, а також $0^{-1} = 0$ Тому був написаний інший алгоритм, що знаходить потрібну нам перестановку, ось як виглядає цей алгоритм

```
import random

def mnog(a, b):
    result=0
    for _ in range(8):
        if b&1:
            result ^=a
        f=a&0x80
        a=(a<<1)&0xFF
        if f:
            a^=0x1B
        b>>=1
    return result



def stepin(a, p):
    st=1
    for _ in range(p):
        st=mnog(st, a)
    return st
```

```
def obernene(a):
    if a==0:
        return 0
    else:
        return stepin(a, 254)

def linia(a, mat):
    result=0
    for i in range(8):
        k=0
        for j in range(8):
            if (mat[i]>>j) &1:
                k^=(a>>j)&1
        result=result|(k<<i)
    return result

def randoma_matrixa():
    b=[]
    vuc=set()

    while len(b)<8:
        k=random.randint(1, 255)
        if k not in vuc:
            b.append(k)
            vuc.add(k)
    return b


def sblok(a, b):
    sbox=[]
```

```
    for i in range(256):
        x=linia(i, b)
        x=obernene(x)
        x=linia(x, a)
        sbox.append(x)
    return sbox



def perev(sbox):
    return len(set(sbox)) == 256

def delta(a, b):
    if a!=b:
        return 0
    else:
        return 1

def ddeeee(alfa:int, beta:int, sblok:list, v):

    sum=0
    for x in v:
        x_alfa=x^alfa
        s_sx=sblok[x_alfa]^sblok[x]
        sum+=delta(s_sx, beta)
    sum=sum/pow(2, 8)
    return sum


def generasia():
    alfa = []
```

```
    with open("alfa.txt", "r") as f:
        for line in f:
            alfa.append(int(line.strip()))

    beta = []
    with open("alfa.txt", "r") as f:
        for line in f:
            beta.append(int(line.strip()))

    v = []
    with open("chisl.txt", "r") as f:
        for line in f:
            v.append(int(line.strip()))
    while True:
        a=randoma_matrixa()
        b=randoma_matrixa()
        sbox= sblok(a, b)

        if perev(sbox):
            prov = True
            for a1 in alfa:
                for b1 in beta:
                    if ddeeee(a1, b1, sbox, v) > 4/256:
                        prov = False
                        break
                if not prov:
                    break
            if prov:
                return sbox

print(generasia())
```

Тепер треба було перевіряти ці блоки на їх стійкість до різнецевого криптоалізу, тому треба було обрахувати чи є він стійким. І було виявленно, що він є стійким до різнецевого криптоаналізу

3.4 Детальний розгляд однієї з перестановок

В цьому підрозділі ми розглянемо однин з S-блоків, котрий згенерував алгоритм, що був наведений вижче, та аналіз стійкості алгоритму з таким S-блоком. В наведеній нижче таблиці, означає, що

Таблиця 3.1 – Перестановка S-блоку

00	6C	ED	31	F2	C8	DC	9F	73	2A	25	5A	41	79	19	74
9D	13	14	E5	2E	91	50	53	69	CC	84	EE	E2	C9	B2	D4
1C	8E	61	2F	B5	12	0C	F1	75	67	A8	87	7E	08	58	E9
51	DB	6B	B8	62	D5	D1	46	89	7B	56	BB	0E	04	23	FE
03	EB	76	B4	B1	A0	BF	5C	7C	2D	68	B9	DE	07	02	9C
AD	9E	30	72	96	F8	64	D9	0B	18	A3	F0	1A	DF	39	7D
D2	43	EF	0A	05	DD	0F	3A	F3	DA	F4	CB	4C	3B	54	90
44	26	D8	B3	35	47	C5	3D	80	8D	C7	98	55	AF	16	83
82	AA	B0	B6	E6	86	48	E4	C3	11	9A	95	65	FA	8F	E3
24	5E	F7	37	36	CA	FB	8B	2B	C0	FD	A1	A7	AB	66	7A
40	E0	D7	63	4D	2C	52	E7	6E	CF	C4	6F	77	81	BD	1F
10	15	A6	49	D0	E1	5B	93	06	38	3E	97	A9	B7	28	6D
21	8C	9B	FF	5D	1B	22	F9	CD	42	EA	29	BA	3F	F5	33
57	71	1D	20	EC	BC	94	09	7F	60	8A	45	D3	1E	59	AC
78	17	4F	CE	32	88	D6	3C	C2	85	0D	BE	5F	4B	E8	C6
AE	34	C1	A4	4A	01	27	99	FC	6A	A2	92	F6	4E	70	A5

перетворення виглядає наступним чином $s(00) = 00$, $s(01) = 6C$, $s(02) = ED$, $s(FF) = A5$. Отже далі проаналізуємо дві теореми, для побудови шифру на такій перестановці.

Теорема 3.1. *В 4-раундового шифрі кількість активних S-блоків є не меншою ніж 6.*

Доведення

Для довення треба розлянути декілька наступних випадків

Випадок 1

В кожному раунді активними є як мінімум два S-блоки.

Очевидно, що в такому випадку кількість активних S-блоків за 4 раунди становить, як мінімум 6.

Випадок 2

У першому раунді активним є тільки один S-блок. Тому тут, поділемо цей випадок на два підвипадки.

Випадок 2а

Вхідна різниця цього блоку містить тільки один ненульвий біт. Отже, в другому раунді так само буде лише один активний S-блок, разом з цим його взідна різниця містить один ненульовий біт. Якщо згадати, що диференційну властивість за якої максимальна диференціальна ймовірність не повинна перевищувати $\frac{1}{64}$, а далі звернемося до другої властивості наведеної в першому підрозділі. Тому вихідна різниця цього блока повинна містити якмінімум два ненульві біти.

Відповідно до нашої перестановки, то в третьому раунді в нас також будуть активними як мінімум два S-блоки, а разом з цим кожен з них матиме на вході рівно один ненульовий біт.

Так як вихідна різниця кожного з цих S-блоків містить, що найменше два ненульові біти, то і в четвертому все буде аналогічно. Підсумовуючі загальна кількість активних S-блоків дорівнює, як мінімум

$$1 + 1 + 2 + 2 = 6$$

Випадок 2б

Вихідна різниця активного S-блоку першого раунду містить якмінімум два ненульові біти.

З цього слідує, що в другому раунді активними будуть якмінімум два S-блоки, кожен з яких буде мати один ненульовий біт на вході.

Кожен з цих блоків складає вихідну різницю з якумінімум двома ненульовими бітами, тому в третьому раунді так само активними будуть не менше, ніж двох S-блоків.

Підсумовуючі якмінімум один активний S-блок четвертого раунду, отримуємо

$$1 + 1 + 2 + 2 = 6$$

Випадок 3

У другому раунді активним є тільки один S-блок.

Вхідна різниця містить один ненульовий біт.

У той час в першому раунді також активний лише один S-блок. Вихідна різниця є якмінімум двобітною, а в третьому раунді активними будуть якмінімум два S-блок. Їхні вихідні різниці також містять якмінімум два ненульові біти, отже в четвертому раунді активними будуть не менше двох S-блоків.

Таким чином, сумарна кількість активних S-блоків становить не менше шести.

Випадок 4

У третьому раунді активний лише один S-блок.

Можливі два підвипадки.

Випадок 4а

Вихідна різниця містить один ненульовий біт.

Отже в другому раунді існує якмінімум два активних S-блоки, а в четвертому— якмінімум один.

Так як в третьому раунді активний лише один S-блок, то активні S-блоки другого раунду повинні мати вихідні різниці, що містять лише один активний біт. Тоді їхні вхідні різниці містять якмінімум два ненульові біти, а отже, в першому раунді активними є не менше двох S-блоків.

Тому загальна кількість активних S-блоків становить якмінімум шість.

Випадок 4б

Вихідна різниця містить якмінімум два ненульові біти.

Тоді в четвертому раунді активними будуть якмінімум два S-блоки, а в другому — не менше одного.

Якщо в другому раунді активний рівно один S-блок, то його вихідна різниця містить один ненульовий біт, а вхідна різниця— якмінімум два ненульові біти. Отже, у першому раунді активними є якмінімум два S-блоки.

У протилежному випадку в другому раунді активними є якмінімум два S-блоки, а в першому— якмінімум один.

В обох ситуаціях сумарна кількість активних S-блоків становить не менше шести.

Випадок 5

У четвертому раунді активний лише один S-блок.

Поділимо на два підвипадки

Випадок 5а

Вихідна різниця містить один ненульовий біт.

Тоді в третьому раунді активними є якмінімум два S-блоки, причому кожен із них формує вихідну різницю з одним ненульовим бітом.

Звідси випливає, що в другому раунді активними будуть якмінімум два S-блоки, а загальна кількість активних S-блоків становитиме не менше шести.

Випадок 5б

Вихідна різниця містить якмінімум два ненульові біти.

Тоді в третьому раунді активним є якмінімум один S-блок. Якщо він єдиний, то в другому та першому раундах активними будуть якмінімум по два S-блоки.

Отже, і в цьому випадку сумарна кількість активних S-блоків якмінімум ніж шість.

Таким чином, теорему доведено.

Теорема 3.2. *Імовірність диференціальної характеристики шифру зверху обмежується такими значеннями:*
Для 4-раундової версії $\leqslant 2^{-36}$
Для 6-раундової версії $\leqslant 2^{-48}$
Для 7-раундової версії $\leqslant 2^{-60}$
Для 8-раундової версії $\leqslant 2^{-72}$
Доведення цієї теореми прямо випливає з першою та другої властивості для нашого алгоритму.

А при обрахунку диференційної стійкості такої перестановки отримаємо наступне значення 0.03125, що показує, що вона є стійкою.

Висновки до розділу 3

Отже в цьому розділі було написано алгоритм, що знаходить потрібні нам S-блоки та потім перевіряє їх на те, підходять вони нам чи ні. Як ми могли побачити не кожен алгоритм нам підходить, треба підходити до цього більш складними способами. Такий спосіб може допомогти прискорити роботу приблизно в 6 разів. Тому така реалізація є корисною. А згенерована нею перестановка є диференційно стійкою.

ВИСНОВКИ

Отже за виконанням цієї роботи були виконані такі завдання, як розбір того що таке малоресусні алгоритми. Для чого вона треба, та які задачі вона виконує. Разом з тим є певні відміності від звичайної криптографії, тому є інші вимоги до алгоритмів. Разом з цим ми і знайшли ці вимоги та розібрали їх.

В цьому розділи ми розглянули, двох представників малоресурсних алгоритмів. Потоковий ENOCORO не зручний через його вразливіть до точної передачі даних, а в умовах поганої передачі даних, повністю може зламатися шифрування або розшифрування. В свою чергу PRESENT набагато приємніший в цьому плані, але тут викикає інша проблема, в якій час роботи алгоритма та довжина блоку роблять його теж проблематичним для використання.

А в останньому розділі було написано алгоритм, що знаходить потрібні нам S-блоки та потім перевіряє їх на те, підходять вони нам чи ні. Як ми могли побачити не кожен алгоритм нам підходить, треба підходити до цього більш складними способами. Такий спосіб може допомогти прискорити роботу приблизно в 6 разів. Тому така реалізація є корисною.

Підсумовуючі всі задачі, що були поставлені були виконані успішно, і цей алгоритм може допомогти генерувати різні перестановки видозмінюючи алгоритм, проте ідейно його не змінюючи. А ці перестановки є стійкими до диференційного аналізу.

ПЕРЕЛІК ПОСИЛАНЬ

[1] A. Bogdanov та ін. «PRESENT: An Ultra-Lightweight Block Cipher». В: *Cryptographic Hardware and Embedded Systems - CHES 2007*. За ред. Pascal Paillier та Ingrid Verbauwhede. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, с. 450—466. ISBN: 978-3-540-74735-2.

[2] Thomas Eisenbarth та ін. «A Survey of Lightweight-Cryptography Implementations». В: *IEEE Design & Test of Computers* 24.6 (2007), с. 522—533. DOI: 10.1109/MDT.2007.178.

[3] Jaber Hosseinzadeh та Abbas Ghaemi Bafghi. *Evaluation of Lightweight Block Ciphers in Hardware Implementation: A Comprehensive Survey*. 2017. DOI: 10.48550/ARXIV.1706.03878. URL: https://arxiv.org/abs/1706.03878.

[4] Mohsin Khan, Elisavet Kozyri та Håvard Dagenborg. *Systematic Review of Lightweight Cryptographic Algorithms*. 2026. DOI: 10.48550/ARXIV.2602.14731. URL: https://arxiv.org/abs/2602.14731.

[5] Vishal A. Thakor, M. A. Razzaque та Muhammad R. A. Khandaker. *Lightweight Cryptography for IoT: A State-of-the-Art*. 2020. DOI: 10.48550/ARXIV.2006.13813. URL: https://arxiv.org/abs/2006.13813.

[6] Dai Watanabe та ін. «Update on Enocoro stream cipher». В: *2010 International Symposium On Information Theory & Its Applications*. IEEE, жовт. 2010, 778–783. DOI: 10.1109/isita.2010.5649627. URL: http://dx.doi.org/10.1109/ISITA.2010.5649627.