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

«На правах рукопису»
004.056.55:519.2

«До захисту допущено»
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2023 р.

Дипломна робота

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

зі спеціальності: 113 Прикладна математика
на тему: **«Особливості розподілів імовірностей диференціалів S-блоків спеціального виду»**

Виконав:
студент IV курсу, групи ФІ-94
Маринін Іван Павло Ігорович _____

Керівник:
доцент кафери ММЗІ, к.т.н.
Яковлєв Сергій Володимирович _____

Рецензент:
старший викладач кафедри ІБ, к.ф-м.н.
Рибак Олександр Владиславович _____

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

Студент _____

Київ — 2023

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
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2023 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Маринін Іван Павло Ігорович

1. Тема роботи: «*Особливості розподілів імовірностей диференціалів S-блоків спеціального виду*», науковий керівник роботи: доцент кафери ММЗІ, к.т.н. Яковлєв Сергій Володимирович,

затверджені наказом по університету №__ від «__» _____ 2023 р.

2. Термін подання студентом роботи: «__» _____ 2023 р.

3. Об'єкт дослідження: *інформаційні процеси у симетричних криптосистемах*

4. Предмет дослідження: *методи виявлення прихованої алгебраїчної структури S-блоків*

5. Перелік завдань: *провести огляд опублікованих джерел за тематикою дослідження та аналіз конкретних випадків відновлення прихованої алгебраїчної структури S-блоку; експериментально перевірити гіпотези про розподіли ймовірностей диференціалів відносно різних операцій для S-блоків зі структурою фейстель-подібної мережі.*

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: *Презентація доповіді*

7. Дата видачі завдання: 10 вересня 2022 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2022 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-жовтень 2022 р.	Виконано
3	Огляд наявних досліджень щодо криптоаналізу перетворення S-блоку	Листопад-грудень 2022 р.	Виконано
4	Аналіз відновлення структури перетворення S-блоку у хеш-функції Стрибога	Січень-лютий 2023 р.	Виконано
5	Дослідження статистичних методів інтерпретації значень перестворення S-блоку	Березень 2023 р.	Виконано
6	Побудова програмної реалізації для перевірки статистичних гіпотез, її застосування та аналіз отриманих результатів	Квітень-Травень 2023 р.	Виконано
7	Оформлення та захист дипломної роботи	Червень 2023 р.	Виконано

Студент _____ Іван Павло МАРИНІН

Керівник _____ Сергій ЯКОВЛЄВ

РЕФЕРАТ

Кваліфікаційна робота містить: 62 стор., 20 рисунків, 13 таблиць, 20 джерел.

Дана робота містить аналіз та застосування методів відтворення алгебраїчної структури S-блоку, яка є невідомою. У роботі викладені деякі алгоритми та статистичні методи криптоаналізу, які допоможуть в оцінюванні числових перетворень конкретного виду у симетричних криптосистемах.

Робота викладена у 3 частини: огляд та збір загальних теоретичних даних; опис відтворення прихованої алгебраїчної структури S-блоку на прикладі хеш-функції Стрибога; самостійне проведення статистичного дослідження, викладення та аналіз отриманих результатів.

У ході проведеного дослідження окреслено алгоритм, який виявляє потенційні принципи побудови перетворення, наведено методи, які дозволяють аналізувати S-блок знаючи лише його представлення. Також проведено статистичне дослідження про ймовірнісний розподіл значень, які були отримані внаслідок застосування одного із вищезгаданих методів, та висновок отриманих результатів.

СИМЕТРИЧНА КРИПТОГРАФІЯ, СИМЕТРИЧНЕ БЛОКОВЕ ШИФРУВАННЯ, МЕТОДИ СТАТИСТИЧНОГО КРИПТОАНАЛІЗУ, ПЕРЕТВОРЕННЯ S-БЛОК, ВІДНОВЛЕННЯ ПРИХОВАНОЇ АЛГЕБРАЇЧНОЇ СТРУКТУРИ

ABSTRACT

The thesis consists of 62 pages, 20 figures, 13 tables, and 20 references.

This work includes an analysis and application of methods for reconstructing the algebraic structure of an unknown S-box. The paper presents algorithms and statistical methods for cryptanalysis, which help in evaluating numerical transformations of a specific type in symmetric cryptosystems.

The paper is divided into three parts: an overview and collection of general theoretical information, a description of reconstructing the hidden algebraic structure of the S-box using the example of the Streebog hash function, and independent statistical research, presentation, and analysis of the obtained results.

During the conducted research, an algorithm is outlined that identifies potential principles for constructing the transformation, and methods are provided that allow for the analysis of the S-box based solely on its representation. Additionally, a statistical investigation is conducted on the probability distribution of values obtained through the application of one of the aforementioned methods, along with a conclusion on the obtained results.

SYMMETRIC CRYPTOGRAPHY, SYMMETRIC BLOCK CIPHERS, STATISTICAL CRYPTANALYSIS METHODS, S-BOX TRANSFORMATIONS, RECONSTRUCTION OF HIDDEN ALGEBRAIC STRUCTURE.

ЗМІСТ

ВСТУП

Актуальність дослідження. Полягає в її внеску в галузь криптографії. S-блок – є фундаментальними компонентами багатьох криптографічних алгоритмів, а їхні властивості, такі як нелінійність, диференціальна однорідність і стійкість до криптоаналізу, є вирішальними для забезпечення безпеки цих алгоритмів. Використовуючи статистичні методи для аналізу криптографічних властивостей S-блока, робота має на меті забезпечити кількісну та об'єктивну оцінку їх міцності та вразливості. Статистичні методи можуть допомогти виявити будь-які шаблони, упередження або слабкі місця в дизайні S-блока, якими можуть скористатися зловмисники.

Метою дослідження є удосконалення методів виявлення та відновлення прихованих алгебраїчних структур криптографічних S-блоків. Для досягнення мети необхідно розв'язати **задачу дослідження**, яка полягає у виявленні відмінностей у статистичних характеристиках S-блоків, які мають специфічну будову. Для розв'язання задачі необхідно виконати такі завдання:

1) провести огляд опублікованих джерел за тематикою дослідження;

2) провести аналіз конкретних випадків відновлення прихованої алгебраїчної структури S-блоку;

3) експериментально перевірити гіпотези про розподіли ймовірностей диференціалів відносно різних операцій для випадкових S-блоків, які мають структуру фейстель-подібної мережі.

Об'єктом дослідження є інформаційні процеси у симетричних криптосистемах.

Предметом дослідження є методи виявлення прихованої алгебраїчної структури S-блоків.

При розв'язанні поставлених завдань використовувались такі *методи дослідження*: методи лінійної, абстрактної та прикладної

алгебри, математичної та прикладної статистики, дискретної математики, комп'ютерного моделювання.

Наукова новизна отриманих результатів: було показано, що S-блоки, які мають структуру фейстель-подібної мережі, мають розподіли ймовірностей диференціалів відносно різних операцій, які суттєво відрізняються від аналогічних розподілів випадкових S-блоків.

Практичне значення роботи. Одержані результати можна використати для пришвидшення виявлення прихованої алгебраїчної структури типу схеми Фейстеля або її аналогів у таблично заданих S-блоках.

1 ОГЛЯД ТЕОРЕТИЧНИХ ВІДОМОСТЕЙ ПРО ПЕРЕТВОРЕННЯ S-БЛОК

У даному розділі проведено огляд криптопримітиву S-блоку, його застосування та важливості. Також розглянуто деякі методи декомпозиції/зворотньої розробки S-блоку із математичної точки зору, а також теоретичні відомості, які є необхідними для опису.

1.1 Криптопримітив S-блок та методи його побудови

Кожна криптосистема тим краща, чим вона більш стійка він до атак різного роду: лінійних, диференціальних, алгебраїчних тощо. Задля покращення захисту вводиться перетворення під назвою S-блок (з англ. – S-box). Отже, S-блок — це базова складова алгоритмів із симетричним ключем[8], яка виконує заміну вхідного n – бітного вектора у m – бітний вектор, при чому, не обов'язково $n = m$. Математично S-блок визначається як однозначне відображення із вхідного простору у вихідний простір. Він може бути представлений у вигляді таблиці пошуку (з англ. – look-up table, тут і далі – LUT) або алгебраїчного виразу/нелінійної функції, яка визначається наступним чином: $f(x) = y, x \in \{0,1\}^n, y \in \{0,1\}^m$. Зазвичай $n = m = 8$ або 16 біт, що має просте обґрунтування: задля зручного співставлення із двійковим записом шістнадцяткових чисел.

Для кращого сприйняття S-блок представляється у вигляді таблиці. Наприклад, Таблиця 1.1 візуалізує перетворення S-блок у шифрі AES, який перетворює 8 біт входу на 8 біт виходу. У даному випадку таблиця заповнена шістнадцятковими числами: $c3 \longrightarrow 2e$.

Значення S-блоку можуть обиратися або випадковим чином, або виходячи із певних міркувань криптографічних властивостей. У першому

випадку результат не обов'язково дасть кращу стійкість до різного роду атак, радше, навпаки – важко буде передбачити чи така зміна вносить покращення, чи не з'являються певного роду кореляційні, лінійні чи диференційні залежності, тобто результат є випадковим. У другому випадку творці мають обгрунтування стійкості системи із відомими сильними та слабкими сторонами. У деяких випадках алгебраїчна структура утаємничується.

Таблиця 1.1 – представлення S-блока AES[15]

	.0	**.1**	**.2**	**.3**	**.4**	**.5**	**.6**	**.7**	**.8**	**.9**	**.a**	**.b**	**.c**	**.d**	**.e**	**.f**
0.	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1.	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2.	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3.	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4.	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5.	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6.	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7.	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8.	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9.	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a.	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b.	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c.	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d.	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e.	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f.	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

S-блок AES – це нелінійна байтова заміна, яка застосовується незалежно до кожного байту певного поточного стану (входу)[15]. Це перетворення конструюється за допомогою поєднання двох наступних трансформацій:

1. Знаходження мультиплікативного обернення до числа у скінченному полі $GF(2^8)$.

2. Застосування афінного перетворення над полем $GF(2)$:

$b'_i = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$, де, для $0 \leqslant i < 8$, b_i – це i-ий біт байту c_i – це i-ий біт байту і $c = 01100011_2$.

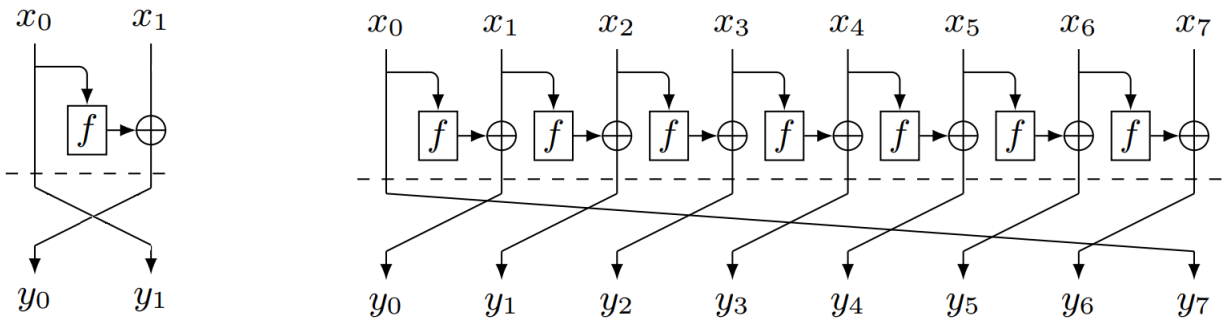
У матричні формі афінна трансформація елементу S-блока може бути представлена як:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

Таким чином і був побудований S-блок AES. Варто зауважити, що, хоча у даному випадку не використовувалася мережа Фейстеля при конструюванні перетворення, проте, вона, так як і їй-подібні схеми, наприклад схема Місті та R-схема, доволі часто застосовуються при проєктуванні заміни.

Класичний вигляд мережі Фейстеля [18] розділяє вхідний масив на дві половини, із лівої частини робить нову праву, а також пропускає ліву через певну функцію, над виходом якої та початковою правою частиною проводиться операція побітового додавання, результат якої записується у ліву частину. Такий процес називається одним раундом мережі. Схематично він зображено на Рисунку 1.1.a. Таке представлення мережі застосовується, наприклад, у DES[17] та Camellia[3]. Також є узагальнена мережа Фейстеля, яка розділяє вхідний вектор на $k \geqslant 2$ частини. Суть операцій не змінюється, тільки у даному випадку вони застосовуються до

всіх частин, нагадуючи, певним чином, різновид лінійного регістру зсуву. Схема роботи зображена на Рисунку 1.1.б. Узагальнені мережа Фейстеля використовуються, наприклад, у SHA-1[14], Mars[7] та CAST-256[1].



(а) **(б)**

Рисунок 1.1 – Мережа Фейстеля: (а) класична, (б) узагальнена.

Схема Містi та R-схема за своєю сутністю не відрізняються від мережі Фейстеля. Відмінність полягає у послідовності застосування перетворення функції f та операції побітового додавання. Схематичне зображення на Рисунку 1.2

(а) **(б)**

Рисунок 1.2 – (а) схема Містi, (б) R.

Зауваження. Функція f, яка використовувалась в усіх схемах вище, може мінятися на іншу із кожним наступним раундом, тобто, у перетворенні не обов'язково використовується лише f – може бути і f_1, і

f_2, і т.д.

1.2 Методи аналізу S-блока

Розглянувши деякі основні методи побудови S-блока необхідно розглянути способи, за допомогою яких можна інтерпретувати значення табличного представлення цього перетворення.

До базових технік оцінювання лінійних та диференційних залежностей S-блока належать побудови Таблиці лінійної апроксимації та Таблиці диференційного розподілу відповідно.

Таблиця лінійної апроксимації

Таблиця лінійної апроксимації (з англ – Linear Approximation Table, тут і далі – LAT) – це інструмент, який використовується для аналізу криптографічних компонентів, зокрема для вивчення нелінійності операцій підстановки, таких як S-блоки[12]. Він забезпечує систематичний спосіб вимірювання лінійних залежностей між вхідними та вихідними бітами криптографічної функції. З математичної точки зору:
нехай $\pi : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ – бієктивне n-бітне відображення. Для n-бітного вектора $X \in \mathbb{Z}_2^n$ позначимо X_i за i-ий біт X. Тоді LAT – це матриця розміром $2^n \times 2^n$, яка визначається наступним чином:

$$LAT_\pi(\alpha, \beta) = \#\{X | X \in \mathbb{Z}_2^n, \bigoplus_{i=1}^{n} X_i \cdot \alpha_i = \bigoplus_{i=1}^{n} \pi(X)_i \cdot \beta_i\},$$

де $\alpha, \beta \in \mathbb{Z}_2^n$ – вхід та відповідний вихід перетворення S-блоку, а операція «$\cdot$» визначається як операція бітового множення (логічного І).

Записи в LAT вказують на міцність лінійних зв'язків між конкретними вхідними та вихідними значеннями S-блока. Більші абсолютні значення позначають сильніші лінійні зв'язки. Також їх можна інтерпретувати як ймовірності. Аналізуючи значення можна оцінити стійкість перетворення до різного роду лінійних та алгебраїчних атак.

Таблиця диференційного розподілу

Таблиця диференційного розподілу (з англ – Differential Distribution Table, тут і далі – DDT) – це інструмент криптографічного аналізу, який використовується для вивчення алгоритмів із симетричним ключем, зокрема блочних шифрів. DDT надає інформацію про розподіл вхідних і вихідних відмінностей для даного блокового шифру. З математичної точки зору:
нехай $\pi : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ – n-бітне відображення. $X \in \mathbb{Z}_2^n$ – n-бітний вектор. Тоді DDT – це матриця розміром $2^n \times 2^n$, яка визначається наступним чином:

$$DDT_\pi(\alpha, \beta) = \#\{X | X \in \mathbb{Z}_2^n, S(X \oplus \alpha) \oplus S(X) = \beta\},$$

де $\alpha, \beta \in \mathbb{Z}_2^n$ – вхід та відповідний вихід перетворення S-блока, S – саме перетворення, а операція «$\oplus$» визначається як операція бітового додавання (виключного АБО).

Диференційна рівномірність – важлива характеристика для аналізу стійкості S-блока до диференційного криптоаналізу[13]. Визначається як: $max\{DDT_\pi(\alpha, \beta), \alpha \in \mathbb{Z}_2^n \backslash \{0\}^n, \beta \in \mathbb{Z}_2^n\}$. Чим нижче значення, тим краще. Найнижче можливе значення диференційної рівномірності це 2. S-блоки із такою диференційною ймовірністю називаються *майже ідеальними нелінійними.*

Варто зауважити, що кожен елемент DDT – це ціле число в межах від 0 до 2^n, де n – кількість біт входу та виходу S-блока. Крайнє ліве верхнє значення у таблиці завжди рівне 2^n, а усі інші елементи у цьому рядку рівні 0. Сума елементів кожного рядка рівна 2^n.

Таблиця 1.2 та Таблиця 1.3 візуалізують розподіл значень DDT для таких криптосистем як Стрибог та Skipjack відповідно[4].

значення	256	0	2	4	6	8
частота	1	38235	22454	4377	444	25
% **від загального**	0.001	58.342	34.262	6.679	0.677	0.038

Таблиця 1.2 – розподіл значень DDT Стрибог

значення	256	0	2	4	6	8	10	12
частота	1	39104	20559	4855	686	69	5	2
% **від загального**	0.001	60.140	31.620	7.467	1.055	0.106	0.008	0.003

Таблиця 1.3 – розподіл значень DDT Skipjack

На даний момент опубліковано доволі багато праць по лінійному та диференційному криптоаналізу S-блока[16]. Так, наприклад, висвітлений дизайн шифру на базі мережі Фейстеля із не менш, ніж 5 раундами, який є стійким до диференційного криптоаналізу[2]; описано хороші критерії для створення S-блока з імунітетом до диференційного криптоаналізу[9]; показано, що існують DES-подібні ітеровані шифри, які не піддаються диференційного криптоаналізу[11]; проведено криптоаналіз шифру DES на базі лінійних залежностей[10]; та багато інших.

Висновки до розділу 1

У даному розділі проведено огляд основних теоретичних відомостей, які стосуються тематики S-блока. Зокрема, наведено визначення даного перетворення та популярні методи при його створенні і описано два методи, які характеризують об'єкт з точки зору лінійних та диференційних залежностей. Задача відновлення прихованої алгебраїчної структури S-блока уже не одноразово розглядалася багатьма дослідниками. Враховуючи, що розробка такого перетворення дозволяє користуватися багатьма математичними суб'єктами, то до різних їх комбінацій потрібно буде застосовувати і різний аналіз. На даний момент задача статистичного аналізу DDT із нетрадиційними операціями доволі погано вивчена, тому прийнято рішення взятися за неї.

2 ПРИКЛАД ВІДНОВЛЕННЯ ПРИХОВАНОЇ АРХІТЕКТУРИ S-БЛОКУ

Серед науковців, які займалися вивченням даної тематики були Алекс Бірюков, Лео Перрін і Олексій Удовенко. Вони сформували загальний алгоритм для декомпозиції прихованої алгебраїчної структури S-блоку та певні методи і критерії, від яких необхідно відштовхуватися при декомпозиції. Також вони знайшли дизайн перетвореня S-блоку, який використовується у таких криптосистемах як російська стандартизована хешфункція Стрибог (з англ. – Streebog), російський стандартизований блочний шифр Кузьнечіка (з англ. – Kuznyechik), блочний шифр СтрибобР1 (з англ. – STRIBOBr1). У цьому розділі усі результати, методи, алгоритми та зображення належать їхнім справжнім власникам [6] [4] [5].

2.1 Узагальнений алгоритм для декомпозиції невідомої структури S-блока

Перш, ніж приступити до огляду алгоритму, необхідно ввести визначення об'єктів, які використовуються у ньому.

• Метод Розпізнавання Образів Поллока (з англ. – Pollock's Pattern Recognition, тут і далі – PPR) — метод, який полягає у перетворенні значень таблиці у зображення задля подальшого аналізу людським зором. Таким чином, можна переглянути усю таблицю водночас, попередньо присвоївши різним коефіцієнтам різні кольори, і впевнитися у (не-)випадковому вибору S-блока.

• LAT — таблиця, яка візуалізує залежність між вхідними та вихідними змінними S-блока наступним чином: назви рядків містять усі можливі суми пар значень перетворення на вході у порядку зростання, а

назви стовпчиків – суми його пар значень на виході. Кожна комірка містить кількість збігів/пар елементів S-блока, які мають таку суму на вході та таку суму на виході. Детальніше у Розділі 1.2.

• DDT — визначається аналогічним чином до LAT, тільки замість суми в усіх випадках розглядається різниця пар елементів S-блока на вході та виході. Детальніше у Розділі 1.2.

• SASAS — мережі, у яких S-блоки дуже великі та побудовані з використанням так званої структури ASASA або ASASASA, де A позначає афінний рівень, а S — нелінійний рівень S-блоку. Запобігання зловмиснику розкласти ці S-блоки на рівні A та S є основою безпеки для цієї схеми.

Нижче наведено алгоритм–схему, якої автори рекомендують дотримуватися при спробі відновити структуру S-блоку[4]. Варто зауважити, що дозволяється використовувати частину алгоритму, або ж використовувати його у поєднанні з іншими методами зворотньої розробки. Алгоритм не обов'язоково у всіх випадках видасть бажаний результат, проте точно наблизить до мети.

Алгоритм 2.1 (Декомпозиція невідомої структури S-блока)**.**

Вхід: S-блок

Вихід: прихована алгебраїчна структура

1: Створити візуалізацію LAT та DDT для S у стилі PPR.
2: Перевірити чи лінійні та диференціальні властивості S є сумісними із випадковими функціями та перестановками.
3: Знайти підпис $\sigma(S)$
4: Якщо підпис парний, тоді:

1) Спробувати атаку на SASAS.

2) Спробувати відокремити S від мережі Фейстеля за допомогою виключного АБО.

3) Якщо хоча би одне відокремлення запрацювало, то запустити алгоритм відновлення структури мережі Фейстеля від трійки змінних

(S, R, $\oplus$) для відповідного значення R.

5: Запустити алгоритм відновлення структури мережі Фейстеля від трійки змінних (S, R, $\oplus$) для $R \in [2,5]$, де $\oplus$ – це додавання за модулем.

6: Запустити алгоритм злому арифметики від S.

У алгоритмі 2.1 на певних кроцах використовуються алгоритми відновлення структури мережі Фейстеля та злому арифметики. Описи даних об'єктів наведено нижче.

Алгоритм 2.2 (Відновлення структури мережі Фейстеля)**.**

1: Нехай $f : \{0,1\}^n \longrightarrow \{0,1\}^n$, деяка невідома функція. Для кожного вихідного біта під порядковим номером i на основі входу x асоціюється унікальне значення z_i^x. Тоді таблиця істинності при $n = 3$ має вигляд як Таблиця 2.1, де бінарні змінні $y_i, i \in [0, n-1]$ – це вихід

Таблиця 2.1

$\mathbf{x_2}$	$\mathbf{x_1}$	$\mathbf{x_0}$	$\mathbf{y_2}$	$\mathbf{y_1}$	$\mathbf{y_0}$
0	0	0	z_2^0	z_1^0	z_0^0
0	0	1	z_2^1	z_1^1	z_0^1
...	...	...	...	...	...
1	1	1	z_2^7	z_1^7	z_0^7

функції f для заданих вхідних змінних x_i. Далі, у вигляді $bit_i(b)$ позначається і-ий біт двійкового представлення будь-якого цілого числа $b < 2^n$, при чому $b = \sum_{i=0}^{n} bit_i(n) * 2^{n-i}$. Також a^1 позначає змінну як саму себе, а a^0 як її заперечення. Процедура використовує КНФ на базі наступної імплікації: якщо $\{x_i\}_{i<n}$ відповідає бінарному представленню числа $x < 2^n$ і $\{y_i\}_{i<n}$ у бінарному представленні $y = f(x)$, то $y_i \oplus z_i^x = 0$ для всіх $i < n$. Ідея може бути представлена у наступному вигляді: $(\wedge_{i<n} x_i^{bit_i(x)}) \Rightarrow (y_i \oplus z_i^x = 0)$. І, як результат:

$$((\vee_{i<n} x_i^{1-bit_i(x)}) \vee y_i^1 \vee z_i^0) \wedge ((\vee_{i<n} x_i^{1-bit_i(x)}) \vee y_i^0 \vee z_i^1)$$

Якщо конкантенувати КНФ згенеровану таким чином для усіх значень $x < 2^n$, то буде отримано "$CNF(f, \{x_i\}, \{y_i\})$" із $(2 * n * 2^n)$ реченнями по $(n * 2^n + 2 * n)$ змінних.

2: *Генерація повної КНФ та її рішення.* Використовуючи "$CNF(f, \{x_i\}, \{y_i\})$" із $(2n * 2^n)$ та LUT даного S-box $S : \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ можна відновити функції Фейстеля, що використовувалися для генерації S, якщо вони були згенеровані таким чином або заперечити це, якщо не були. Щоб продовжити необхідно ввести певні нові змінні $\{x_i^r\}_{i<2n}$, $\{y_i^r\}_{i<2n}$ та $\{c_i^r\}_{i<2n}$, де R – це кількість раундів, і визначити, що $\oplus$ – це додавання за модулем. Візуалізація процесу наведена на Рисунку 2.1. Головна ідея полягає в

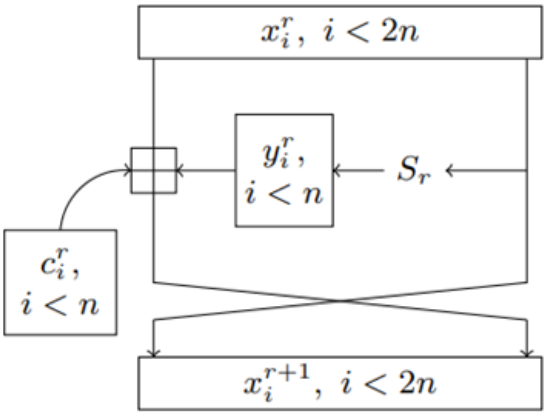


Рисунок 2.1

тому, щоб побудувати КНФ, виходячи з того факту, що $S(p) = c$ для кожної вхідної/вихідної пари (p, c) окремо, сконкатенувати ці КНФ і розв'язати отримане рівняння. До кожної функції Фейстеля асоціюється набір із $(n * 2^n)$ змінних, які описані вище. Це використовується при кодуванні половини поточного стану на $(r + 1)$ раунді мережа Фейстеля проходить через відповідну функцію Фейстеля. Єдина складність, що залишилася – це комбінація лівої

гілки та виходу із функції Фейстеля:

$$x_i^r = c_i^r \oplus x_{i+n}^r \oplus y_i^r$$

$$c_{i+n}^r = (c_i^r \wedge x_{i+n}^r) \vee (y_i^r \wedge x_{i+n}^r) \vee (x_{i+n}^r \wedge y_i^r)$$

Алгоритм 2.3 (Алгоритм ‘злому арифметики’)**.**

Це оптимізований пошук по дереву, здатний відновлювати прості операції, використані для створення S-блоку, побудованого як довільна послідовність базових операцій. Він був створений із наступних міркувань. Нехай $s = \phi_r \circ \phi_{r-1} \circ ... \circ \phi_1$, де ϕ_i – це одна із наступних алгебраїчних операцій: виключне АБО, додавання за модулем 2^n, множення за модулем 2^n, бітовий зсув. Тоді $s \circ \phi_1^{-1} = \phi_r \circ \phi_{r-1} \circ ... \circ \phi_2$ є менш складним і "ближчим до істинності ніж s сама по собі. Метою алгоритму є пройти усі прості операції одну за одною, щоб знайти ті, які мають найкраще значення критерія, тобто, потенційно підходять на роль тих, за допомогою кого будувався S-box. Щоб це працювало, необхідно зафіксувати концепт ’відстані до спражньості’ за допомогою метрики, яку можна ефективно реалізувати. А.Бірюков та Л.Перрін запропонували використовувати, базуючись на DDT, а не LAT в силу спрощення розрахунків, наступну метрику: $M(s) = \sum_{l>2} N_l * (l-2)^2$. Кандидати відбираються за значеннями $M(s \circ \phi_1^{-1})$ ближчими до $M(Id)$, де Id — функція ідентифікації.

2.2 Процес відновлення прихованої алгебраїчної структури на конкретному прикладі

У 2012 році російська структура стандартизації ГОСТ утвердила новий стандарт хеш-функції Стрибог. Є 2 типи функції в залежності від кількості біт входу: 512 та 256. Розглянемо перший варіант. Стрибог працює з 512-бітовими блоками вхідних даних, використовуючи

конструкцію Меркле–Дамгорда для обробки вхідних даних довільного розміру. Стискаюча функція даного хеша оперує у режимі Міягучі-Преніля і використовує 12-рандовий шифр типу AES із 512-бітним блоком та 512-бітним ключем. Було опубліковано уже декілька криптоаналізів проти даної хеш-функції. У 2014 році був проведений пошук другого прообразу, який увінчався успіхом за 2^{266} викликів стискаючої функції замість очікуваних 2^{512}. Інша атака націлена на модифікацію алгоритму, в якій змінюються лише константи: для деяких нових раундових констант можна було найти колізії хеш-функцій.

Нехай π — S-блок для хеш-функції Стрибога, який зображений на Таблиці 2.2. Варто зауважити, що даний криптопримітив використовується також і у російському стандартизованому блочному шифрі Кузьнечіка та кандидатом у CAESAR Стрибобом. Насправді, доволі мало відомо про π від самих творців – лише матриця, яка, власне, і є перетворенням. Причину такого вибору вони не розголошували, а лише повідомили, що вибір значень матриці відбувався випадковим чином до того моменту, поки вона не почала задовільняти необхідним критеріям. Проте, у 2015 році трійка науковців Алекс Бірюков, Лео Перрін і Олексій Удовенко довели, що даний S-блок не був згенерованим випадовим чином чи обраний із випадковий матриць, які задовольняли певним диференціальним, лінійним та алгебраїчним критеріям. Згідно їхнього дослідження вибір даної структури із можливих еквівалентний $2^{-82.7}$.

У загальному, процес зворотньої розробки дизайну S-box полягає в атаках проти конструкцій високого рівня, які потенційно могли би використовуватися при його створенні. Далі, порівнюючи результати зі статистично нормальними або за критерієм змістовності (який визначається індивідуально для кожного випадку), можна робити висновок про причетність того чи іншого алгоритму, операції, значення тощо, у проектуванні перетворення.

Перш за все, розглянемо результати, яких досягли дослідники при відновленні. Структуру декомпозиції S-блоку вони навели у схемі, що

Таблиця 2.2 – LUT-представлення π

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

зображена Рисунком 2.2.

Схему характеризують наступні об'єкти:

1) *множення у скінченному полі* " $\odot$ ": це множення у $GF(2^4)$ визначеному незвідним поліномом $(x^4 + x^3 + 1)$.

2) *4-бітні нелінійні функції*: зображені Таблицею 2.3.

Таблиця 2.3

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
I	0	1	C	8	6	F	4	E	3	D	B	A	2	9	7	5
υ_0	2	5	3	B	6	9	E	A	0	4	F	1	8	D	C	7
υ_1	7	6	C	9	0	F	8	1	4	5	B	E	D	2	3	A
ϕ	B	2	B	8	C	4	1	C	6	3	5	8	E	3	6	B
σ	C	D	0	4	8	B	A	E	3	9	5	2	F	1	6	7

3) *мультиплексор*: виходи υ_0 і υ_1 йдуть у мультиплексорі. Якщо значення на правій гілці в поточний момент рівне 0, то вибирається вихід

υ_0. В іншому випадку цей компонент повертає вихід υ_1.

4) *8-бітні лінійні перестановки*: лінійні функції відповідають множенням на наступних бінарних матрицях α та β.

$$\alpha = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \beta = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

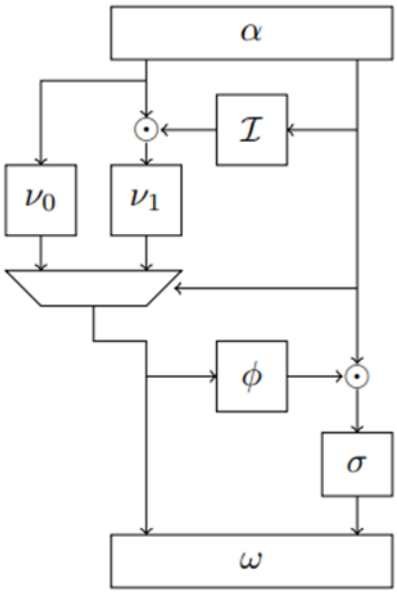


Рисунок 2.2 – Структура побудови S-блоку

Очевидною проблемою цієї структури є те, що множення на 0 не є оборотним. Дизайнери S-блоку підходили до кожного множення

по-різному. По-перше, у випадку множення на 0 використовується специфічний шлях даних. Під час другого множення проблему можна уникнути за допомогою функції, яка не має прообразу для 0.

У загальному, процес зворотньої розробки дизайну S-блоку полягає в атаках проти конструкцій високого рівня, які потенційно могли би використовуватися при його створенні. Далі, порівнюючи результати зі статистично нормальними або за критерієм змістовності (який визначається індивідуально для кожного випадку), можна робити висновок про причетність того чи іншого алгоритму, операції, значення тощо, у проектуванні перетворення.

Для того, щоб розпочати опис зворотньої розробки перетворення π хеш-функції Стрибога дослідники вводять наступні означення:

1. $\mathbb{F}_p$ — це скінченне поле числа p. Векторна двійкова функція —- це функція із $\mathbb{F}_p^n$ на $\mathbb{F}_p^m$. Двійкова перестановка довжини n —- перестановка на $\mathbb{F}_p^n$. Операнд '$\odot$' позначає множення у скінченному полі.

2. $\oplus$ – позначає виключне АБО – XOR; $\wedge$ – позначає логічне І – AND; скалярний добуток двох елементів $x = (x_{n-1}, ..., x_0)$ та $y = (y_{n-1}, ..., y_0)$, які належать $\mathbb{F}_2^n$, позначається як '$\cdot$', еквівалетний $x \cdot y = \oplus_{i=0}^{n-1}(x_i \wedge y_i)$

3. Таблиця Диференційного Розподілу (DDT) — це функція f, яка конвертує n біт у m у вигляді матриці $T : 2^n \times 2^m$,
де $T[\delta, \Delta] = \#\{x \in \mathbb{F}_2^n, f(x \oplus \delta) \oplus f(x) = \Delta\}$.

4. Таблиця Лінійної Апроксимаці (LAT) — це функція f, яка конвертує n біт у m у вигляді матриці $A : 2^n \times 2^m$,
де $A[a, b] = \#\{x \in \mathbb{F}_2^n, a * x = b * f(x)\} - 2^{n-1}$.
Коефіцієнти $T[\delta, \Delta]$ та $A[a, b]$ називаються кардиналом диференціала $\delta \rightsquigarrow \Delta$ та біасом апроксимації $a \rightsquigarrow b$ відповідно. З точки зору дизайнера краще тримати ці обидва коефіцієнти якомога малими. Наприклад, максимальне значення кардинала диференціала називається диференціальною рівноймовірністю і обирається малим у багатьох криптопримітивах, включаючи AES. Такий підхід зменшує індивідуальну ймовірність усіх диференціальних і лінійних слідів.

5. Афінна еквівалентність: дві двійкові векторні функції f та g є афінно-еквівалентними, якщо існує два афінних відображення μ та η, таких що: $f = \eta \circ g \circ \mu$.

Зворотнє проектування π хеш-функції Стрибога.

Спершу було знайдено підпис, який був парним. Це означає, що, можливо, справа йде із мережею Перестановок Заміни із простим бітом заміни, або із мережею Фейстеля. Проте, SASAS-атака та атака відновлення на базі SAT проти 3-, 4- та 5-раундової мережі Фейстеля закінчилася безуспішно. Також була відкинута гіпотеза про те, що π афінно-еквівалентна одночлену на базі того, що якщо перестановка f афінно-еквівалентна одночлену, то кожен рядок її DDT, що відповідає ненульовій вхідній різниці, містить однакові коефіцієнти (хоча зазвичай в іншому порядку).

Як вже згадувалося у Алгоритмі 2.1 для декомпозиції невідомої структури S перетворення S-блоку рекомендується створення візуальної репрезентації DDT та LAT за 'стилем Полока'. Для того, щоб могти краще розпізнати зображення була введена $\oplus$-текстура: $\oplus$-текстура для LAT A із S-блоку матриці $T^{\oplus}$ із коефіцієнтами $[i, j]$ — це $T^{\oplus}[i, j] = \#\{(x, y), |A[x \oplus i, y \oplus i]| = |A[x,y]|\}$.

Результат наведено на Рисунку 2.3.

На зображенні вирізняються вертикальні лінії, а також чорні точки у першій колонці. Індекси обох рядків, що містять чорні точки, та колонок, що містять лінії, однакові та відповідають двійковому векторному простору V, записаному у шістнадцятковій формі: $V = \{00, 1a, 20, 3a, 44, 5e, 64, 7e, 8a, 90, aa, b0, ce, d4, ee, f4\}$.

Щоб згрупувати стовпці ліворуч від зображення та темні крапки вгорі, було застосовано лінійне відображення L, щоб отримати нову таблицю A'_π, де $A'_\pi = A_\pi[L(i), L(j)]$. Результат наведено на Рисунку 2.4.

L визначається наступним чином: воно відображає $i \in \mathbb{F}_2^4$ у i-ий елемент простору V і доповнює його таким чином, щоб отримати лінійну перестановку на $\mathbb{F}_2^8$.

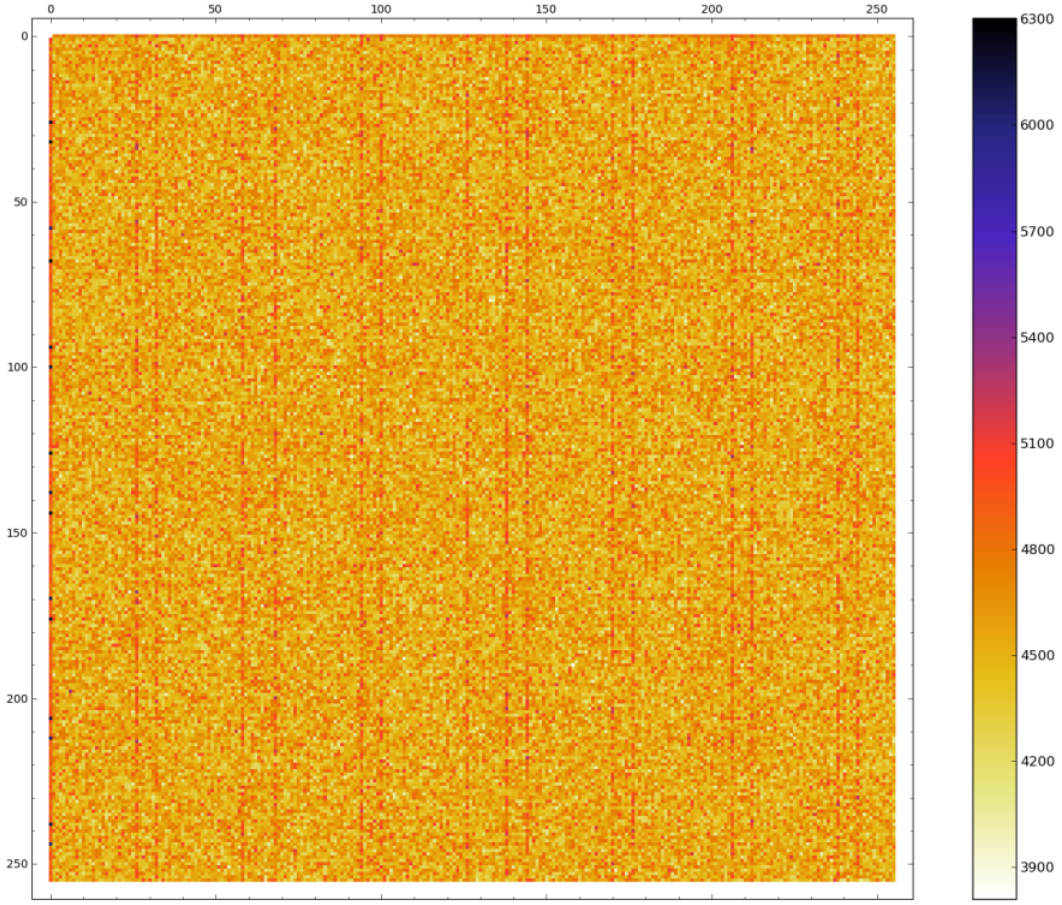


Рисунок 2.3 – $\oplus$-текстура для LAT A із S-блоку матриці $T^{\oplus}[i,j]$

Теорема 2.1. *Нехай f – це перестановка із n бітів на n та A – це її LAT. Тоді LAT оберненої перестановки f^{-1} – це A^t.*

Наслідок 2.1. *Із Теореми 2.1 випливає, що S-блок $L^t \circ \pi \circ (L^{-1})^t$ має L'_π у якості LAT. Відображення L^t міститься у лінійному раунді Фейстеля разом із 4-ьох бітними правих та лівих шматків. Тобто: $\pi' = L^* \circ \pi \circ L^*$, де L^* – це раунд Фейстеля у L^t, як візуалізовано на Рисунку 2.5.*

Перша декомпозиція

Результат, отриманий на попередньому кроці, має певні мультимножинні властивості. Нехай C визначає 4-ьох бітний сталий шматок, P – це 4-ьох бітний шматок, який визначається усіма можливими 16 значеннями, а '?' – визначає сет із невідомою струтурою, тоді Таблиця

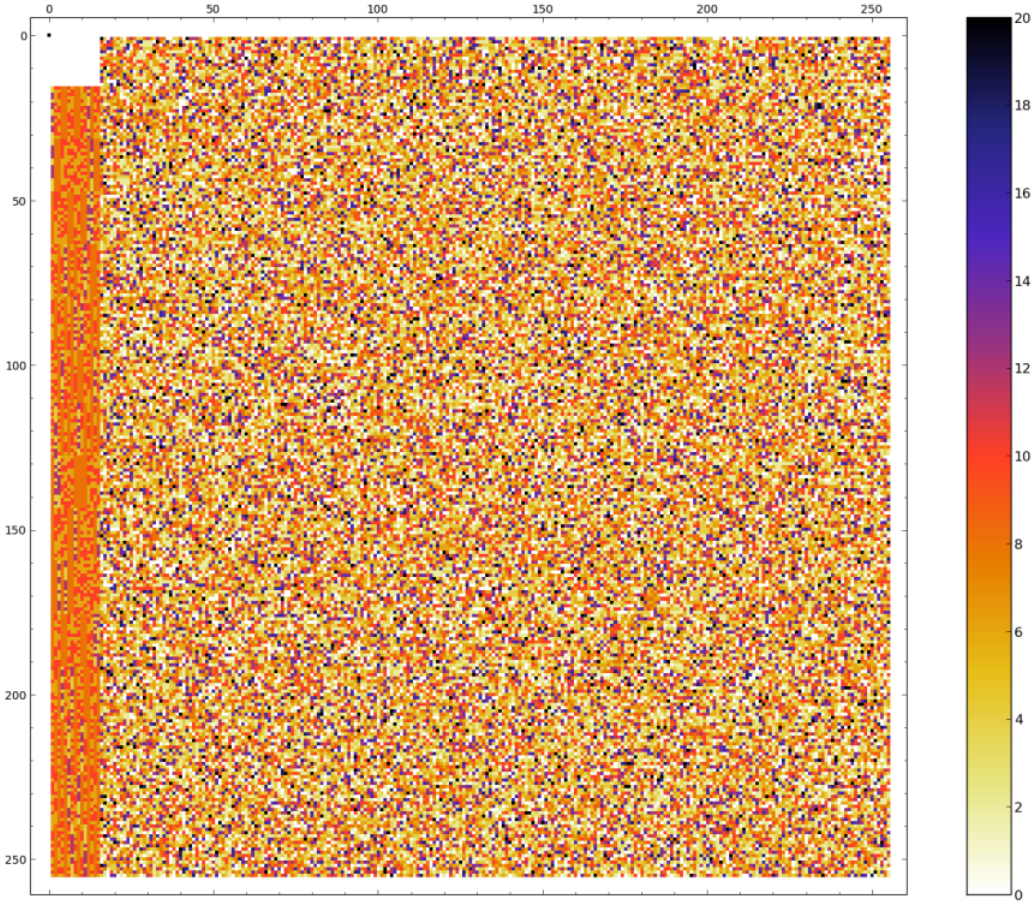


Рисунок 2.4 – Модифікація Рисунку 2.3

підсумовує мульмножинні властивості для π' та π'^{-1}.

Таблиця 2.4

	π'		π'^{-1}	
Вхід	(P,C)	(C,P)	(P,C)	(C,P)
Вихід	$(?,?)$	$(P,?)$	$(?,?)$	$(P,?)$

Логічно було би припустити, що це може бути 3-ьох раундова мережа Фейстеля, проте SAT-алгоритм виключив дану можливість.

Якщо оцінювати π'^{-1}, то можна звернути увагу, що мульмножинні властивості є сильнішими у даному випадку. Дійсно, для будь-якої константи l множина $S_l = \{\pi'^{-1}(l\|r), \forall r \in [0{,}15]\}$ – є майже векторним простором V_l, де права частина є лінійною функцією від лівої частина. Як було сказано раніше, ліва частина приймає усі можливі значення. Якщо

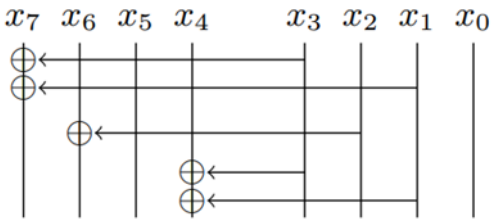


Рисунок 2.5 – Схема раунда Фейстеля у L^t

відкласти виходи форми $(?\|0)$, тоді π'^{-1} може розглядатися як $\pi'^{-1}(l\|r) = T_l(r)\|V_l(T_l(r))$. У даній декомпозиції T – це 4-ьох бітний блочний шифр із 4-ьох бітним ключем, де лівий вхід π'^{-1} поводить себе як ключ. З іншої сторони, V – це ключова лінійна функція: $\forall l : V_l$ – це лінійна функція, яка відображає 4 біті у 4 біти.

Далі доповнюється альтернативний вигляд заміною $V_l(0)$ на $\pi'^{-1}(l\|T_l^{-1}(0))$, що дозволяє знайти декомпозицію π'^{-1} вищого рівня.

Нарешті, визначається ключова функція $U_r(l) = V_l(r)$, де $\forall r : U_r$ – перестановка: $\pi'^{-1}(l\|r) = T_l(r)\|U_{T_l(r)}(l)$. На даному етапі уже є таблиці заповнень для T та U зі структурую, що наведена на Рисунку 2.6.

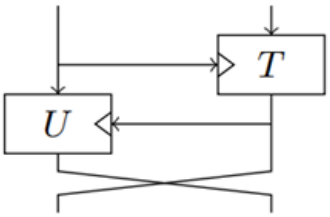


Рисунок 2.6

Зворотня розробка T

Нехай блочний шифр T' визначається наступним чином:
$T'_k(x) : x \mapsto T_k(x \oplus t_{in}(k) \oplus 0xC)$, для $t_{in}(k) = 0\|k_2\|k_3\|0, \forall k : T'_k(0) = 0$.

Більше того, T' такий, що усі рядки T'_k можуть бути виражені через лінійну комбінацію T'_6, T'_7, T'_8 та T'_9:

1. $T'_0 = T'_7 \oplus T'_9$
2. $T'_1 = T'_8 \oplus T'_9$

Таблиця 2.5 – T

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
T_0	E	F	2	5	7	B	8	1	3	C	D	A	0	9	4	6
T_1	2	9	A	4	E	6	7	B	1	8	3	D	0	C	F	5
T_2	E	F	2	5	7	B	8	1	3	C	D	A	0	9	4	6
T_3	5	D	4	2	6	7	B	8	C	1	9	F	0	3	A	E
T_4	5	E	6	7	4	3	F	A	0	1	D	2	8	B	C	9
T_5	9	D	F	A	C	6	8	1	0	5	B	3	2	4	E	7
T_6	3	9	D	F	1	E	B	8	0	2	7	C	4	A	5	6
T_7	5	E	6	7	4	3	F	A	0	1	D	2	8	B	C	9
T_8	7	B	8	5	9	D	C	3	2	E	A	F	6	1	0	4
T_9	D	F	A	C	E	6	2	5	1	3	B	7	9	4	0	8
T_A	E	6	7	4	C	3	8	1	A	2	D	9	5	B	0	F
T_B	4	2	5	D	B	8	6	7	9	F	C	1	A	E	0	3
T_C	2	5	A	4	3	9	D	8	C	F	0	7	B	1	6	E
T_D	E	6	2	5	D	F	A	C	9	4	0	8	1	3	B	7
T_E	9	D	C	3	7	B	8	5	6	1	0	4	2	E	A	F
T_F	8	1	7	B	2	5	E	F	4	6	0	9	D	A	3	C

3. $T'_2 = T'_7 \oplus T'_9$
4. $T'_3 = T'_6 \oplus T'_7$
5. $T'_4 = T'_7$
6. $T'_5 = T'_7 \oplus T'_8$
7. $T'_a = T'_6 \oplus T'_7 \oplus T'_8 \oplus T'_9$
8. $T'_b = T'_6 \oplus T'_7$
9. $T'_c = T'_6 \oplus T'_7 \oplus T'_8$
10. $T'_d = T'_9$
11. $T'_e = T'_8$
12. $T'_f = T'_7 \oplus T'_9$

Також можна сказати, що T'_6, T'_7, T'_8 та T'_9 є афінно-еквівалентними. Дійсно, лінійне відображення A, визначене $A : 1 \mapsto 4, 2 \mapsto 1, 4 \mapsto 8, 8 \mapsto a$ таке, що: $T'_7 = A \circ T'_6, T'_7 = A^2 \circ T'_6, T'_9 = A \circ T'_6$.

Якщо поміняти місцями 2 найменш значущі біти (ця операція у подальшому називатиметься *swapsignbits*) до і після прийняття A, то

стане очевидною структура ЛРЗ (див. Рисунок 2.7).

(а) **(б)**

Рисунок 2.7 – (а) визначення A; (б) A після операції *swapsignbits*.

Відтак, зроблено висновок, що поліномом виступає $x^4 + x^3 + 1$. А з цього вже виступає множення у скінченному полі та відображення $\hat{A} = swapsignbits \circ A \circ swapsignbits$, яке можна розглядати як множення X у $\mathbb{F}_{2^4} = \mathbb{F}_2[X]/(X^4 + X^3 + 1)$. Щоб підігнати свап до вихідної схеми, модифікується T'_6 і нижній лінійний шар:

$$A^i = (swapsignbits \circ \hat{A} \circ swapsignbits)^i = swapsignbits \circ \hat{A}^i \circ swapsignbits$$

Далі з'єднується один *swapsignbits* разом із T'_6, а інший *swapsignbits* проходить через виключне АБО поза межами T'. Нехай $t = swapsignbits \circ T'_6$, тоді $swapsignbits \circ T'_k$ – лінійна комбінація $X^i \odot t(x)$, де $i \in \{0,1,2,3\}$, а операція '$\odot$' – це множення у спеціально визначеному полі. Тоді T може знайтися наступним шляхом:$T_k(x) = swapsignbits(f(k) \oplus t(x \oplus t_{in}(k) \oplus 0xC))$, де f фіксує лінійні співвідношення із рівнянь 1.-12. в яких $T'_i, i \in [0x0, 0xF]$ виражається через T'_6, T'_7, T'_8 та T'_9. Перетворення f, і t наведено в Таблиці 2.6, а схема, що представляє структуру T, наведено на Рисунку 2.8.

Зворотня розробка U

Оскільки $U_k(x) = V_x(k)$ і V_x – це лінійна функція, коли $x \neq 0$, то:

Таблиця 2.6

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
f	A	C	A	3	2	6	1	2	4	8	F	3	7	8	4	A
t	2	D	B	8	3	A	E	F	4	9	6	5	0	1	7	C

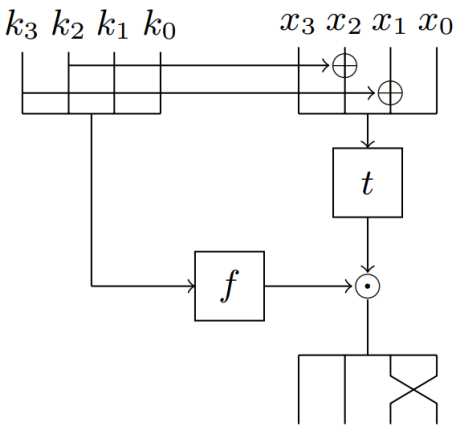


Рисунок 2.8 – Міні-блочний шифр T

$U_k(x) = (k_3 \times U_8(x)) \oplus (k_2 \times U_4(x)) \oplus (k_1 \times U_2(x)) \oplus (k_0 \times U_1(x))$, де $k = \sum_{i \leqslant 3} k^i * 2^i, k \neq 0$.

Більше того, перестановки U_2, U_4, U_8 походять від U_1 із використанням деякої афінної функції B_k : $U_k = B_k \circ U_1$. Дані B_k наведено у Таблиці 2.8.

Якщо позначити $B(x) = B_4(x) \oplus 1$, тоді $B_2(x) = B^{-1}(x) \oplus 5$ і $B_8(x) = B^2(x) \oplus 5$. Тому можна визначити лінійну функцію u_{out} таку, що:

1) $U_1(x) = B^0 \circ U_1(x) \oplus u_{out}(1)$
2) $U_2(x) = B^{-1} \circ U_1(x) \oplus u_{out}(2)$
3) $U_4(x) = B^1 \circ U_1(x) \oplus u_{out}(4)$
4) $U_8(x) = B^2 \circ U_1(x) \oplus u_{out}(8)$

Нехай M_2 – матриця представлення множення на X у скінченному полі, в якому була відновлена архітектура T, тобто у $\mathbb{F}_{2^4} = \mathbb{F}_2[X]/(X^4 + X^3 + 1)$. Було створено лінійне відображення u_f таке, що: $1 \mapsto 5, 2 \mapsto 2, 4 \mapsto 6, 8 \mapsto 8, B = u_f \circ M_2 \circ u_f^{-1}$:

1) $U_1(x) = u_f \circ M_2^0 \circ u_f^{-1} \circ U_1(x) \oplus u_{out}(1)$
2) $U_2(x) = u_f \circ M_2^{-1} \circ u_f^{-1} \circ U_1(x) \oplus u_{out}(2)$
3) $U_4(x) = u_f \circ M_2^1 \circ u_f^{-1} \circ U_1(x) \oplus u_{out}(4)$
4) $U_8(x) = u_f \circ M_2^2 \circ u_f^{-1} \circ U_1(x) \oplus u_{out}(8)$

Якщо ми змінити місцями два найменші значущі біти k, то експоненти матриці M_2 підуть у порядку зростання: $(-1, 0, 1, 2)$. Нехай

Таблиця 2.7 – U

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
U_0	8	F	0	2	D	5	6	9	E	3	1	7	C	B	4	A
U_1	8	C	7	3	D	F	2	0	E	4	1	B	6	5	9	A
U_2	3	4	E	9	D	8	0	5	1	2	C	F	7	B	A	6
U_3	B	8	9	A	0	7	2	5	F	6	D	4	1	E	3	C
U_4	C	2	5	B	E	8	7	1	4	F	D	6	9	3	0	A
U_5	4	E	2	8	3	7	5	1	A	B	C	D	F	6	9	0
U_6	F	6	B	2	3	0	7	4	5	D	1	9	E	8	A	C
U_7	7	A	C	1	E	F	5	4	B	9	0	2	8	D	3	6
U_8	A	F	B	E	C	4	D	5	7	0	6	1	8	3	9	2
U_9	2	3	C	D	1	B	F	5	9	4	7	A	E	6	0	8
U_A	9	B	5	7	1	C	D	0	6	2	A	E	F	8	3	4
U_B	1	7	2	4	C	3	F	0	8	6	B	5	9	D	A	E
U_C	6	D	E	5	2	C	A	4	3	F	B	7	1	0	9	8
U_D	E	1	9	6	F	3	8	4	D	B	A	C	7	5	0	2
U_E	5	9	0	C	F	4	A	1	2	D	7	8	6	B	3	E
U_F	D	5	7	F	2	B	8	1	C	9	6	3	0	E	A	4

Таблиця 2.8

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
$\mathbf{B_2}$	5	c	0	9	2	b	7	e	3	a	6	f	4	d	1	8
$\mathbf{B_4}$	1	d	7	b	f	3	9	5	c	0	a	6	2	e	4	8
$\mathbf{B_8}$	5	6	d	e	0	3	8	b	a	9	2	1	f	c	7	4

$u_1 = M_2^{-1} \circ u_f^{-1} \circ U_1(x)$. Оскільки M_2 – множення на X у скінченному полі, можна записати для $U_k(x)$:

$$U_k(x) = u_f(u_1(x) \odot swapsignbits(k)) \oplus u_{out}(k)$$

.

Результат декомпозиції U наведено на Рисунку 2.9. Тут використовується 4-ьох бітні перестановки u_0 та u_1, визначені у Таблиці . Не було знайдено залежності між u_1 та $u_0 = u_f^{-1} \circ U_0(x)$, тому було

зроблено висновок про розгалуження U:

$$U_k(x) = \begin{cases} U_k(x) = u_f(u_1(x) \odot swapsignbits(k)) \oplus u_{out}(k), & \text{якщо } k \neq 0 \\ u_f(u_0(x)), & \text{якщо } k = 0 \end{cases}$$

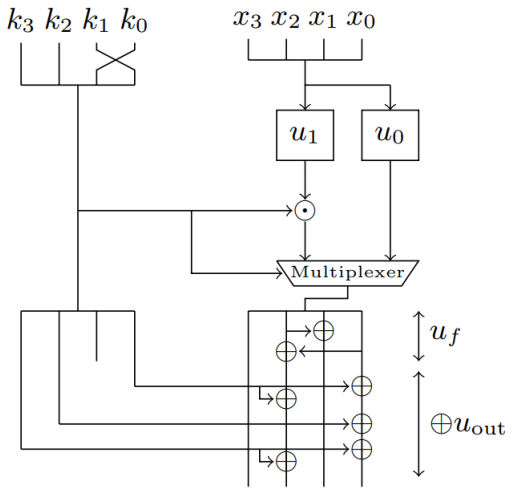


Рисунок 2.9 – Міні-блочний шифр U

Структура π

У двох попередніх секціях відбулося відновлення двох блочних шифрів T та U, які можуть бути використані для побудови π'^{-1}-обернення до $L^* \circ L^*$. Ці шифри базуються на 4-ьох бітних функціях f, t, u_0, u_1 множенні у скінченному полі та «трюком», щоб обійти необоротність множення на 0 і простих лінійних функцій. На даному етапі уже можна відновлювати π.

Спершу пов'язуються лінійні функції та L^* із α та ω, двома лінійними перестановками застосовними до початку та кінця виразу обчислення відповідно:

Таблиця 2.9 – Перестановки u_0 та u_0

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
$\mathbf{u_0}$	8	b	0	2	9	1	4	f	c	5	7	3	e	d	6	a
$\mathbf{u_1}$	4	7	d	e	8	9	1	0	6	3	f	a	2	c	b	5

• α: нехай L^* – це зміна місцями правої та лівої частини входу, що преставлено у декомпозиції π'^{-1}. Далі, потрібно звернути увагу на те, що ключ для U та ключ для T потребують *swapsignbits*. Потім приймається u_{out} та u_f

• ω: це композиція від співставлення t_{in} та L^*.

Таким чином отримуються матричне представлення для α та ω, які були описані на початку Підрозділу 2.2.

Для того, щоб обернути U визначається $v_0 = u_0^{-1}$ та $v_1 = u_1^{-1}$. Якщо $l = 0$, то вихід обернення U – це $v_0(r)$, в іншому випадку це $v_1(r \odot I(l))$, де $I : x \mapsto x^{14}$ – це множення на зворотній у полі $\mathbb{F}_2^4$. Щоби обернути T визначається $\sigma = e^{-1}$ та $\phi = I \odot f$ і знаходиться $\sigma(\phi(l) \odot r)$.

Тепер отримано усі результати описані у 2.2.

Висновки до розділу 2

Даний розділ описує зворотнє проєктування дизайну прихованої алгебраїчної структури перетворення S-блоку. Розглянуто загальні методи та підходи, які покращать ефективність даної процедури та наведено алгоритми, від яких варто відштовхуватися при дослідженні S-блоку у загальному випадку. У роботі описано процес відновлення прихованої архітектури перетворення π на прикладі російський стандартизованих криптосистем. Зроблено висновок про те, що π вибрано не випадковим чином із усієї можливої множини перестановок чи із множини, яка би задовільняла певним критеріям, а побудовано свідомо згідно певних правил. Також визначено певні переваги та недоліки даної структури, наприклад, використання множення у скінченному полі, що не є характерним при побудові даного криптопримітиву.

3 ПЕРЕВІРКА КРИПТОГРАФІЧНИХ ВЛАСТИВОСТЕЙ S-БЛОКІВ СТАТИСТИЧНИМИ МЕТОДАМИ

У Розділі 2 наведено криптоаналіз S-блоку, який використовується у деяких справжніх криптосистемах. Він базувався на знанні самого перетворення (без відомостей про причину вибору цих конкретних значень) та властивостей потенційних, на думку дослідників, об'єктів та методів, які використовувалися для його створення. Саме тому цей розділ присвячено практичній спробі знайти взаємозв'зок між преставленням S-блоку та його специфічним методом побудови, використовуючи певний статистичний метод.

3.1 Постановка задачі

Як уже згадувалося у Розділі 1 для побуви S-блоку зазвичай використовуються різні існуючі криптографічні перетворення, їх модифікації та комбінації, щоби досягти максимальної незалежності між вхідними та вихідними значеннями. Серед них – мережа Фейстеля, схема Місті та R-схема. Прийнято рішення згенерувати достатньо велику вибірку випадково-згенерованих S-блоків, які, проте, були побудовані за допомогою вищезгаданих структур. Згідно [19] значення у DDT для відповідних типів S-блоків потенційно мало би підпорядковуватися розподілу Пуассона із параметром $\lambda = 1$ або $\lambda = 0.5$ в залежності від типу побудови перетворення та способу обчислення його DDT. Тому задача полягає у тому, щоб виявити чи справді S-блоки, які побудовані цими конкретними методами, підпорядковуються даному розподілу із зазначеними параметрами.

Варто зауважити, що використовувалися класичні варіанти мереж Фейстеля, схем Місті та R-схем, проте, для кожної із двома варіантами

внутрішньо-раундових операцій, а саме: виключне АБО та додавання за модулем 2^n, де n – це довжина двійкового вектора, що подається на вхід операції. Таким чином, отримано 6 варіантів побудов S-блоків.

Аналогічне зауваження стосується і DDT: використовувався як і класичний метод побудови, який виражається формулою $DDT_{\oplus,\oplus} : outputDiff = SBox(inputDiff) \oplus SBox(inputDiff \oplus variable)$, де $\oplus$ – операція виключного АБО, так і дві варіації операцій визначених у формулі:

$$DDT_{+,\oplus} : outputDiff = SBox(inputDiff) + SBox(inputDiff \oplus variable),$$

$$DDT_{+,+} : outputDiff = SBox(inputDiff) + SBox(inputDiff + variable),$$

де операція $+$ визначає додавання за модулем 2^n, де n – це довжина двійкового вектора, що подається на вхід перетворення.

Таким чином, маємо 3 варіанти знаходження DDT для кожного із 6 типів побудови S-блоків – загалом 18 класів отриманих таблиць. Для того, щоб не зберігати усі отримані таблиці, прийнято рішення будувати хеш-мапи – таблиці, які своїми розмірами відповідають DDT, проте, комірки всередині таких таблиць містять інформацію про усі унікальні значення, які зустрічалися у відповідній комірці у вибірці DDT, та їх кількість серед вибірки. Очевидно, що сума кількостей усіх унікальних значень кожної окремої комірки рівна величині вибірки.

Отже, зібравши достатньо велику вибірку S-блоків та побудувавши до кожного із них DDT прийшов час перевірити гіпотези про розподіл Пуассона. Вирішено скористатися критерієм узгодженості Пірсона (методом χ^2) для перевірки простої гіпотези H_0 – вибірка підпорядковується розподілу Пуассона із параметром $\lambda = 0.5$, проти H_1 – вибірка не підпорядковується даному розподілу.

Зауваження. Гіпотеза H_0 дещо відрізнятиметься у значенні параметра розподілу Пуассона для різних типів DDT. А саме:
нехай α – вхідна різниця DDT (позначає рядки у таблиці), а α – вихідна

різниця (позначає стовпчики), тоді:

- $DDT_{\oplus,\oplus}$ – тут так і залишиться $\lambda = 0.5$
- $DDT_{(+,\oplus),(+,+)}$ – тут коефіцієнт визначається наступним чином:

$$\lambda = \begin{cases} 0.5, & \text{якщо } ord(\alpha) = ord(\beta) = 2 \\ 1, & \text{в іншому випадку} \end{cases},$$

де $ord(x)$ – порядок ел. x у групі $\mathbb{G}_{2^n}$, n – к-сть біт, якими оперує S-блок.

$ord(x) = 2$ можна розглядати як піднесення до квадрату $x^2 = 0$ у відповідній групі $x \in \mathbb{G}_{2^n}$. Наступні твердження випливають із загальновідомих результатів абстрактної алгебри.

Твердження 3.1. *Нехай $\mathbb{G}_{2^n}$ – це група із операцією побітового додавання (виключного АБО) $\oplus$ визначеною на ній, $x \in \mathbb{G}_{2^n}$. Тоді:*
$ord_{\oplus}(x) = 2 \iff x \neq 0$

Доведення. Нехай $\mathbb{G}_{2^n}$ – це група із операцією побітового додавання (виключного АБО) $\oplus$ визначеною на ній, $x \in \mathbb{G}_{2^n}$. Тоді:
x у двійковому записі має вигляд $x = (x_{n-1}, ..., x_1, x_0)$, де $x_i \in \{0,1\}$. Тоді піднесенням до квадрату через операцію побітового додавання буде наступний вираз:
$x^2 = (x_{n-1} \oplus x_{n-1}, ..., x_1 \oplus x_1, x_0 \oplus x_0) = (0, ..., 0, 0) = 0.$ □

Твердження 3.2. *Нехай $\mathbb{G}_{2^n}$ – це група із операцією додавання за модулем 2^n визначеною на ній, $x \in \mathbb{G}_{2^n}$. Тоді:*
$ord_{+}(x) = 2 \iff x = 2^{n-1}$

Доведення. Нехай $\mathbb{G}_{2^n}$ – це група із операцією додавання за модулем 2^n визначеною на ній, $x = 2^{n-1}, x \in \mathbb{G}_{2^n}$. Тоді піднесенням до квадрату буде наступний вираз:
$x^2 = x + x = 2^{n-1} + 2^{n-1} = 2 * (2^{n-1}) = 2^n = 0(mod 2^n).$ □

Щодо коваріації параметрів λ розподілу Пуассона у відповідних типах DDT: при $DDT_{+,+}$ лише одна комірка повинна була би мати

параметр $\lambda = 0.5$ – та, що на перетині 2^{n-1} рядка та 2^{n-1} стовпчика, а при $DDT_{+,\oplus}$ – увесь рядок під індексом 2^{n-1}.

Розподіл Пуассона

Це один із найбільш відомих та поширених ймовірнісних розподілів випадкових величин[20]. Дискретна випадкова величина ξ має розподіл Пуассона з параметром $\lambda > 0$, якщо: $Probability\{\xi = n\} = \dfrac{\lambda^n * e^{-\lambda}}{n!}$.

Критерій узгодженості Пірсона – метод χ^2

Це один з найвідоміших критеріїв, що використовується для перевірки гіпотези про закон розподілу[20]. Щоб скористатися цим критерієм, вибіркові дані попередньо групують, тобто переходять до частотного представлення вихідних даних. Область значень вибірки ділять на певну кількість інтервалів, після чого будують функцію відхилення по різницях теоретичних імовірностей потрапляння в інтервали групування й емпіричних частот. Визначається це наступною математичною формулою:

$$p = \sum_{i=0}^{k} \frac{(v_i - np_i)^2}{np_i}$$

де k – кількість інтервалів, n – величина вибірки, p_i – ймовірність значення попасти в інтервал i, v_i – спостережувана частота для інтервалу i.

У даному методі також вводиться поняття критичної межі $\chi^2_{(1-\alpha),(N-1)} = (N-1) + Z_{1-\alpha} * (2N-2)^{1/2}$ у залежності від рівня значущості α, де Z – квантиль стандартного нормального розподілу, а N – кількість ступенів свободи.

Таким чином, критерій згоди χ^2 має наступний вигляд:
нехай задані рівень значущості α, вибірка об'ємом n, вектор кількостей реальних входжень в кожен інтервал вибірки та вектор теоретичних ймовірностей потрапити в кожен інтервал вибірки, то якщо значення статистики p менше за значення критичної межі $\chi^2_{(1-\alpha)}$ за заданих умов, то гіпотеза H_0 приймається, в іншому випадку приймається гіпотеза H_1.

Критерієм узгодженості Пірсона було вирішено застосовувати до

кожної конкретної комірки кожної із 18 хеш-мап різних класів на рівні значущості $\alpha = 0{,}01$. У якості інтервалів вибірки приймалися усі можливі значення від 0 до $(2^n - 1)$ включно, де n – це довжина двійкового запису входу та виходу перетворення S-блоку, спостережувані кількості входжень в кожен інтервал $v_i, i \in [0, 2^n - 1]$ – кількості відповідних значень i серед вибірки, а теоретичні ймовірності попасти у заданий інтервал $p_i, i \in [0, 2^n - 1]$ визначалися за допомогою формули Пуассона від i.

Зауваження. Важливо наголосити на тому, що у критерії Пірсона не було змоги самостійно обирати інтервали, на які розбивається числова вісь. У якості інтервалів автоматично постають 2^n конкретних цілих чисел в інтервалі від 0 до $(2^n - 1)$ включно.

3.2 Програмна реалізація та практичні результати

Програмна реалізація виконана за допомогою високорівневої мови програмування Python3 версії *3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)]* у середовищі розробки *Jupyter Notebook* від Anaconda. Код програми у вигляді окремих функцій наведений у Додатку А 3.2.

Для кожного із 6 класів побудови S-блоку (Фейстель$_\oplus$, Фейстель$_+$, Місті$_\oplus$, Місті$_+$, R$_\oplus$, R$_+$) згенеровано 100000 псевдо-випадкових екземлярів на базі Алгоритму 3.1. Усі S-блоки приймають на вхід 8-бітні вектори і повертають вектори такої ж самої довжини. Випадковість перестановки на кроці 3 визначається методом *shuffle* від бібліотеки *random*. Цей метод базується на методі випадкового перетасування Фішера-Йетса (з англ – Fisher–Yates shuffle), який у свою чергу використовує генератор випадкових чисел Мерсена Твістера (з англ – Mersenne Twister) для вибору випадкових індексів у новій послідовності.

Алгоритм 3.1 (Побудова вибірки S-блоку)**.**

Вхід: -

Вихід: вибірка зі 100000 екземлярів псевдо-випадкових S-блоків.

1: Створити пустий масив.
2: **for** $i \in [1, 100000]$ **do**
3: Створити пустий S-блок розмірами $n \times n$.
4: **for** $i \in [1, 2^n]$ **do**
5: Створити 3 псевдо-випадкові перестановки усіх можливих 4-бітних векторів.
6: Створити 1 впорядкований за зростанням масив усіх можливих 8-бітних векторів.
7: Провести 3 раунди відповідної схеми побудови S-блоку на кожному з яких використовувати випадкову перестановку із кроку 3 в якості функції.
8: Додати отриманий результат до відповідної комірки S-блоку.
9: **end for**
10: Додати отриманий S-блок до масиву.
11: **end for**
12: Зберегти масив S-блоків.

Для кожного згенерованого S-блоку побудовано відповідну DDT у трьох варіантах – $(\oplus, \oplus), (+, \oplus), (+, +)$. Загалом, 1 800 000 таблиць для 18 класів. Кожна така таблиця має розмір 256×256.Очевидно, що зберігати у пам'яті такий об'єм інформації не є доцільно. Тому для кожного класу створено хеш-мапу у вигляді таблиці розміром 256×256, яка містить інформацію про усі унікальні значення, які зустрічалися у відповідній комірці у вибірці зі 100000 екземплярів DDT. Таким чином, маємо 18 таблиць.

Для кожної хеш-мапи, в залежності від її класу, перевірено гіпотезу про розподіл Пуассона із відповідним параметром базуючись на Алгоритмі 3.2.

Алгоритм 3.2 (Перевірка гіпотези про розподіл)**.**

Вхід: хеш-мапа вибірки DDT.

Вихід: таблиця із прийняттям/відхиленням гіпотези про розподіл.

1: Створити пустий таблицю розмірами як в хеш-мапи.
2: Зафіксувати клас хеш-мапи.
3: **for** $i \in [1, 2^n]$ **do**
4: **for** $j \in [1, 2^n]$ **do**
5: Зафіксувати комірку хеш мапи на перетині i-ого рядка та j-ого стовпчика
6: Запустити тест Пірсона для значень заданої комірки.
7: **if** гіпотеза приймається **then**
8: записати 1 у відповідній комірці у пустій таблиці
9: **else**
10: записати 0 у відповідній комірці у пустій таблиці
11: **end if**
12: **end for**
13: **end for**
14: Зберегти таблицю із прийняттям/відхиленням гіпотез.

Результати

Абсолютно в усіх випадках гіпотеза про розподіл Пуассона з відповідним параметром відхилилася. Очевидною проблемою, що виникала при знаходженні статистики χ^2 було те, що кількість більшості входжень унікальних спостережуваних значень була рівною 0, тобто вони просто не зустрічалися у вибірці. У більшості випадків, інтервали, які відповідали числам '40' і більше, взагалі ніколи не мали попадань. Найбільшу ж кількість входжень мало значення '0' для кожної комірки усіх класів хеш-мап. Орієнтовно '0' займав $30 - 80\%$ вибірки.

Проте, не можна було не звернути увагу на певну тенденцію для усіх 6 класів S-блоків DDT яких обчислювалося операціями $(+, +)$. Розглянемо комірку хеш-мапи, яка містить інформацію про S-блоки, побудовані за допомогою мережі Фейстеля через операцію виключного АБО, на перетині першого рядка та першого стовпчика. Серед спостережуваних значень i вибірки зустрічалися лише 16 чисел –

$[0, 1, 2, ..., 14, 16]$. Частота входжень строго спадає зі зростанням значення самого i:

0	1	2	3	4	5	6	7	8
46306	24910	14875	7309	3604	1633	724	328	155
9	10	11	12	13	14	15	16	
68	43	23	14	5	2	0	1	

Розглянемо тепер усю хеш-мапу для $DDT_{+,+}$ та мережі Фейстеля із операцією виключного АБО. На Рисунку 3.1 наведено кількості входжень кожного конкретного значення у вибірку із 100 000 екземплярів. Якщо ж обежетися першими 20 значеннями з усіх, що могли би зустрітися у хеш-мапі, тобто (0,1,..., 19), то отримаємо Рисунок 3.2. Кожна лінія позначає перебіг входжень окремої комірки цієї таблиці. Варто зауважити, що ситуація абсолютно аналогічна зі всіма 6 класами побудови S-box для яких будувався DDT за операціями $+, +$. Якщо розглядати класи для $DDT_{\oplus,\oplus}$, то строгого спадання кількості входжень не спостерігається, «0» зустрічається частіше та він у більшій кількості випадків розмежовує значення, що зустрічаєтсья у вибірці. Результати наведені на Рисунках 3.3 та 3.4. На Рисунках 3.5 та 3.6 зображено типовий розподіл диференціалців $DDT_{+,\oplus}$ для усіх класів фейстель-подібних схем.

Висновки до розділу 3

У даному розділі описана власна спроба криптоаналізу перетворення S-блоків, які генерувалися випадковим чином, але, відповідаючи певним критеріям. Для кожного із 6 класів побудови S-блоку (Фейстель$_\oplus$, Фейстель$_+$, Місті$_\oplus$, Місті$_+$, R$_\oplus$, R$_+$) згенеровано 100000 псевдо-випадкових екземлярів. Для кожного згенерованого S-блоку побудовано відповідну DDT у трьох варіантах – $(\oplus, \oplus), (+, \oplus), (+, +)$. Загалом, 1 800 000 таблиць для 18 класів. Для

кожної хеш-мапи, в залежності від її класу, перевірено гіпотезу про розподіл Пуассона із відповідним параметром. У всіх випадках гіпотеза про розподіл Пуассона з відповідним параметром відхилилася. Наведено по 2 графіки для кожного типу DDT, які візуалізують розподіли диференціалів фейстель-подібних S-блоків.

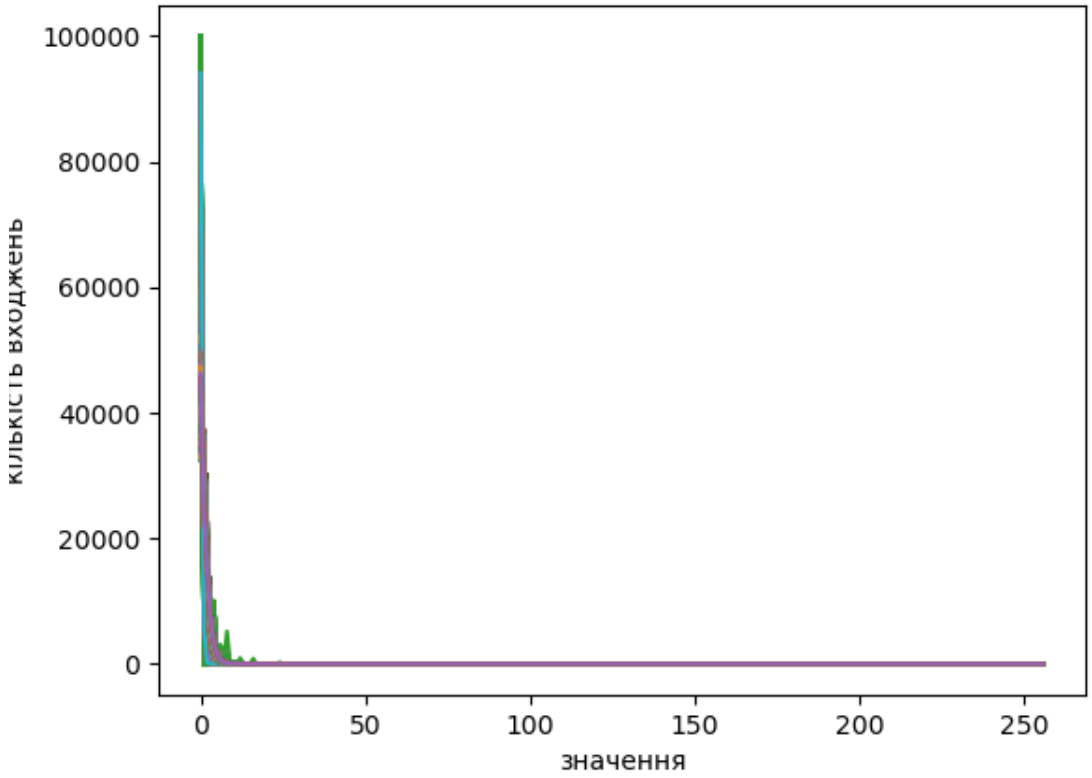


Рисунок 3.1 – Візуалізація $DDT_{+,+}$

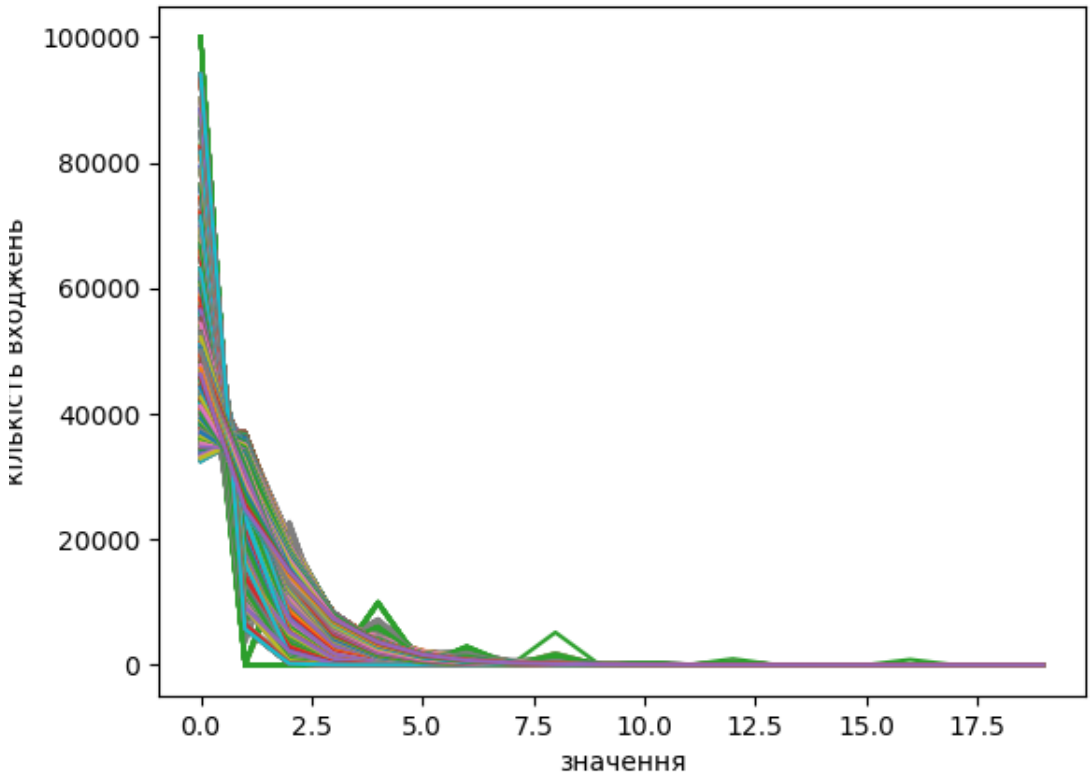


Рисунок 3.2 – Візуалізація $DDT_{+,+}$ (масштабована)

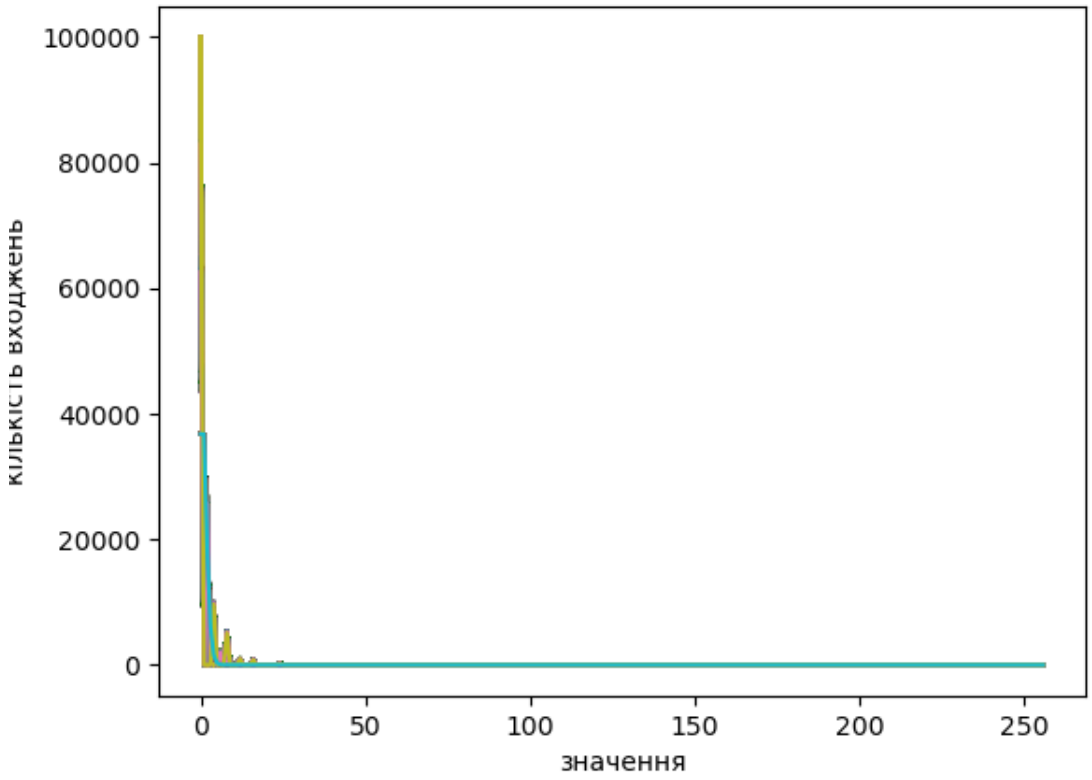


Рисунок 3.3 – $DDT_{\oplus,\oplus}$

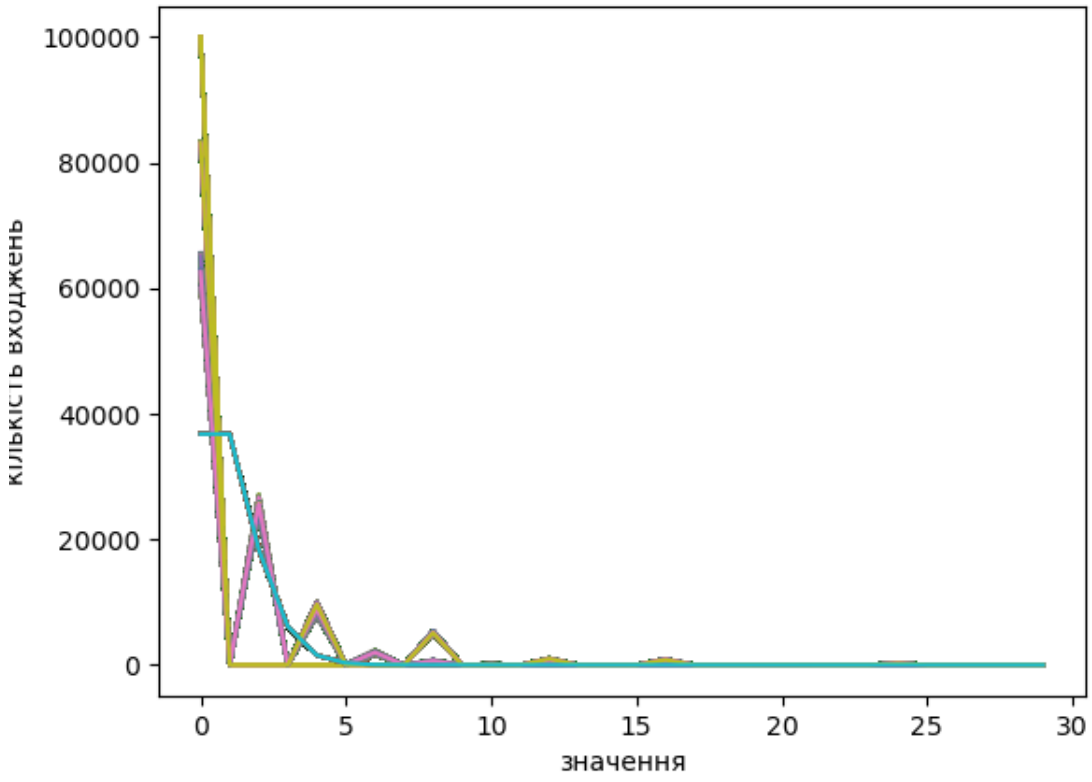


Рисунок 3.4 – $DDT_{\oplus,\oplus}$ (масштабована)

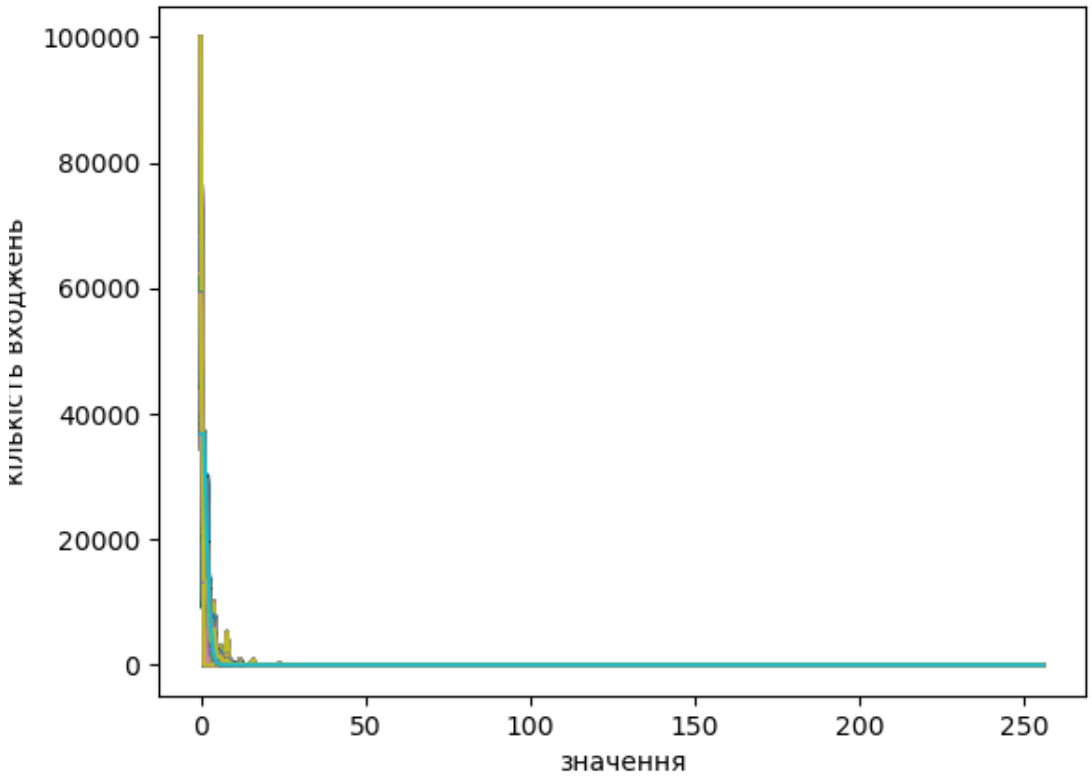


Рисунок 3.5 – $DDT_{+,\oplus}$

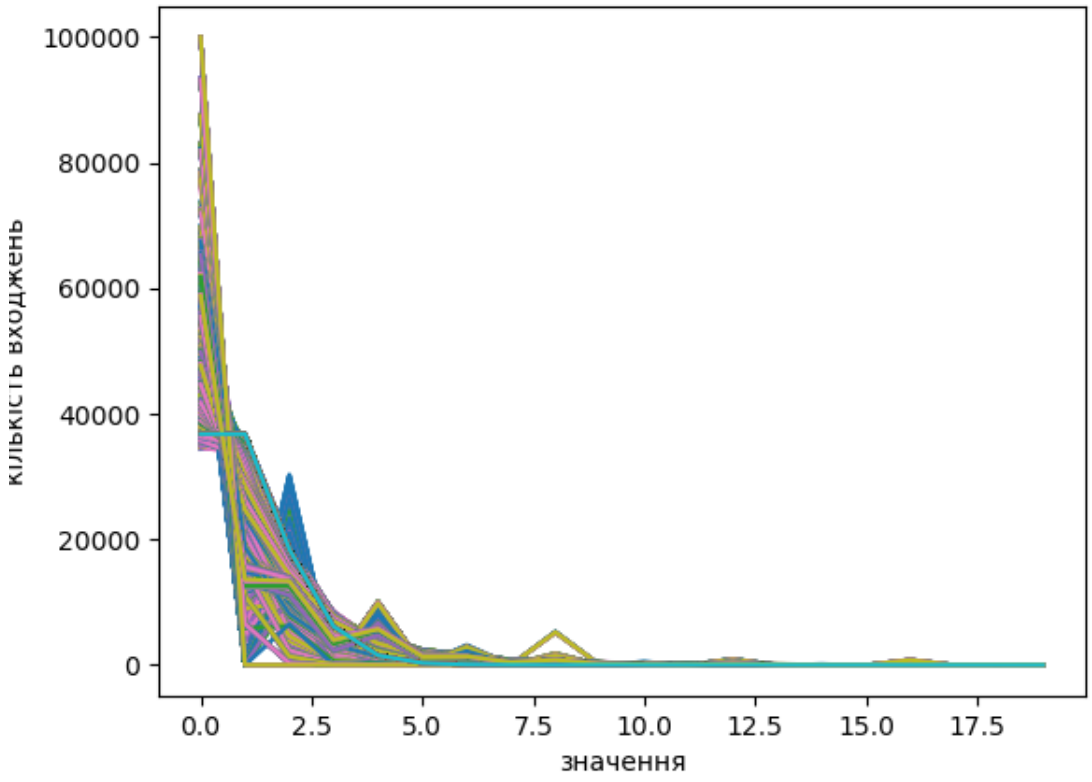


Рисунок 3.6 – $DDT_{+,\oplus}$ (масштабована)

ВИСНОВКИ

У даній роботі розглядалась задача виявлення та відновлення прихованої алгебраїчної структури у S-блоках. Розглядалися випадки, коли перевторення мали структуру схеми Фейстеля або її аналогів, побудованих за допомогою різних операцій. Проведено аналіз деяких методів та висунено припущення, що можна виявляти такі S-блоки розглядаючи розподіли ймовірностей диференціалів. Для цього була перевірена статистична гіпотеза щодо поведінки розподілів імовірностей диференціалів відносно операції побітового додавання, модульного додавання та їх суміщення, для S-блоків, які мають структуру схеми Фейстеля, схеми Місті чи R-схеми із різними операціями всередині. Виявлено, що розподіли для S-блоків заданих типів статистично відрізняються для розподілів просто випадкових S-блоків. Загалом було побудовано 6 класів S-блоків, кожен з яких має структуру схеми Фейстеля, Місті чи R- з операцією бітового додавання або бітового множення. Для кожного такого класу подубовано по 3 таблиці диференціалів із різними комібнаціями операцій при їх створенні. Визначено, що в усіх випадках розподіл диферениціалів відхиляється від тих, що справді згенеровані випадковим чином, тобто структурні залежності таки виявляються. За допомогою цих результатів можна перевірити чи має таблично заданий S-блок структуру фейстель-подібної мережі.

ПЕРЕЛІК ПОСИЛАНЬ

[1] C. Adams та J. Gilchrist. *The CAST-256 Encryption Algorithm.* Англ. 1999. URL: https://datatracker.ietf.org/doc/html/rfc2612.

[2] Anne Canteaut. *Differential cryptanalysis of Fesitel ciphers and differentially d-uniform mappings.* Англ. 1991. URL: https://www.researchgate.net/publication/2357350_Differential_cryptanalysis_of_Feistel_ciphers_and_differentially_delta-uniform_mappings.

[3] K. Aoki та ін. *Camellia: A 128-bit block cipher suitable for multiple platforms - design and analysis.* Англ. 2000. URL: https://info.isl.ntt.co.jp/crypt/camellia/dl/reference/sac_camellia.pdf.

[4] A. Biryukov та L. Perrin. *On Reverse-Engineering S-Boxes with Hidden Design Criteria or Structure.* Англ. 2015, с. 1—20. URL: https://eprint.iacr.org/2015/976.

[5] A. Biryukov, L. Perrin та A. Udovenko. *Reverse-Engineering the S-Box of Streebog, Kuznyechik and STRIBOBr1 (Full Version).* Англ. 2016, с. 1—34. URL: https://eprint.iacr.org/2016/071.

[6] A. Biryukov, L. Perrin та A. Udovenko. *The Secret Structure of the S-Box of Streebog, Kuznechik and Stribob.* Англ. 2015, с. 1—6. URL: https://eprint.iacr.org/2015/812.

[7] C. Burwick та ін. *SHS. Secure Hash Standard.* Англ. 1999.

[8] J. Chandrasekaran. *A Chaos Based Approach for Improving Non Linearity in the S-box Design of Symmetric Key Cryptosystems.* Англ. 2011, с. 516. URL: https://books.google.com.ua/books?id=pXOS4ZTUJLYC&pg=PA516&redir_esc=y#v=onepage&q&f=false.

[9] M. Dawson та S. Tavares. *An Expanded Set of S-box Design Criteria Based on Information Theory and its Relation to Differential-Like Attacks.* Англ. 1991. URL: https://www.researchgate.net/publication/225493748_An_Expanded_Set_of_S-box_Design_Criteria_Based_on_Information_Theory_and_its_Relation_to_Differential-Like_Attacks.

[10] Mitsuru Matsui. *The First Experimental Cryptanalysis of Data Encryption Standard.* Англ. 1991. URL: https://link.springer.com/chapter/10.1007/3-540-48658-5_1.

[11] K. Nyberg. *Perfect nonlinear S-boxes. Advances in Cryptology.* Англ. 1991. URL: https://link.springer.com/chapter/10.1007/3-540-46416-6_32.

[12] L. O'Connor. *Properties of Linear Approximation Tables.* Англ., с. 132—133. URL: https://link.springer.com/content/pdf/10.1007/3-540-60590-8_10.pdf.

[13] Orr Dunkelman та Senyang Huang. *Reconstructing an S-box from its Difference Distribution Table.* Англ. 2019, с. 194—196. URL: https://tosc.iacr.org/index.php/ToSC/article/view/8319.

[14] FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION. *SHS. Secure Hash Standard.* Англ. 2012. URL: https://csrc.nist.gov/publications/detail/fips/180/4/final.

[15] Federal Information Processing Standards Publications. *ADVANCED ENCRYPTION STANDARD (AES).* Англ. 2001, с. 15—16. URL: https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf#page=17&zoom=100,0,0.

[16] Sankhanil Dey та Ranjan Ghosh. *A Review of Existing 4-bit Crypto S-box cryptanalysis Techniques and Two New Techniques with 4-bit Boolean Functions for Cryptanalysis of 4-bit Crypto S-boxes.* Англ. 1997, с. 2—3. URL: https://eprint.iacr.org/2017/1161.pdf.

[17] National Institute of Standards та Technology. *Data Encryption Standard*. Англ. 1977. URL: https://csrc.nist.gov/CSRC/media/Publications/fips/46/archive/1977-01-15/documents/NBS.FIPS.46.pdf.

[18] Thierry Pierre Berger, Marine Minier та Gaël Thomas. *Extended Generalized Feistel Networks using Matrix Representation*. Англ. 2014, с. 1—2. URL: https://hal.science/hal-00913881.

[19] S.V. Yakovliev та V. Yu. Bakhtigozin. *Asymptotic Distributions for S-Box Heterogeneous Differential Probabilities*. Англ. 2019, с. 1—5. URL: http://tacs.ipt.kpi.ua/article/view/169029.

[20] М.В. КАРТАШОВ. *ІМОВІРНІСТЬ, ПРОЦЕСИ, СТАТИСТИКА*. 2008, с. 37, 411—412. URL: https://probability.knu.ua/userfiles/kmv/VPS_Pv.pdf.

ДОДАТОК А ТЕКСТИ ПРОГРАМ

Як уже зазначалося у Розділі 3 програмна реалізація виконувалася за допомогою високорівневої мови програмування Python версії *3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)]* у середовищі розробки *Jupyter Notebook* від Anaconda.

Для коректної роботи програмного коду завантажено та використано наступні бібліотеки Python:

```
from itertools import product
import random
import datetime
import numpy as np
import ast
import math
from scipy.stats import poisson, chi2, chisquare
```

А.1 Генерація випадкових S-блоків 6 класів

```
def feistel_xor(f1, f2, f3):
    x_list = list(product([0, 1], repeat=8))
    S_box = []
    for x in x_list:
        left_part = x[:4]
        right_part = x[4:]
        for f in [f1, f2, f3]:
            new_right = left_part
            binary_string = ''.join(str(bit) for bit in
  left_part)
            decimal_left_part = int(binary_string, 2)
            new_left = [a ^ b for a, b in zip(right_part, f[
```

```
    decimal_left_part])]
                left_part = new_left
                right_part = new_right
            S_box.append(int(''.join(str(bit) for bit in (left_part
     + right_part)), 2))
     return S_box

def feistel_plus(f1, f2, f3):
     x_list = list(product([0, 1], repeat=8))
     S_box = []
     for x in x_list:
         left_part = x[:4]
         right_part = x[4:]
         for f in [f1, f2, f3]:
             new_right = left_part
             decimal_left_part = int(''.join(str(bit) for bit in
     left_part), 2)
             a = int(''.join(str(bit) for bit in right_part), 2)
             b = int(''.join(str(bit) for bit in f[
    decimal_left_part]), 2)
             new_left = [int(bit) for bit in bin((a + b) % 16)
    [2:].zfill(4)]
             left_part = new_left
             right_part = new_right
         S_box.append(int(''.join(str(bit) for bit in (left_part
     + right_part)), 2))
     return S_box

def misty_xor(f1, f2, f3):
     x_list = list(product([0, 1], repeat=8))
     S_box = []
```

```
    for x in x_list:
        left_part = x[:4]
        right_part = x[4:]
        for f in [f1, f2, f3]:
            new_left = right_part
            decimal_left_part = int(''.join(str(bit) for bit in
    left_part), 2)
            new_right = [a ^ b for a, b in zip(right_part, f[
   decimal_left_part])]
            left_part = new_left
            right_part = new_right
        S_box.append(int(''.join(str(bit) for bit in (left_part
    + right_part)), 2))
    return S_box

def misty_plus(f1, f2, f3):
    x_list = list(product([0, 1], repeat=8))
    S_box = []
    for x in x_list:
        left_part = x[:4]
        right_part = x[4:]
        for f in [f1, f2, f3]:
            new_left = right_part
            decimal_left_part = int(''.join(str(bit) for bit in
    left_part), 2)
            a = int(''.join(str(bit) for bit in right_part), 2)
            b = int(''.join(str(bit) for bit in f[
   decimal_left_part]), 2)
            new_right = [int(bit) for bit in bin((a + b) % 16)
   [2:].zfill(4)]
            left_part = new_left
```

```
            right_part = new_right
        S_box.append(int(''.join(str(bit) for bit in (left_part
    + right_part)), 2))
    return S_box

def r_sheme_xor(f1, f2, f3):
    x_list = list(product([0, 1], repeat=8))
    S_box = []
    for x in x_list:
        left_part = x[:4]
        right_part = x[4:]
        for f in [f1, f2, f3]:
            decimal_left_part = int(''.join(str(bit) for bit in
    left_part), 2)
            new_right = list(f[decimal_left_part])
            new_left = [a ^ b for a, b in zip(right_part, f[
   decimal_left_part])]
            left_part = new_left
            right_part = new_right
        S_box.append(int(''.join(str(bit) for bit in (left_part
    + right_part)), 2))
    return S_box

def r_sheme_plus(f1, f2, f3):
    x_list = list(product([0, 1], repeat=8))
    S_box = []
    for x in x_list:
        left_part = x[:4]
        right_part = x[4:]
        for f in [f1, f2, f3]:
            decimal_left_part = int(''.join(str(bit) for bit in
```

```
    left_part), 2)
            new_right = list(f[decimal_left_part])
            a = int(''.join(str(bit) for bit in right_part), 2)
            b = int(''.join(str(bit) for bit in f[
   decimal_left_part]), 2)
            new_left = [int(bit) for bit in bin((a + b) % 16)
   [2:].zfill(4)]
            left_part = new_left
            right_part = new_right
        S_box.append(int(''.join(str(bit) for bit in (left_part
    + right_part)), 2))
    return S_box
```

A.2 Побудова DDT у трьох його варіаціях

```
def build_ddt(s_box):
    ddt = [[0] * 256 for _ in range(256)]
    for input_diff in range(256):
        for input_value in range(256):
            output_diff = s_box[input_value] ^ s_box[input_diff
    ^ input_value]
            ddt[input_diff][output_diff] += 1
    return ddt

def build_ddt_2(s_box):
    ddt = [[0] * 256 for _ in range(256)]
    for input_diff in range(256):
        for input_value in range(256):
            output_diff = (s_box[(input_diff + input_value) %
   256] - s_box[input_value]) % 256
```

```
            ddt[input_diff][output_diff] += 1
    return ddt

def build_ddt_3(s_box):
    ddt = [[0] * 256 for _ in range(256)]
    for input_diff in range(256):
        for input_value in range(256):
            output_diff = s_box[(input_diff + input_value) %
  256] ^ s_box[input_value]
            ddt[input_diff][output_diff] += 1
    return ddt
```

A.3 Запит на побудову хеш-мапи DDT та її запис у текстовий файл

```
def launch_write_hashmap(ddt_type, group_type, txtfile_name):
    qwerty = datetime.datetime.now()
    print(f'start: {qwerty}')
    caps = unique_hashmap(ddt_type, group_type)
    with open(f'{txtfile_name}.txt', 'w') as file:
        file.write(str(caps))
    print(f'time: {datetime.datetime.now() - qwerty}')
```

A.4 Побудова хеш-мапи DDT

```
def unique_hashmap(ddt_type, group_type):
    result = [[{} for _ in range(256)] for _ in range(256)]
    for i in range(100000):
        f1 = list(product([0, 1], repeat=4))
```

```
f2 = list(product([0, 1], repeat=4))
f3 = list(product([0, 1], repeat=4))
random.shuffle(f1)
random.shuffle(f2)
random.shuffle(f3)

if ddt_type == 'xor-xor':
    if group_type == 'fx':
        a = build_ddt(feistel_xor(f1, f2, f3))
    if group_type == 'fp':
        a = build_ddt(feistel_plus(f1, f2, f3))
    if group_type == 'mx':
        a = build_ddt(misty_xor(f1, f2, f3))
    if group_type == 'mp':
        a = build_ddt(misty_plus(f1, f2, f3))
    if group_type == 'rx':
        a = build_ddt(r_sheme_xor(f1, f2, f3))
    if group_type == 'rp':
        a = build_ddt(r_sheme_plus(f1, f2, f3))

if ddt_type == 'plus-plus':
    if group_type == 'fx':
        a = build_ddt_2(feistel_xor(f1, f2, f3))
    if group_type == 'fp':
        a = build_ddt_2(feistel_plus(f1, f2, f3))
    if group_type == 'mx':
        a = build_ddt_2(misty_xor(f1, f2, f3))
    if group_type == 'mp':
        a = build_ddt_2(misty_plus(f1, f2, f3))
    if group_type == 'rx':
        a = build_ddt_2(r_sheme_xor(f1, f2, f3))
```

```
            if group_type == 'rp':
                a = build_ddt_2(r_sheme_plus(f1, f2, f3))

        if ddt_type == 'plus-xor':
            if group_type == 'fx':
                a = build_ddt_3(feistel_xor(f1, f2, f3))
            if group_type == 'fp':
                a = build_ddt_3(feistel_plus(f1, f2, f3))
            if group_type == 'mx':
                a = build_ddt_3(misty_xor(f1, f2, f3))
            if group_type == 'mp':
                a = build_ddt_3(misty_plus(f1, f2, f3))
            if group_type == 'rx':
                a = build_ddt_3(r_sheme_xor(f1, f2, f3))
            if group_type == 'rp':
                a = build_ddt_3(r_sheme_plus(f1, f2, f3))

        for j in range(256):
            for k in range(256):
                var1 = a[j][k]
                if var1 in result[j][k]:
                    result[j][k][var1] += 1
                else:
                    result[j][k][var1] = 1
    return result
```

A.5 Побудова таблиці прийняття гіпотез про розподіл та її запис у текстовий файл

```
def launch_write_hypotesis(ddt_type, txtfile_name):
    qwerty = datetime.datetime.now()
    print(f'start: {qwerty}')
    with open(f'{txtfile_name}.txt', 'r') as file:
        file_content = file.read()
    string_repr = file_content
    caps = ast.literal_eval(string_repr)
    hypotesis_caps = perform_chi_square_test(ddt_type, caps)
    print(max(max(sublist) for sublist in hypotesis_caps))
    with open(f'hypotesis_{txtfile_name}.txt', 'w') as file:
        file.write(str(hypotesis_caps))
    print(f'time : {datetime.datetime.now() - qwerty}')
```

A.6 Перевірка гіпотези χ^2

```
def perform_chi_square_test(ddt_type, hash_map):
    chi_square_stat = 0
    dof = 255
    lambd = 1
    hypotesis_table = [[0] * 256 for _ in range(256)]
    chi_2 = math.sqrt(2 * dof) * 2.33 + dof
    expt_list_half = [poisson.pmf(i, 0.5) * 100000 for i in
   range(257)]
    expt_list_one  = [poisson.pmf(i, 1) * 100000 for i in range
   (257)]

    if ddt_type == 'xor-xor':
        for input_diff in range(1, 256):
            for output_diff in range(1, 256):
                current = hash_map[input_diff][output_diff]
```

```
            chi_square_stat = 0
            freq_list = [current[i] if i in current else 0
for i in range(257)]
            chi_square_stat = sum([((freq_list[i] -
expt_list_half[i]) ** 2) / expt_list_half[i] for i in range
(257)])
            if chi_square_stat <= chi_2:
                hypotesis_table[input_diff][output_diff] =
1
            else:
                hypotesis_table[input_diff][output_diff] =
0

 if ddt_type == 'plus-plus':
     for input_diff in range(1, 256):
         for output_diff in range(1, 256):
            current = hash_map[input_diff][output_diff]
            chi_square_stat = 0
            freq_list = [current[i] if i in current else 0
for i in range(257)]
            chi_square_stat = sum([((freq_list[i] -
expt_list_one[i]) ** 2) / expt_list_one[i] for i in range
(257)])
            if chi_square_stat <= chi_2:
                hypotesis_table[input_diff][output_diff] =
1
            else:
                hypotesis_table[input_diff][output_diff] =
0
     chi_square_stat = 0
     current = hash_map[128][128]
```

```
    freq_list = [current[i] if i in current else 0 for i in
 range(257)]
    chi_square_stat = sum([((freq_list[i] - expt_list_half[
i]) ** 2) / expt_list_half[i] for i in range(257)])
    if chi_square_stat <= chi_2:
        hypotesis_table[input_diff][output_diff] = 1
    else:
        hypotesis_table[input_diff][output_diff] = 0

 if ddt_type == 'plus-xor':
    for input_diff in range(1, 256):
        for output_diff in range(1, 256):
            current = hash_map[input_diff][output_diff]
            chi_square_stat = 0
            freq_list = [current[i] if i in current else 0
for i in range(257)]
            chi_square_stat = sum([((freq_list[i] -
expt_list_one[i]) ** 2) / expt_list_one[i] for i in range
(257)])
            if chi_square_stat <= chi_2:
                hypotesis_table[input_diff][output_diff] =
1
            else:
                hypotesis_table[input_diff][output_diff] =
0
    for output_diff in range(1, 256):
        chi_square_stat = 0
        current = hash_map[128][output_diff]
        freq_list = [current[i] if i in current else 0 for
i in range(257)]
        chi_square_stat = sum([((freq_list[i] -
```

```
expt_list_half[i]) ** 2) / expt_list_half[i] for i in range
(257)])
            if chi_square_stat <= chi_2:
                hypotesis_table[input_diff][output_diff] = 1
            else:
                hypotesis_table[input_diff][output_diff] = 0
    return hypotesis_table
```