НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ

«До захисту допущено»
В.о. завідувача кафедрою
_____ М.М.Савчук
(підпис) (ініціали, прізвище)
“___”_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки: 6.040301 «Прикладна математика»
(код і назва)

на тему: Криптографічні властивості S-блоків, афінно еквівалентних, експоненційні перетворення над скінченним полем

Виконав (-ла): студент (-ка) 4 курсу, групи ФІ-52
(шифр групи)

Кратт Ярослав Володимирович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник: к. т. н. доцент кафедри ММЗІ ФТІ
НТУУ “КПІ ім. Ігоря Сікорського” Яковлєв С.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант: _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент:_____

_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

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

Київ – 2019 року

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

Кафедра математичних методів захисту інформації

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки - 6.040301 «Прикладна математика»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедрою

М.М.Савчук

_____
(підпис) (ініціали, прізвище)

«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту

Кратту Ярославу Володимировичу ,
(прізвище, ім'я, по батькові)

1. Тема роботи Криптографічні властивості S-блоків, афінно еквівалентних, експоненційні перетворення над скінченним полем.

керівник роботи к.т.н. доцент кафедри ММЗІ ФТІ

НТУУ "КПІ ім. Ігоря Сікорського" Яковлєв С. В. ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від _____ р. № _____

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи _____
_____
_____
_____

4. Зміст роботи аналіз криптографічних властивості експоненційних перетворень над скінченними полями. Афінні перетворення експоненційних відображень, які зберігають певні криптографічні властивості. Експериментальні дослідження криптографічних властивостей одержаної множини відображень.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) _____

_____

_____

_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	провести огляд опублікованих джерел за тематикою дослідження;	20.09.18	
2.	проаналізувати криптографічні властивості експоненційних перетворень над скінченними полями;	30.11.18	
3.	побудувати афінні перетворення експоненційних відображень, які зберігають певні криптографічні властивості;	31.03.19	
4.	експериментально дослідити криптографічні властивості одержаної множини відображень	20.05.19	

Студент _____ Кратт Я. В.
(підпис) (ініціали, прізвище)

Керівник роботи _____ Яковлєв С.В.
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Тема роботи: Криптографічні властивості S-блоків, афінно еквівалентних, експоненційні перетворення над скінченним полем.

Кваліфікаційна робота містить: 69 стор., 14 рисунків, 6 таблиць, 9 джерел. Метою роботи була розробка та дослідження надійності криптографічних перетворень та їх складових елементів, що дозволяють підвищити захищеність засобів криптографічного захисту інформаціх.

У ході данної роботи був проведений аналіз роботи по експоненціальним S-блокам Сергія Агієвича та Андрія Афоненко. Саме на роботі цих вчених базувалися проведені мною дослідження. Також було опрацьовано роботи Л. Будагян та К.Карле. Ці роботи присвячені різним видам еквівалентності булевих функцій. Результати цих вчених дали поштовх для наших досліджень в області еквівалентності експоненційних перетворень.

Запропоновано і доведено новий вид еквівалентності експоненційних S-блоків, що зберігає диференціальні імовірності $DP_{+,\oplus}$. Доведено можливість розширення множини доступних експоненційних перетворень для використання в криптографії. Також показано, що одночасне застосування афінного перетворення на вході та на виході експоненційного S-блоку не призводить до розширення цієї множини.

У ході проведення обчислень та досліджень програмним способом було знайдено 64 основи для експоненційного перетворення, що мають низькі диференціальні характеристики та максимальний алгебраїчний степінь. Для цих основ було показано, що множину експоненціальних S-блоків можна розширити шляхом афінних перетворень на вході або на виході зі збереженням низької диференціальної імовірності $DP_{+,\oplus}$, а за деяких умов вдалось навіть покращити інші криптографічні характеристики.

Результати цієї роботи були частково представлені на XVII

Науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики» (26-27 квітня 2019р., м. Київ).

СИМЕТРИЧНА КРИПТОГРАФІЯ,S-БЛОКИ,ЕКСПОНЕНЦІЙНІ S-БЛОКИ, АФІННІ ПЕРЕТВОРЕННЯ В ЕКСПОНЕНЦІЙНИХ S-БЛОКАХ

ABSTRACT

The topic of work: Cryptografic Properties of S-boxes Wich are Affine Equivalent to Exponential Mapping Over Finite Field.

The qualifying paper contains: 69 pages, 14 figures, 6 tables, 9 sources. The purpose of the work was to develop and investigate the reliability of cryptographic transformations and their constituent elements, which allow to increase the security of cryptographic information security tools.

In the course of this work, an analysis of the work on the exponential S-blocks of Sergei Agievich and Andrei Afonenko was conducted. It was at the work of these scientists that I was based on research. Also, works by L. Budagyan and K. Karle were worked out. These papers are devoted to various types of equivalence of Boolean functions. The results of these scientists gave impetus to our research in the field of equivalence of exponential transformations.

A new kind of equivalence of exponential S-blocks is proposed and proved, preserving differential probabilities $DP_{+,\oplus}$. The possibility of expanding the set of available exponential transformations for use in cryptography is proved. It is also shown that the simultaneous application of the affine transformation at the input and output of the exponential S-block does not lead to the expansion of this set.

In the course of calculations and studies, the software method found 64 basis for exponential transformation, which have low differential characteristics and maximum algebraic degree. For these foundations, it has been shown that the set of exponential S-blocks can be expanded by affine transformations at the input or output, with the preservation of a low differential probability $DP_{+,\oplus}$, and under certain conditions it was possible to even improve other cryptographic characteristics.

The results of this work were partially presented at the XVIIth Scientific and Practical Conference of Students, Aspirants and Young Scientists

«Theoretical and Applied Problems of Physics, Mathematics and Informatics» (April 26-27, 2019, Kyiv).

SYMMETRIC CRYPTOGRAPHY, S-BLOCKS, EXPONENTIAL S-BOXES, AFFINE TRANSFORMATIONS IN EXPONENTIAL S-BOXES

ЗМІСТ

ВСТУП

Актуальність дослідження полягає у тому, що симетричні криптографічні примітиви набули широкого поширення завдяки високій продуктивності і відносно легкій реалізації реалізації. Крім забезпечення конфіденційності за допомогою блокових і поточних шифрів, симетричні примітиви використовуються для забезпечення цілісності на основі кодів аутентифікації повідомлень і хеш-функцій, як компоненти електронного цифрового підпису для захисту автентичності, генерації псевдовипадкових послідовностей, в складі протоколів підтвердження автентичності і т.п. Однією з основних складових симетричних криптосистем є S-блоки. Перспективними в цій області є експоненційні S-блоки, що описані в роботі білоруських вчених Сергія Агієвича та Андрія Афоненко. В статті вони описали та довели основні критерії для побудови експоненційних відображень.

Метою дослідження є розробка та дослідження надійності криптографічних перетворень та їх складових елементів, що дозволяють підвищити захищеність засобів криптографічного захисту інформації. Для розв'язання задачі необхідно вирішити такі завдання:

1) провести огляд опублікованих джерел за тематикою дослідження;

2) проаналізувати криптографічні властивості експоненційних перетворень над скінченними полями;

3) побудувати афінні перетворення експоненційних відображень, які зберігають певні криптографічні властивості;

4) експериментально дослідити криптографічні властивості одержаної множини відображень.

Об'єктом дослідження є інформаційні процеси в системах криптографічного захисту.

Предметом дослідження є експоненційні перетворення над скінченними полями та їх криптографічні властивості.

При розв'язанні поставлених завдань використовувались такі *методи дослідження*: методи абстрактної алгебри, теорії імовірностей, математичної статистики, комбінаторного аналізу, теорії складності алгоритмів, методи комп'ютерного та статистичного моделювання.

Наукова новизна отриманих результатів. У роботі вперше досліджено параметри стійкості до диференціального та лінійного криптоаналізу множини афінно модифікованих експоненційних перетворень над скінченними полями характеристики 2. Показано,що афінні модифікації в деяких випадках підсилюють криптографічні властивості експоненційних S-блоків.

Практичне значення результатів полягає в тому, що їх можна використати при побудові надійних симетричних криптопримітивів.

Апробація результатів та публікації. Результати цієї роботи були частково представлені на XVII Науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики» (26-27 квітня 2019р., м. Київ).

1 ТЕОРЕТИЧНИЙ ОГЛЯД

В цьому розділі будть наведені необхідні означення, теоретичні відомості по криптографічним властивостям булевих функцій, S-блокам та видам їх еквівалентності, необхідні терміни з диференціального криптоаналізу. Також буде розглянуто результати, що отримали Сергій Агієвич та Андрій Афоненко

1.1 Булеві функції та їх основні криптографічні властивості

Векторною булевою функцією (n,m) – F [1] називається відображення

$$(n, m) - F : \mathbb{Z}_2^n \to \mathbb{Z}_2^m,$$

де *n,m* – розмірності вхідного і вихідного векторів відповідно. Для того, щоб булеві функції можна було безпечно використовувати в криптографії, вони повинні володіти властивостями, що перешкоджають криптоаналізу. Тому далі буде наведено деякі з них, а також допоміжні означення, для повного розуміння викладу.

Невиродженість

Змінну x_k називають *істотною* для деякої булевої функції $(n, m) - F$, якщо можна знайти такий набір значень

$(\alpha_1, \ldots, \alpha_{k-1}, \alpha_{k+1}, \ldots, \alpha_n)$, де $\alpha_i \in \{0, 1\}$, що

$$F((\alpha_1, \ldots, \alpha_{k-1}, 0, \alpha_{k+1}, \ldots, \alpha_n)) \neq F((\alpha_1, \ldots, \alpha_{k-1}, 1, \alpha_{k+1}, \ldots, \alpha_n))$$

Булеву функцію $(n, m) - F$ називають *невиродженою*, якщо всі її змінні істотні. Якщо функція має неістотні змінні, то це означає, чо не всі біти входу мають вплив на вихідні біти, а це в свою чергу може надати необхідну інформацію для криптоаналізу.

Збалансованість

Булеву функцію $n, m - F$ називають *збалансованою* [2], якщо

$$\forall y \in \{0, 1\} : \#\{F^{-1}(y)\} = 2^{n-m},$$

де $\#\{F^{-1}(y)\}$ – кількість прообразів вектора y.

Ця харакатеристика булевих функцій є надзвичайно важливою Оскільки, якщо функція рівноймовірна, то знання про виходу функції не надає ніякої додаткової інформації про значення вхідних змінних.

Кореляційний імунітет

Розглянемо булеву функцію $(n, m) - F$.Задамо на множині 2^n вхідних векторів $x \in \{0, 1\}^n$ рівномірний розподіл. Він індукує деякий імовірнісний розподіл на множині всіх вихідних векторів $y \in \{0, 1\}^m$ для даної функції.

Кажуть, що булева функція $(n, m) - F$ має *кореляційний імунітет k-ого порядку[3], де* $k \in [1, n]$, якщо для будь-яких

$i_1, \ldots, i_k : i_j \in \{1, \ldots, n\}$ таких, що $i_j \neq i_{j'}$ при $j \neq j'$, для будь-якої координати $y_r, r = \overline{1, m}$ вихідного $m-$вектора взаємна інформація між випадковим булевим вектором $(x_{i_1}, \ldots, x_{i_k})$ і випадковою булевою величиною y_r дорівнює нулю:

$$I(((x_{i_1}, \ldots, x_{i_k})), y_r) = 0.$$

Важливо зазначити, що розглядати відповідні ентропії та взаємну інформацію стало можливо після того, як на множині вхідних векторів $x \in \{0,1\}^n$ було задано ймовірнісний розподіл. Взаємну інформацію між довільними імовірнісними ансамблями X і Y можна знайти за формулою:

$$I(X, Y) = H(X) + H(Y) - H(X, Y)$$

. Дана характеристика дозволяє захиститись від кореляційних атак, які заключаються в тому, що криптоаналітик має можливість знаходити невідомі параметри по частинах.

Лавинний ефект

Нехай на множині вхідних векторів $x \in \{0,1\}^n$ функції $(n, m) - F$ задано рівномірний розподіл:

$$\forall a \in \{0,1\}^n : p(x = a) = \frac{1}{2^n}$$

Тоді імовірнісний розподіл буде задано також на множині всіх вихідних векторів $y \in \{0,1\}^n$ для цієї функції $(n, m) - F$

Позначимо e_i булевий $n-$вектор, $i-$а компонента якого рівна 1, а інші 0; $F|_j$ - $j-$а координатна функція $(n, m) - F$ функції, тоді ймовірність того,

що $j-$а координатна функція булевої функції $F(x) \oplus F(x \oplus e_i)$ рівна 1:

$$p_{ij} = p\Big\{F(x) \oplus F(x \oplus e_i)|_j = 1\Big\}$$

Булева функція $(n, m) - F$ має *строгий лавинний ефект нульогвого порядку*, якщо $\forall i = \overline{1, n}$ та $\forall j = \overline{1, m} : p_{ij} = \dfrac{1}{2}$.

Отже, строгий лавинний ефект нульового порядку означає, що зміна одного біту входу функції призводить до зміни кожного з віхідного бітів з імовірністю $\dfrac{1}{2^n}$. Це означає неможливість для криптоаналітика з'ясувати будь-які зміни на вході функції за змінами, які стались на виході(навіть за мінімальними)

Нелінійність

Однією з найбільш важливих характеристик криптографічних булевих функцій, безперечно, є *нелінійність*, оскільки лінійні та афінні функції можуть бути легко обчислені та обернені. Саме тому рівень нелінійності має буди якнайвищим, для успішного використання в криптографії.

Введемо означення нелійнійності як відстань до класу лінійних функцій:

Нелінійністю булевої функції називається відстань до класу лінійних булевих функцій від n змінних, яка дорівнює:

$$N_f = \min_{l \in L_n} d(l, f) = \min_{l \in L_n}(wt(f \oplus l))$$

1.2 S-блоки та їх властивості

Наведемо означення, які будемо використовувати далі.

S-блок – відображення $\mathbb{F}_2^n \to \mathbb{F}_2^m$ (тобто з n входами та m виходами), що складається з m n-вимірних булевих функцій: $(f_1(x_1, \ldots, x_n), (f_2(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$. Кожна з яких є суттєво нелінійною булевою функцією. Таке відображення приховує залежність між шифротекстом та ключем.

В загальному випадку при конструюванні S-блоків дотримуються критеріїв наведених нижче.

– бієктивність

– строгий лавинний ефект

– кореляційний імунітет

– нелінійність

– збалансованість

Вони є основними, але можуть бути розширені іншими корисними властивостями, що збільшать множину атак, до яких S-блок буде стійким. Варто зауважити, що досягти всіх критеріїв у повному обсязі неможливо. Їх суперечливий характер змушує іти на компроміси. Наприклад, кореляційний імунітет конфліктує з високою нелінійністю, а максимальна нелінійність конфліктує з балансом.

1.2.1 Експоненційні S-блоки

Нехай V_n – множина усіх n-бітових векторів. Дві булеві функції $F, G\colon V_n \to V_n$ є *розширено афінно еквівалентними (EA-еквівалентними)* [4], якщо існують афінні бієктивні функції A_1, A_2

та деяка афінна функція A такі, що виконується

$$F(x) = A_2(G(A_1(x))) \oplus A(x).$$

Для звичайних (багатовимірних) булевих функцій розглядаються афінні перетворення виду $A(x) = L \cdot x \oplus b$, де L – (невироджена) двійкова матриця, x та b – двійкові вектори відповідного розміру.

Існує багато способів генерації S-блоків, але нас цікавлять лише експонеційні S-блоки, спосіб генерації та криптографічні властивості яких наведено в роботі Сергя Агієвича та Андрія Афоненко

Нехай $\mathbb{F}_{2^n}$ – скінченне поле, елементи якого мають представлення як бітові вектори: $x = (x_0 \ldots x_{n-1}) \in \mathbb{F}_{2^n}$.

Експоненційне відображення над полем $\mathbb{F}_{2^n}$ [5] – це відображення виду

$$s : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n} \quad s(x) = \begin{cases} 0 & x = 0 \\ \alpha^{\overline{x}} & x \neq 0 \end{cases} \tag{1.1}$$

де $\overline{x} = x_0 + 2 \cdot x_1 + ... + 2^{n-1} \cdot x_{n-1}$ – число, двійковим представленням якого виступає вектор $x = (x_0, x_1, \cdots, x_{n-1})$, $\alpha \in \mathbb{F}_{2^n}^*$ — примітивний елемент мультиплікативної групи поля. Примітивність елементу α необхідна та достатня для бієктивності наведеного експоненційного відображення. Зауважимо, що $s(0) = 0$, в той час як $a^0 = 1$ (і значення $s(x) = 1$ досягається також при $x = 2^n - 1$). Втім, у тих випадках, коли мова буде йти про значення експоненційного S-блоку у точці 0, ми будемо вважати $a^0 = 0$, а під час певних алгебраїчних перетворень у скінченному полі a^0 традиційно дорівнює 1. Ці випадки зазвичай зрозумілі з контексту та не будуть викликати непорозумінь.

1.2.2 Диференціальний криптоаналіз

Основною ідеєю диференціального криптоаналізу [6] є спроба розкриття секретного ключа блокових шифрів, які працюють за методом повторного застосування криптографічно слабкої цифрові операції шифрування r разів. При аналізі передбачається, що на кожній штерації використовується свій підключ шифрування. Диференціальний криптоаналіз може використовувати як обрані, так і відомі відкриті тексти.

Успіх таких спроб зламу r-циклічного шифру залежить від існування диференціалів $(r-1)$-го циклу, які мають велику ймовірність.

Нехай є деяка функція $f : V_n \to V_n$. Тоді *диференціал функції* f – це пара двійкових вектоорів $(\alpha, \beta) \in V_n^2$, для якої виконується співвідношення:

$$f(x \oplus \alpha) \oplus f(x) = \beta$$

Імовірність диференціалу:

$$DP_f = \frac{1}{2^n} \sum_{x \in V^n} \left[f(x \oplus \alpha) \oplus f(x) = b \right]$$

$$(0,0) - \textit{тривіальний диференціал}$$

Якщо $DP_f(\alpha, \beta) = 0$, тобто такі α та β, що для жодного x не виконується наше співвідношення, то будемо називати такий диференціал *неможливим*. Диференціали, які не є тривіальними та неможливими називають *нетривіальними*.

Для диференціального криптоаналізу важливими та корисними є саме нетривіальні можливі диференціали з максимальною імовірністю. Чим більша ймовірність, тим успішніша буде атака.

Властивості диференціальних імовірностей

1) $DP_f(0,\beta) = [\beta = 0], \forall\beta$;
2) $\forall\alpha : DP_f(\alpha, 0) = [\alpha = 0]$, якщо функія $f-$бієктивна;
3) $\forall\alpha : \sum_\beta DP_f(\alpha,\beta) = 1$;
4) $\forall\beta : \sum_\alpha DP_f(\alpha,\beta) = 1$, якщо f - бієкція;

$$MDP(f) = \max_{\alpha\neq 0,\beta} DP_f(\alpha,\beta)$$

Таблиці для аналізу S-блоків

Перш за все нам необхідні *таблиці диферниціального розподілу(з англ. DDT – differential distribution table).* Для наших досліджень знадобиться декілька таблиць диференціального розподілу, які будуть введені далі.

Нехай $\alpha, \beta \in V_n, f : V_n \to V_n$. Позначимо $+$ – додавання за модулем 2^n. Тоді введемо три таблиці, що будемо використовувати в подальшому: $\forall\alpha,\beta$:

$$DDT^f_{\oplus,\oplus}[\alpha,\beta] = \#\Big\{\forall x \in V_n : f(x\oplus\alpha) = f(x)\oplus\beta\Big\}\cdot\frac{1}{2^n}$$

$$DDT^f_{\oplus,+}[\alpha,\beta] = \#\Big\{\forall x \in V_n : f(x\oplus\alpha) = f(x)+\beta\Big\}\cdot\frac{1}{2^n}$$

$$DDT^f_{+,+}[\alpha,\beta] = \#\Big\{\forall x \in V_n : f(x+\alpha) = f(x)+\beta\Big\}\cdot\frac{1}{2^n}$$

Найвищі значення в DDT [7] (за винятком першого входу) називають *диференціальною однорідністю.* Висока диференціальна рівномірність забезпечує характеристики з високою ймовірністю, отже чим нижче, тим краще. Таким чином, теоретично найкращою досяжною диференціальною

однорідністю є 2.

Також в для аналізу будуть використовуватися таблиці лійного наближеня [8] (з англ. LAT – linear approximation table). Будуються вони наступним чином:

$$\forall \alpha, \beta : LAT^f[\alpha, \beta] = \Big(\sum_{x \in V_n} (-1)^{\alpha \cdot x \oplus \beta \cdot f(x)} \Big)^2$$

1.3 Критерії та властивості експоненційних перетворень

В цьому підрозділі буде наведено результати, що отримали С. Агієвич та А.Афоненко [5]

Диференціальні характеристики

Нехай G_1 та G_2 – абелеві групи однакового порядку, s – бієктивне відображення $G_1 \to G_2$. Нехай

$$u_{ab} = \sum_{x \in G_1} \mathbb{1}\{s(x + a) = s(x) + b\}, a \in G_1, b \in G_2$$

і нехай

$$\mathcal{R}(s) = \max_{a \neq 0, b \neq 0} u_{ab}$$

Ця величина показує наскільки є ефективними методи диференціального криптоаналізу, якщо використовувати s як складову блокового шифру. Малі значення цієї величини ускладнюють застосування цих методів.

Нехай $+$ – додавання по модулю 2^n. Наступна теорема показує, що

значення $\mathcal{R}_{+,\oplus}(s)$, близьке до мінімального (тобто до 2), якщо s - експоненційне перетворення (1.1).

Теорема 1.1. *Якщо*

$$\alpha^{2^{n-1}} \oplus \alpha^t \oplus 1 \neq 0, t = 1, \ldots, 2^{n-1} - 1,$$

тоді $\mathcal{R}(s) = 3$, *інакше* $\mathcal{R}(s) = 4$.

Нехай

$$\nu_i = \sum_{a \neq 0, b \neq 0} \mathbb{1}\{u_{ab} = i\}, i = 0, 1, \ldots, |G_1|$$

Наступна теорема точно описує величину ν_i

Теорема 1.2. *Для бієктивного експоненційного відображення (1.1) виконуються оцінки*

$$\frac{1}{12}q^2 + \frac{1}{3}q - \frac{17}{3} \leq \nu_0 \leq \frac{1}{4}q^2 + 3,$$

$$\frac{1}{2}q^2 - 3q - 8 \leq \nu_1 \leq \frac{5}{6}q^2 - \frac{8}{3}q + \frac{46}{3},$$

$$\frac{1}{12}q^2 - \frac{2}{3}q - \frac{32}{3} \leq \nu_2 \leq \frac{1}{4}q^2 + q + 8,$$

$$\nu_3 \leq q,$$

$$\nu_4 \leq 1,$$

$$\nu_i = 0, i \geq 4.$$

Нелінійність

Нехай L_n множина усіх афінних булевих функцій від n змінних,які мають вигляд:

$$l(x) = \langle b, x \rangle \oplus c = b_0 x_0 \oplus \ldots \oplus b_{n-1} x_{n-1} \oplus c, b \in V_n, c \in \mathbb{F}_2.$$

Нехай L_n^* отримано з L_n після видалення функції тотожно рівної нулеві. Позначимо як $\Lambda(s)$ лінійний проміжок (з коефіцієнтами із $\mathbb{F}_2$) координатних функцій підстановки $s : V_n \to V_n$.

Нелінійність $\mathcal{N}(s)$ визначається наступним чином:

$$\mathcal{N}(s) = d(L_n^*, \Lambda(s)) = \min_{l(x) \in L_n^*} d(l(x), \Lambda(s)) = \min_{\substack{l(\overline{x}) \in L_n^* \\ \sigma(x) \in \Lambda(\overline{s})}} d(l(x), \sigma(x)),$$

де $d(l(x), \sigma(x)) = \sum_{x \in V_n} \mathbb{1}\{l(x) \neq \sigma(x)\}$ - відстань Хеммінга між таблицями істинності для $l(x)$ і $\sigma(x)$. Великі значення величини $\mathcal{N}(s)$ підвищують стійкість проти методів лінійного криптоаналізу при використанні s як компоненту блочного шифру.

Пряме обчислення $\mathcal{N}(s)$ може бути спрощене, використовуючи наступну теорему.

Теорема 1.3. *Нехай $l(x) = \langle b, x \rangle$, $l'(x) = \langle b', x \rangle$ - лінійні функції з n змінними і $b' = \rho^d(b)$ для деякого цілого* d. *Тоді для експоненціальної підстановки $s : V_n \to V_n$:*

$$d(l(x), \Lambda(s)) = d(l'(x), \Lambda(s)) \tag{1.2}$$

де ρ – оператор циклічного зсуву вправо визначений на $V_n : \rho(x) = (x_{n-1}, x_0, \ldots, x_{n-2})$

Наступна теорема дає нижню оцінку для нелінійності експоненціальної підстановки.

Теорема 1.4. *Нехай* $r = 2^n - 1$, $K(b)$ *набір індексів ненульових координат із* $b \in V_n$, *і:*

$$\prod(b) = \frac{1}{r} \sum_{h=1}^{r-1} \prod_{k \in K(b)} \left| tan \frac{\pi 2^k h}{r} \right| \tag{1.3}$$

Тоді:

$$\mathcal{N}(s) \geq 2^{n-1} - 1 - 2^{\frac{n}{2}-1} \max_{\substack{b \in V_n \\ b \neq 0}} \prod(b) \tag{1.4}$$

для експоненціальної підстановки $s : V_n \to V_n$.

Нажаль не вдалось знайти задовільної верхньої оцінки для значення $\prod(b)$, так як прямі обчислення показали, що $\prod(b)$ залежить від числа $w(b)$ – числа ненульових координат b. Як правило $\prod(b)$ приймає максимум, коли $b = n$.

Степені координатних функцій

Ненульова функція $\sigma(x) = \sigma(x_0, \ldots, x_{n-1}) \in \Lambda(s)$ може бути представлена як поліном кільця $\mathbb{F}_2[x_0, \ldots, x_{n-1}]$. Коли s використовується як компонента блочного шифру, бажано, щоб степінь $\deg(\sigma)$ цього поліному була великою, тому що це дозволяє ускладнити використання диференціальних атак.

Теорема 1.5. *Якщо* $s : V_n \to V_n$ *- експоненціальна підстановка, то для будь-якої ненульової функції* $\sigma(x_0, \ldots, x_{n-1}) \in \Lambda(s)$*:*

$$deg(\sigma) \geq n - \lceil \log_2(n+1) \rceil, \tag{1.5}$$

де $\lceil z \rceil$ *- найменше ціле число* $\geq z$.

Наступна теорема визначає критерій для ненульових функцій $\Lambda(s)$ коли вони мають максимальну степінь $n - 1$.

Означення 1.1. $\alpha \in \mathbb{F}_{2^n}$ називається *нормальним елементом над* $\mathbb{F}_2$, якщо $\alpha, \alpha^2, \ldots, \alpha^{2^{n-1}}$ утворюють базис в $\mathbb{F}_{2^n}$ над $\mathbb{F}_2$.

Теорема 1.6. *Якщо* $s : V_n \to V_n$ *- експоненціальна підстановка, то* $deg(\sigma) = n - 1$ *для всіх* $\sigma(x_0, \ldots, x_{n-1}) \in \Lambda(s)$ *тоді і тільки тоді, коли* $\alpha = \alpha(1 + \alpha)^{-1}$ *- нормальний елемент над* $\mathbb{F}_2$.

Лавинний ефект

При зміні компоненти x_j для $\sigma(x) \in \Lambda(s$ визначимо наступну ймовірність:

$$p_j(\sigma) = P\{\sigma(x_0, \ldots, x_j, \ldots, x_{n-1}) \neq \sigma(x_0, \ldots, x_j + 1, \ldots, x_{n-1})\}, \qquad (1.6)$$

враховуючи, що x - випадковий вектор із V_n.

Теорема 1.7. *Якщо* $s : V_n \to V_n$ *- експоненціальна підстановка, то для будь-якої ненульової функції* $\sigma(x) \in \Lambda(s)$ *, а також для всіх* $j = 0, 1, \ldots, n - 1$ *виконується умова:*

$$\left| p_j(\sigma) - \frac{1}{2} \right| < \frac{\ln 2^n - 1}{\pi 2^{\frac{n}{2}}} + \frac{1}{2^{\frac{n}{2}+1}} + \frac{1}{2^{n-1}} \qquad (1.7)$$

1.4 Еквівалентність S-блоків

Нехай A та B оборотні лінійні відображення над $GF(2)$, а a та b n-бітні вектори над $GF(2)$. Тоді оборотні S-блоки S_1 та S_2 є *афінно еквівалентними*, якщо виконується наступне співвідношення:

$$S_1(x) = B^{-1} \cdot S_2(A \cdot x \oplus a) \oplus b$$

Множину таких афінно еквівалентних S -блоків називають *класом афінної еквівалентності*

Наступні твердження та теореми було наведено та доведено в роботах Лілі Будагян та Клод Карле[4, 9]. Саме вони запропонували термін *CCZ-еквівалентність*.

Дві векторні функції f та f' з $\mathbb{F}_2^n$ в $\mathbb{F}_2^m$ називають *CCZ-еквівалентними*, якщо їх графи $G_f = \{(x, f(x)); x \in \mathbb{F}_2^n\}$ і $G_{f'} = \{(x, f'(x)); x \in \mathbb{F}_2^n\}$ є афінно еквівалентними, це означає, що існує афінна перестановка $\mathcal{L} : \mathbb{F}_2^n \times \mathbb{F}_2^n$ така, що $\mathcal{L}(G_f) = \mathcal{L}(G_{f'})$

Нехай V_n – множина усіх n-бітових векторів. Дві булеві функції $F, G\colon\ V_n \to V_n$ є *розширено афінно еквівалентними (EA-еквівалентними)*[4], якщо існують афінні бієктивні функції A_1, A_2 та деяка афінна функція A такі, що виконується

$$F(x) = A_2(G(A_1(x))) \oplus A(x).$$

Розширена еквівалентність є частковим випадком CCZ-еквівалентності. Крім того, кожне перетворення CCZ-еквівалентно оберненому перетворенню.

Для звичайних (багатовимірних) булевих функцій розглядаються афінні перетворення виду $A(x) = L \cdot x \oplus b$, де L – (невироджена) двійкова матриця, x та b – двійкові вектори відповідного розміру.

Для одновимірних функцій:

Теорема 1.8. *Нехай f булева функція на $\mathbb{F}_{2^n}$ і f' – (n,m) -функція. Тоді f і f' є CCZ-еквівалентними, тоді і тільки тоді, коли вони є EA-еквівалентними.*

Длябагатовимірних функцій:

Теорема 1.9. *Нехай $n \geq 5$ і $k > 1$ – найменший дільник n. Тоді для*

будь-якого $m \geq k$, CCZ-еквівалентність (n,m)-функцій є строго більш загальною, ніж EA-еквівалентність.

Таким чином, можна зробити підсумок по CCZ-еквівалентності:

– З точки зору CCZ-еквівалентності, багатовимірні функції поводяться інакше, ніж одновимірні.

– Однак, деякі класи векторних функцій поводяться так само, як і одновимірні(тобто в цих класах CCZ=EA). Прикладом такого підкласу є клас бент(цілком нелінійних) функцій.

– Навіть якщо CCZ-еквівалентність і EA-еквівалентність ідентичні для булевих функцій і для бент функцій, можна використовувати CCZ-еквівалентність для отримання, з відомих бент функцій, булевих бент функцій, які є новими аж до EA-еквівалентності.

Висновки до розділу 1

В цьому розділі було наведено основні теоретичні відомості та результати, що вже були отримані. Відмічено основні криптографічні властивості булевих функцій, що є важливими для експоненційних відображень. Було наведено відомості про експоненційні відображення в скінченних полях характеристики 2. Наведено деякі властивості експоненційних S блоків, завдяки яким їх можна використовувати в криптосистемах. Було розглянуто результати робіт по експоненційним S-блокам С. Агієвича та А. Афоненко, а також результати Л. Будагян та К. Карлет по видам еквівалентності одновимірних та багатовимірних булевих функцій.

2 КРИПТОГРАФІЧНІ ВЛАСТИВОСТІ АФІННИХ МОДИФІКАЦІЙ ЕКСПОНЕНЦІЙНИХ S-БЛОКІВ

В цьому розділі буде побудовано новий вид еквівалентності S-блоків який є схожим на CCZ-еквівалентність, але на відміну неї, зберігає диференціальну імовірність $DP_{+,\oplus}$. Це було зроблено шляхом побудови афінного перетворення на вході та виході S-блоку.

2.1 Цілком афінна еквівалентність експоненційних S-блоків

Для спрощення викладення будемо записувати експоненційні S–блоки як

$$s(x) = a^x, \text{ де } x \in \mathbb{Z}_{2^n},\ s(x) \in \mathbb{F}_{2^n}.$$

Зауважимо, що $s(0) = 0$, в той час як $a^0 = 1$ (і значення $s(x) = 1$ досягається також при $x = 2^n - 1$). Втім, у тих випадках, коли мова буде йти про значення експоненційного S-блоку у точці 0, ми будемо вважати $a^0 = 0$, а під час певних алгебраїчних перетворень у скінченному полі a^0 традиційно дорівнює 1. Ці випадки зазвичай зрозумілі з контексту та не будуть викликати непорозумінь.

Оскільки S-блок є бієктивним тоді і тільки тоді, коли a – генератор мультиплікативної групи поля $\mathbb{F}_{2^n}$, то далі будемо розглядати лише такі елементи $a \in \mathbb{F}_{2^n}^*$, які є генераторами.

Використання додаткових афінних перетворень на вході чи на виході S-блоку дозволяє збільшити кількість доступних для застосування перетворень при збереженні багатьох криптографічних властивостей. Саме в сенсі збереження, зокрема, нелінійності, диференціальних імовірностей та лавинних ефектів ми кажемо про «еквівалентність»

S-блоків. Класична EA-еквівалентність розглядає входи та виходи S-блоку як звичайні двійкові вектори; відповідно, афінні перетворення, які використовуються, розглядаються над лінійним векторним простором V_n.

Для експоненційних S-блоків використання EA-еквівалентності не завжди дозволяє зберегти бажані криптографічні властивості, оскільки вхідні значення експоненційного S-блоку фактично є лишками за модулем 2^n, а вихідні значення – елементами скінченного поля $\mathbb{F}_{2^n}$. Тому ми пропонуємо розглянути інший спосіб побудови еквівалентних S-блоків.

Розглянемо афінні перетворення таких двох типів.

1) $A(x) = (u \cdot x + v) \bmod 2^n$, де x, u, v – числа за модулем 2^n, всі операції виконуються за модулем 2^n та $\gcd(u, 2^n) = 1$ (тобто u є непарним числом). За таких обмежень $A(x)$ – невироджене перетворення над $\mathbb{Z}_{2^n}$.

2) $B(x) = c \cdot x \oplus d$, x, c, $d \in \mathbb{F}_{2^n}$, всі операції є операціями з поля та $c \neq 0$. За таких обмежень $B(x)$ – невироджене перетворення над $\mathbb{F}_{2^n}$.

Будемо казати, що S-блоки $s_1(x)$ та $s_2(x)$ є *цілком афінно еквівалентними*, якщо

$$s_2(x) = B(s_1(A(x))).$$

S-блок $s_2(x)$, який є цілком афінно еквівалентним деякому експоненційному S-блоку, будемо називати *розширеним експоненційним S-блоком*. Метою подальшого розгляду є встановлення найпростіших способів будувати розширені експоненційні S-блоки.

Нехай $s(x) = a^x$ – деякий експоненційний S-блок. Застосувавши афінне перетворення лише на вході, одержуємо

$$s(A(x)) = s(u \cdot x + v) = a^{u \cdot x + v}.$$

Позначимо $c = a^v$, $w = a^u$; тоді $s(A(x)) = c \cdot w^x$. Бачимо, що отриманий вираз схожий на результат застосування афінного (навіть лінійного) перетворення на виході S-блоку. Але $s(A(x))$ буде розширеним експоненційним перетворенням лише в тому випадку, коли w – генератор

мультиплікативної групи $\mathbb{F}_{2^n}^*$. А w буде генератором тоді і тільки тоді, коли u та $(2^n - 1)$ – взаємнопрості.

Можемо остаточно сформулювати таке твердження: якщо $\gcd(u, 2^n - 1) = 1$, то для довільного експоненційного S-блоку $s(x)$ існує такий експоненційний S-блок $s'(x)$, що $s(A(x)) = B(s'(x))$, причому $B(x) = c \cdot x$. Отже, значна частина афінних перетворень на вході може бути замінена деякими афінними перетвореннями на виході для іншого експоненційного S-блоку. Наприклад, автором було розраховано кількість таких значень $u \in \mathbb{Z}_{256}$, що $\gcd(u, 2^8 - 1) = 1$ та $\gcd(u, 2^8) = 1$; їх виявилось 64, тобто рівно половина від усіх можливих значень. Відповідно, використання таких перетворень одночасно не призводить до збільшення кількості розширених експоненційних S-блоків.

Важливим частковим випадком є використання відображень зсуву $A(x) = x + v$ (тобто афінних перетворень із $u = 1$). У даному випадку маємо

$$s(A(x)) = a^{x+v} = c \cdot a^x = c \cdot s(x) = B(s(x)),$$

тобто зсуви переводять експоненційний S-блок у розширений (на виході) експоненційний S-блок від тієї ж експоненти. Так само, оскільки a є генератором, а c – елементом мультиплікативної групи поля, то $c \cdot s(x) = a^v \cdot a^x = a^{x+v}$.

Підсумовуючи, можемо стверджувати, що для заданого експоненційного S-блоку існує лише два нерівносильні шляхи побудувати цілком афінно еквівалентний S-блок, а саме: 1) використовувати афінні перетворення $A(x) = (u \cdot x + v) \bmod 2^n$ на вході, де $\gcd(u, 2^n - 1) \neq 1$;

2) використовувати афінні перетворення $B(x) = c \cdot x \oplus d$ на виході, де $d \neq 0$.

Дослідженню властивостей таких класів розширених S-блоків планується присвятити подальші дослідження.

Таким чином було запропоновано новий вид еквівалентності. Показано, що одночасне застосування афінних перетворень перетворення

на вході та виході експоненційного S-блоку не розширює множину доступних для використання в криптографії S-блоків

2.2 Результати досліджень

Проведені дослідження мали на меті дослідити властивості експоненційних S-блоків, а також розширити їх множину шляхом афінних перетворень. Росширення множини таких S-блоків є можливим, це було показано в попередній частині. Досліджено S-блоки з основами, отриманими за критеріями С. Агієвича та А. Афоненко [5], за всіма входами, за афінним перетворенням входів, а також афінним перетворенням виходів. Всі обчислення проводились на комп'ютері з наступними характеристиками:

– процессор Intel Core i7-8700 3.20 GHz
– 32GB оперативної пам'яті
– операційна система – Windows 10
– мова програмування – Java
– Інтегроване середовище розробки – IntelliJ IDEA

Для пришвидшення обрахунку результатів та побудови графіків, обчислення було розпаралелено таким чином, щоб рівномірно навантажити всі ядра процессора.

Всі дослідження були проведені для $\mathbb{F}_{2^8}$. Тому далі розглядатимуться лише $\alpha \in \mathbb{F}_{2^8}$

Відбір основ для побудови S-блоків

Серед наведених в роботі [5] критерів відбору, використано відбір за дифиренціальними характеристиками (Теорема 1.1) та за максимальним алгебраїчним степенем (Теорема 1.6).

Для того, щоб забезпечити бієктивність експоненційного відображення (1.1) необхідно і достатньо, щоб $\alpha \in \mathbb{F}_{2^n}^*$, тобто α –примітивними елементами мультиплікативної групи поля. Тому спочатку серед всіх $\alpha \in \mathbb{F}_{2^8}$ було відібрано лише ті, які є примітивними.

Якщо для всіх простих дільників p_i числа $2^8 - 1$ виконується $(2^8 - 1, p_i) \neq 1$, то α – примітивний.

Було знайдено 127 таких α.

Наступним кроком став відбір основ за диференціальними характеристиками. Після застосування відповідного критерю С. Агієвича та А. Афоненко:

$$\alpha^{2^{n-1}} \oplus \alpha^t \oplus 1 \neq 0, t = 1, \ldots, 2^{n-1} - 1$$

до вже відібраних основ, кількість допустимих основ з оптимальними диференціальними характеристиками зменшилась на 33, тобто допустимих основ після застосування цього критерію залишилось 94.

Оскільки нашою метою було залишити лише ті основи, S-блоки з використанням яких, мали б оптимальні криптографічні властивості, а саме мали низькі диференціальні імовірності та максимальний алгебраїчний степінь, тому було використано ще один критерій з роботи білоруських вчених:

Якщо $\alpha(1 + \alpha)^{-1}$ – нормальний елемент над F_2, то $\deg(\alpha^x)) = n - 1$. Після фільтрації залишилось 64 основи, що задовольняють критеріям:

`3, 5, 6, b, e, 11, 14, 17, 18, 1a, 21, 31, 45, 46, 48, 4c, 52,`

54, 57, 58, 5b, 5f, 6d, 6e, 71, 76, 7b, 7e, 81, 84, 88, 8a, 90,
93, 95, 96, 99, 9b, a0, a5, a7, a9, ad, b4, b7, b8, ba, bf, c0,
c4, c8, da, dd, de, e3, e5, e7, eb, f0, f5, f6, f8, fb, fd

Далі для побудови характеристик використовуються лише ці α.

Експериментальне обчислення розподілів DDT та LAT

В цьому підрозділі буде розглянуто три множини експоненційних S-блоків. А саме, чисто експоненційні S-блоки, таких було знайдено 16384, S-блоки отримані афінним перетворенням на вході (268,435,456) і стільки ж отримано шляхом афінного перетворення на виході S-блоку. Для кожногого S-блоку з множини, було обчислено параметри DDT, LAT, Λ та l. Знайшовши максимуми для кожної з таблиць, було побудовано розподіл максимумів для кожної з множин.

Для заданих чисто експоненціальних S-блоків було реалізовано два способи 2.1 побудувати цілком афінно еквівалентний S-блок, а саме:

1) Застосувати афінні перетворення $A(x) = (u \cdot x + v) \bmod 2^n$ на вході, де $\gcd(u, 2^n - 1) \neq 1$;

2) Застосувати афінні перетворення $B(x) = c \cdot x \oplus d$ на виході, де $d \neq 0$.

Далі будуть наведені розподіли максимумів та отримані результати.

Таблиця 2.1 – Статистика максимумів для таблиць $DDT^s_{\oplus,\oplus}$

$max(DDT^s_{\oplus,\oplus})$	Чисті экпоненти		Афінні по входу		Афінні по виходу	
	Кількість	%	Кількість	%	Кількість	%
8	48	75%	399,297	37,9%	785,797	75%
10	16	25%	612,200	58,7%	262,070	25%
12	0		36,467	3,4%	0	

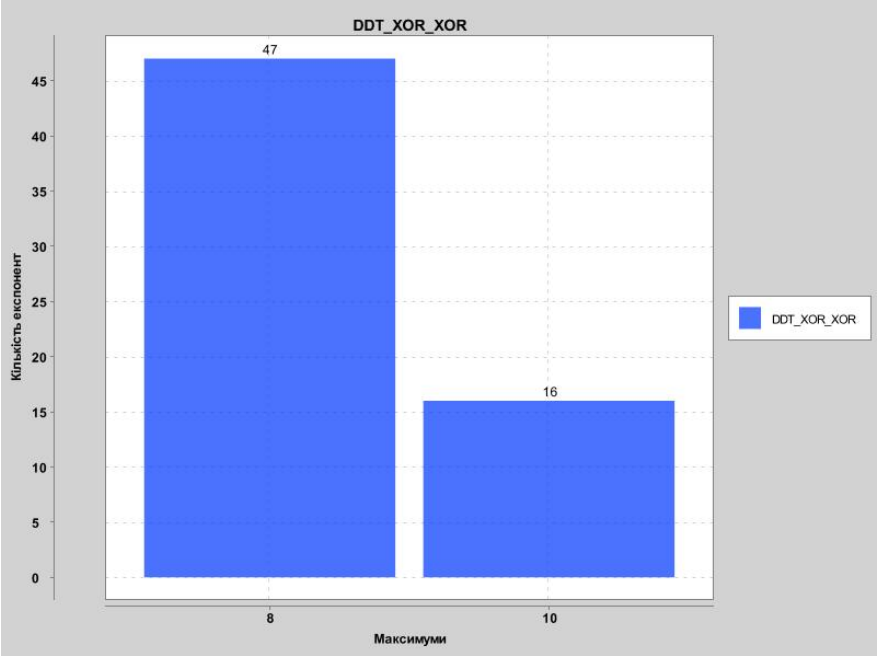

Рисунок 2.1 – $DDT^s_{\oplus,\oplus}$ для чистих експонент

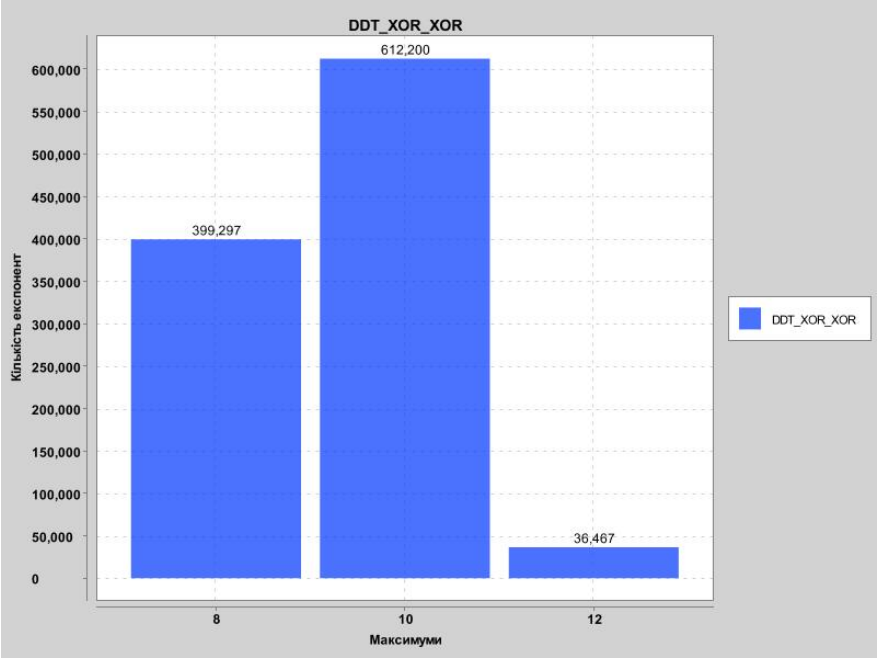

Рисунок 2.2 – $DDT^s_{\oplus,\oplus}$ для афінних по входу

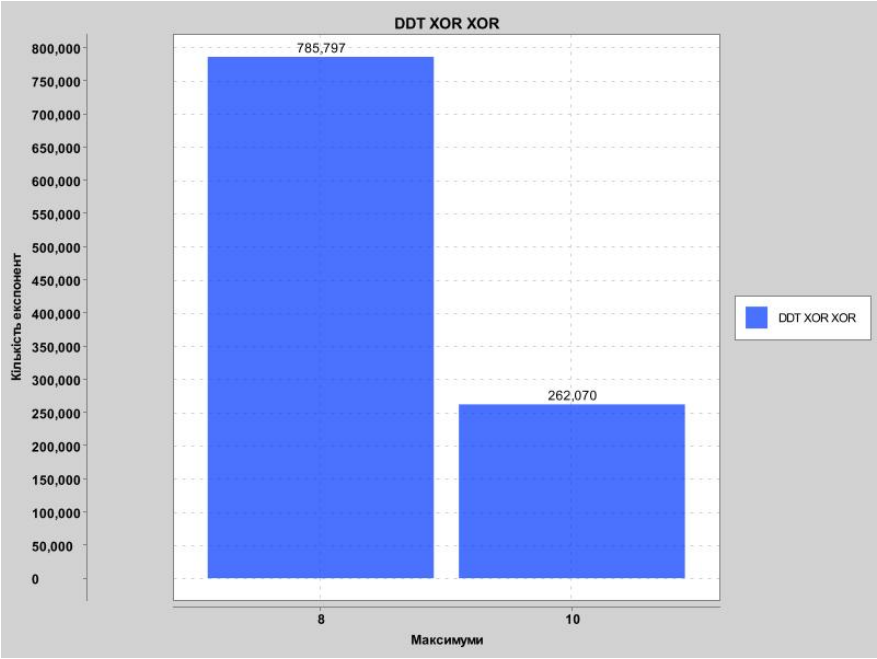

Рисунок 2.3 – $DDT^s_{\oplus,\oplus}$ для афінних по виходу

Таблиця 2.2 – Статистика максимумів для таблиць $DDT^s_{\oplus,+}$

$max(DDT^s_{\oplus,+})$	Чисті экпоненти		Афінні по входу		Афінні по виходу	
	Кількість	%	Кількість	%	Кількість	%
6	0		88,985	8.49%	96,726	9.21%
7	32	50%	426,852	40.72%	393,504	37.48%
8	16	25%	271,772	25.92%	276,310	26.32%
9	0		62,697	5.98%	58,939	5.61%
10	0		44,195	4.22%	47,792	4.55%
11	0		31,023	2.96%	34,351	3.27%
12	0		29,559	2.82%	34,032	3.24%
13	0		27,439	2.62%	30,451	2.9%
14	16	25%	49,455	4.72%	24,313	2.32%
15	0		8,144	0.78%	21,881	2.08%
16	0		7,232	0.69%	15,802	1.5%
17	0		1,008	0.1%	8,958	0.85%
18	0		0		4,799	0.46%
19	0		0		1,791	0.17%
20	0		0		320	0.03%

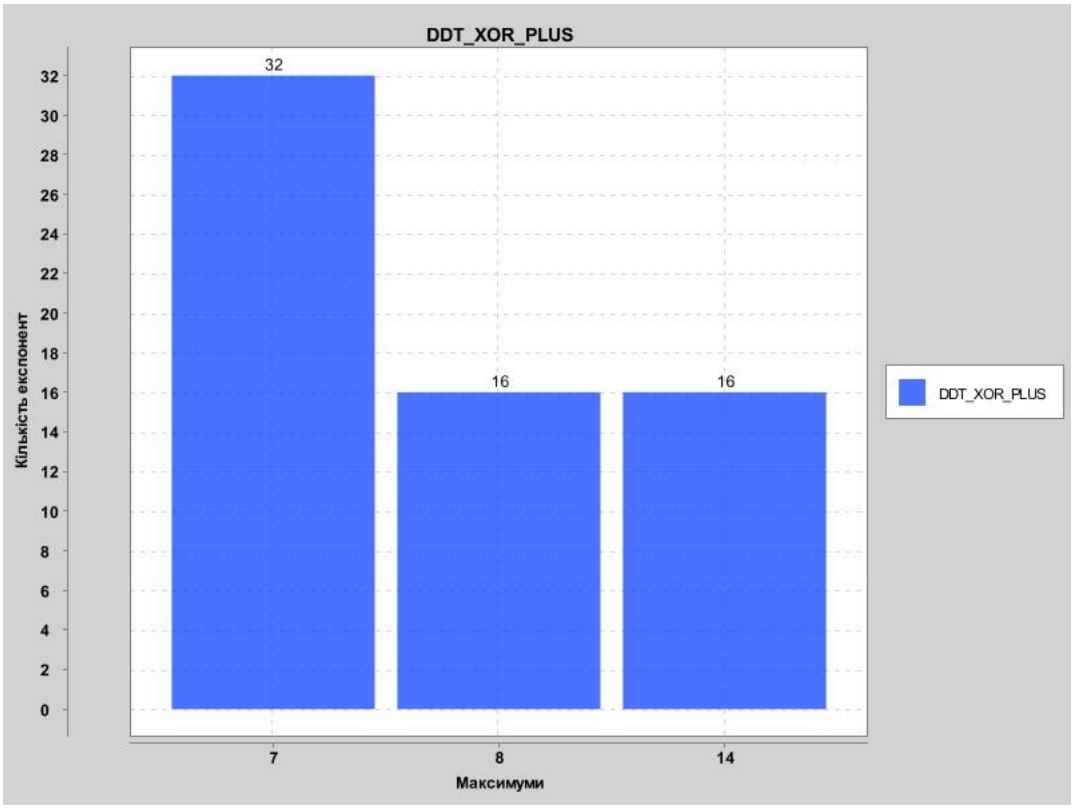

Рисунок 2.4 – $DDT^s_{\oplus,+}$

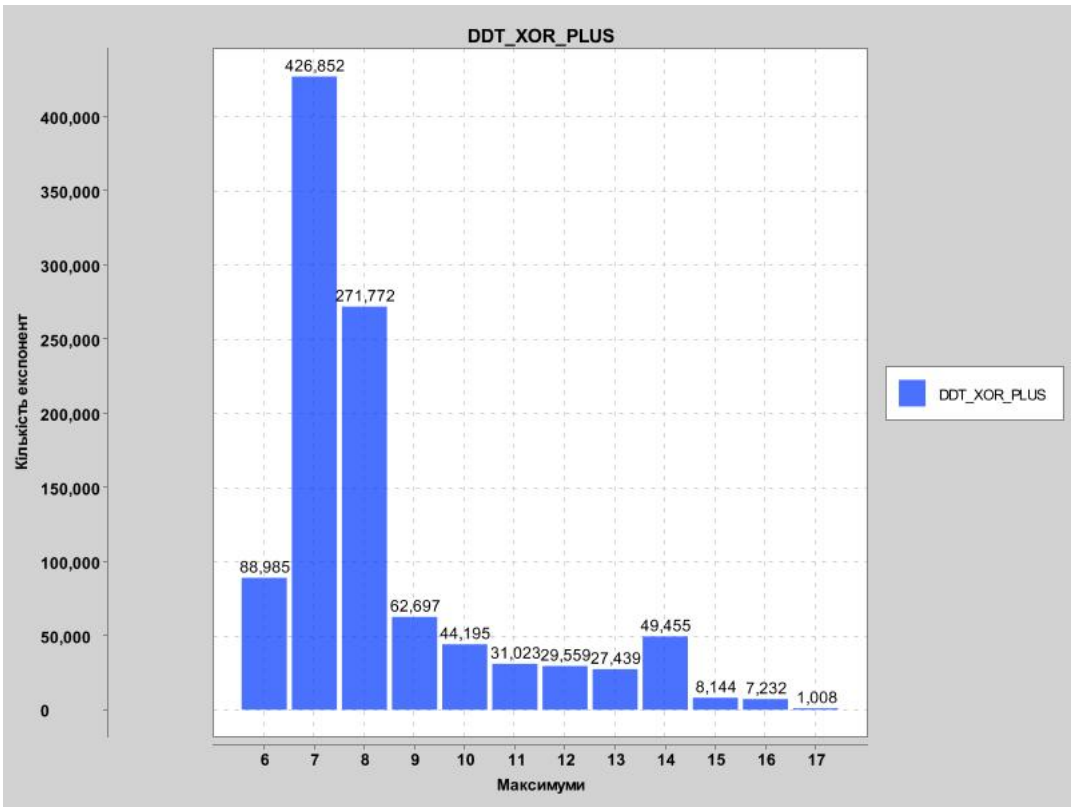

Рисунок 2.5 – $DDT^s_{\oplus,+}$ для афінних по входу

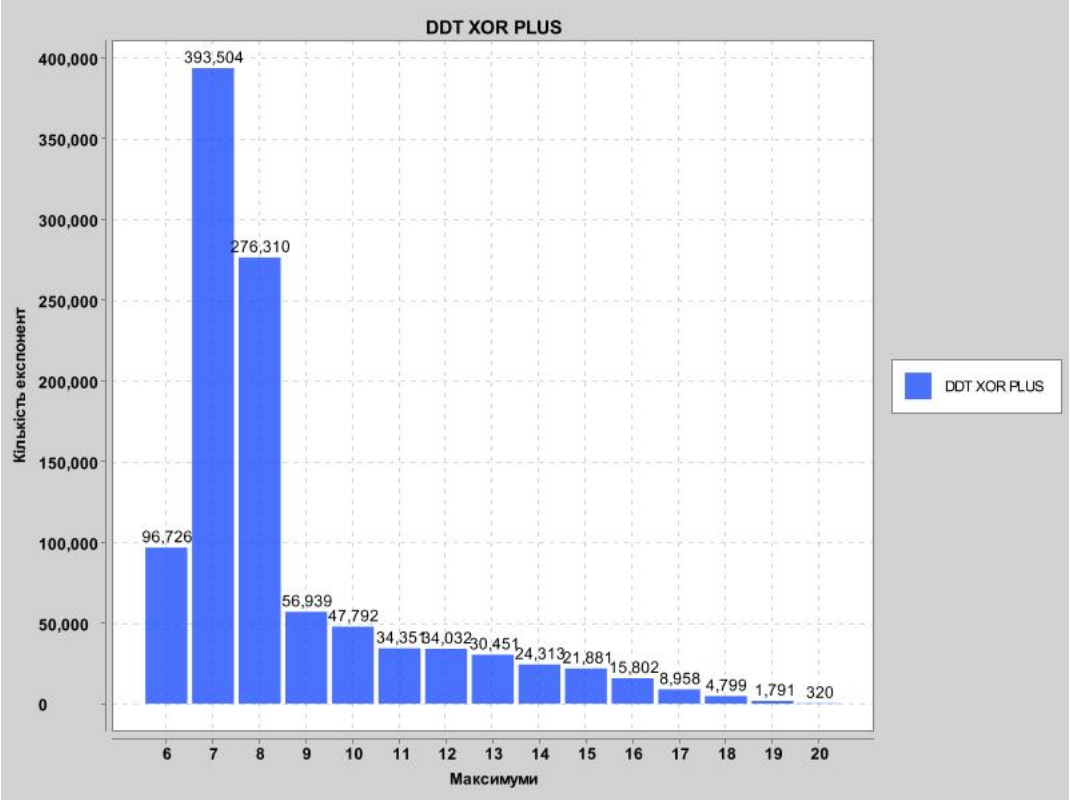

Рисунок 2.6 – $DDT^s_{\oplus,+}$ для афінних по виходу

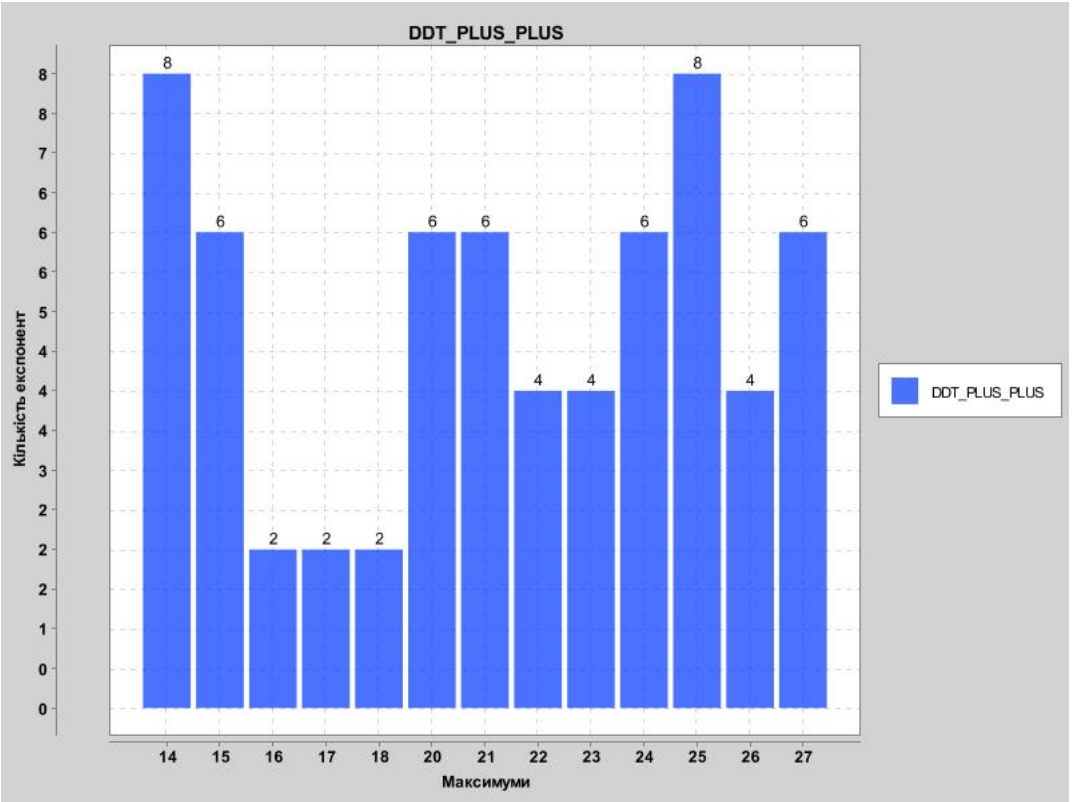

Рисунок 2.7 – $DDT^s_{+,+}$ чисті експоненти

Таблиця 2.3 – Статистика максимумів для таблиць $DDT^s_{+,+}$

$max(DDT^s_{+,+})$	Чисті экпоненти		Афінні по входу		Афінні по виходу	
	Кількість	%	Кількість	%	Кількість	%
8	0				168	0.02%
9	0				5,198	0.52%
10	0				19,590	1.94%
11	0				32,440	3.22%
12	0				44,451	4.41%
13					60,830	6.04%
14	8	12.5%	131,056	12.52%	75,803	7.52%
15	6	9.3%	98,298	9.39%	83,357	8.27%
16	2	3.1%	32,744	3.13%	93,789	9.31%
17	2	3.1%	32,768	3.13%	86,709	8.61%
18	2	3.1%	32,768	3.13%	69,462	6.89%
19	0				68,539	6.79%
20	6	9.3%	98,279	9.39%	53,883	5.35%
21	6	9.3%	98,259	9.39%	51,589	5.12%
22	4	6.25%	65,535	6.26%	55,708	5.53%
23	4	6.25%	32,768	3.13%	52,495	5.21%
24	6	9.3%	131,046	12.52%	38,642	3.84%
25	8	12.5%	131,058	12.52%	38,702	3.84%
26	4	6.25%	65,536	6.26%	28,573	2.84%
27	6	9.3%	98,274	9.39%	23,750	2.34%
28	0		0		16,008	1.59%
29	0		0		20,971	2.08%
30	0		0		10,536	1.05%
31	0		0		9,175	0.91%
32	0		0		2,432	0.24%
33	0		0		2,224	0.22%
34	0		0		3,368	0.33%
35	0		0		48	0.004%

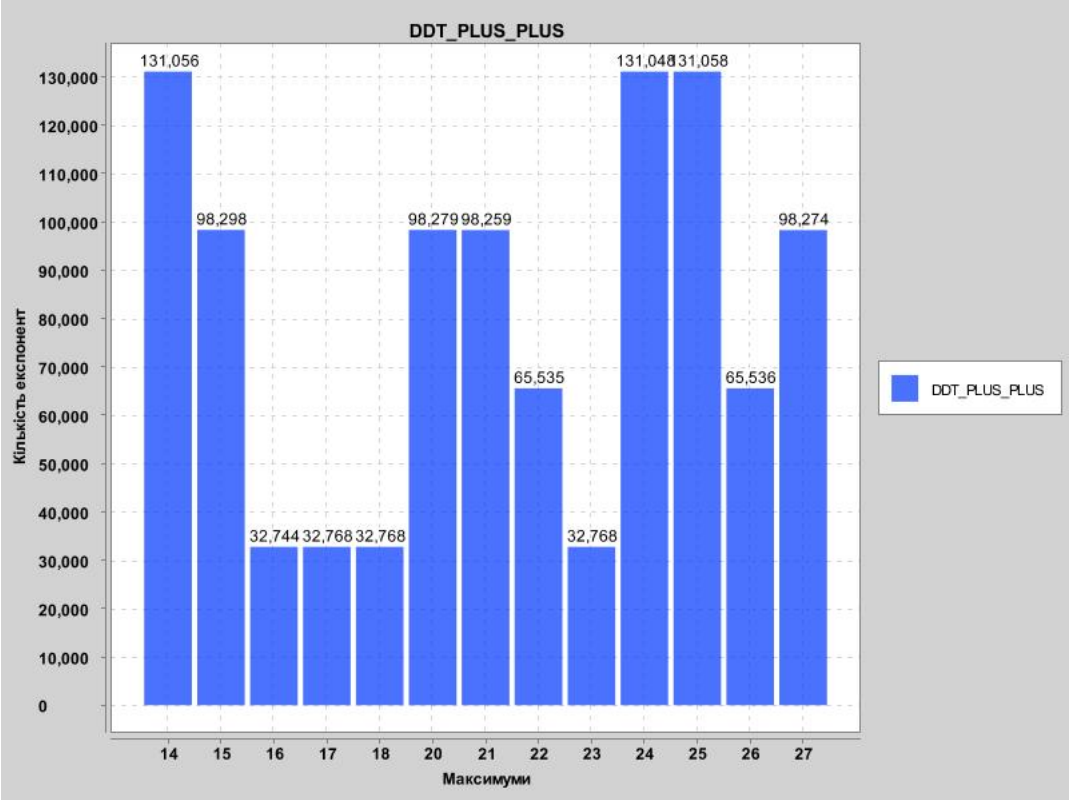

Рисунок 2.8 – $DDT^s_{+,+}$ афінні по входу

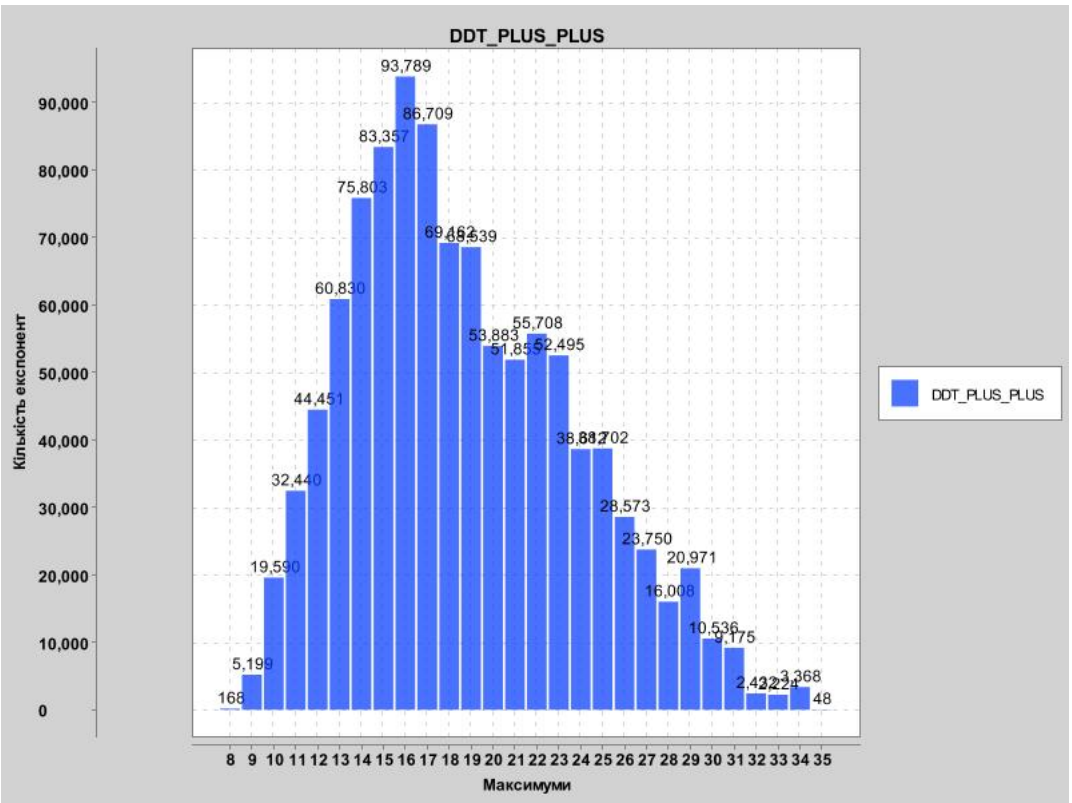

Рисунок 2.9 – $DDT^s_{+,+}$ афінні по виходу

Таблиця 2.4 – Статистика максимумів для таблиць LAT^s

$max(LAT^s)$	Чисті экпоненти		Афінні по входу		Афінні по виходу	
	Кількість	%	Кількість	%	Кількість	%
2704	16	75%	1,023	0.1%	262,011	25%
2916	0		3,711	0.35%	0	
3136	48	25%	11,581	1.1%	785,646	75%
3364	0		68,325	6.52%	0	
3600	0		203,098	19.38%	0	
3844	0		253,642	24.2%	0	
4096	0		188,919	18.02%	0	
4356	0		137,090	13.08%	0	
4624	0		83,172	7.94%	0	
4900	0		45,425	4.33%	0	
5184	0		29,046	2.77%	0	
5476	0		15,932	1.52%	0	
5776	0		5,631	0.54%	0	
6084	0		1,408	0.13%	0	
6400	0		128	0.1%	0	

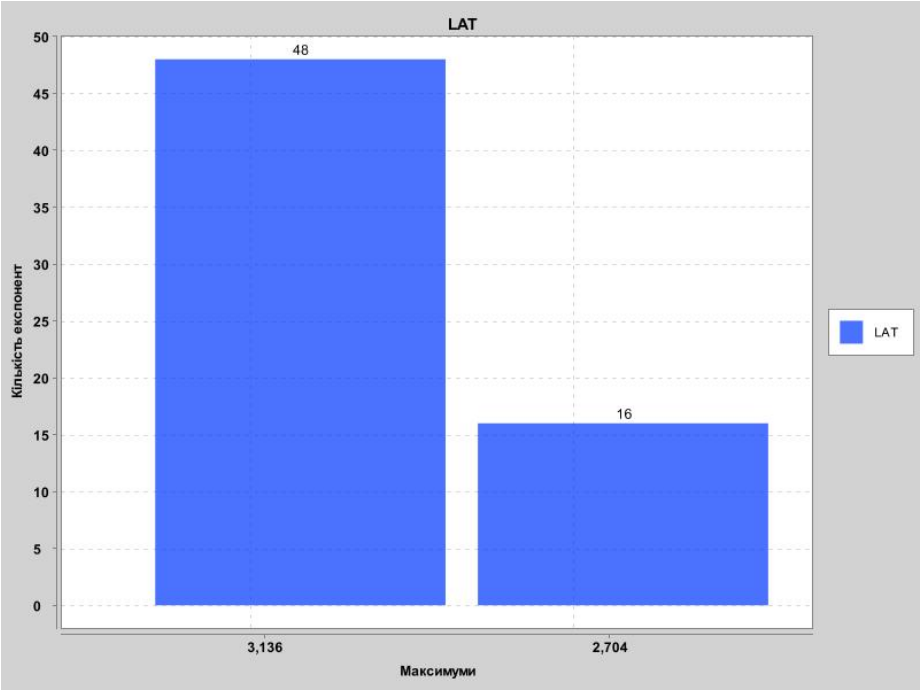

Рисунок 2.10 – LAT^s для чистих експонент

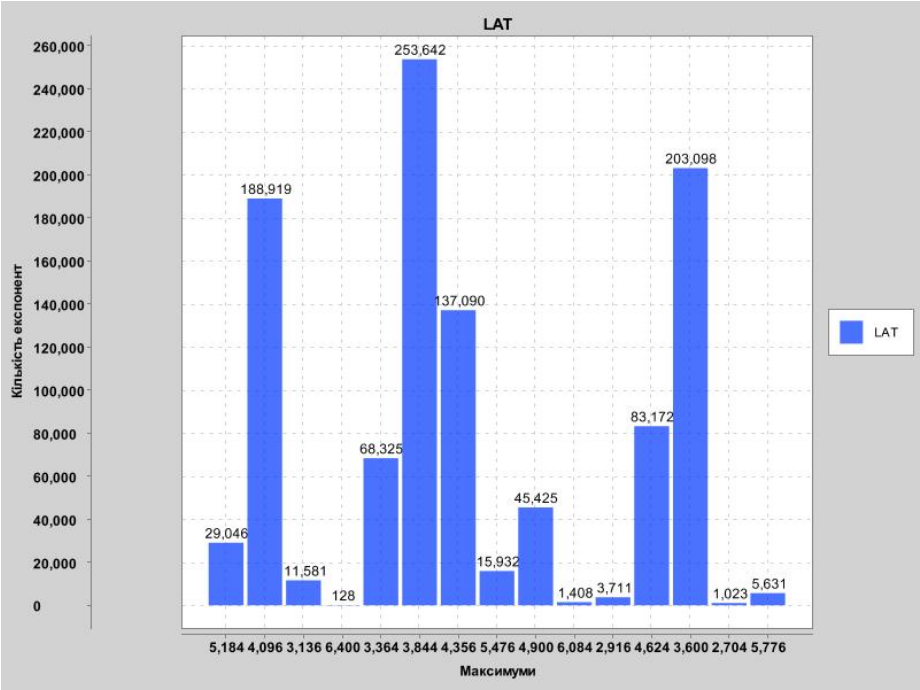

Рисунок 2.11 – LAT^s для афінних по входу

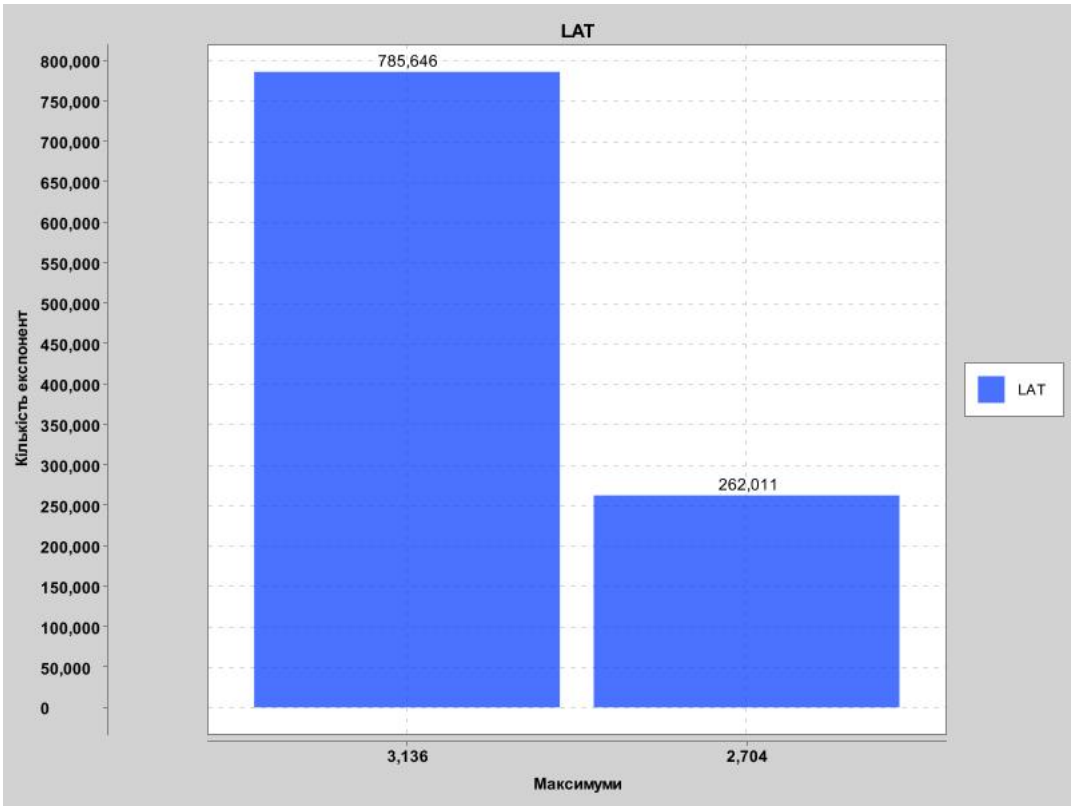

Рисунок 2.12 – LAT^sдля афінних по виходу

Таблиця 2.5 – Статистика максимумів для таблиць l^s

$max(l^s)$	Чисті экпоненти	
	Кількість	%
16	692,224	46.3%
48	802816	53.6%

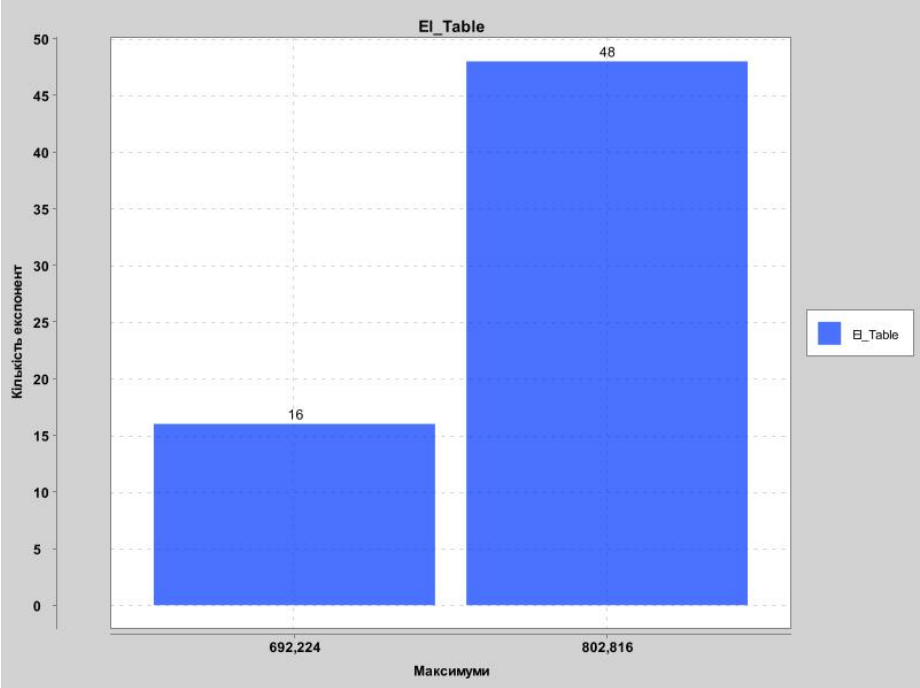

Рисунок 2.13 – l^s

Таблиця 2.6 – Статистика максимумів для таблиць Λ^s

$max(\lambda^s)$	Чисті экпоненти	
	Кількість	%
16	262,276	24,5%
32	262,144	24,49%
8	274,236	25,62%
8	271,820	25.39%

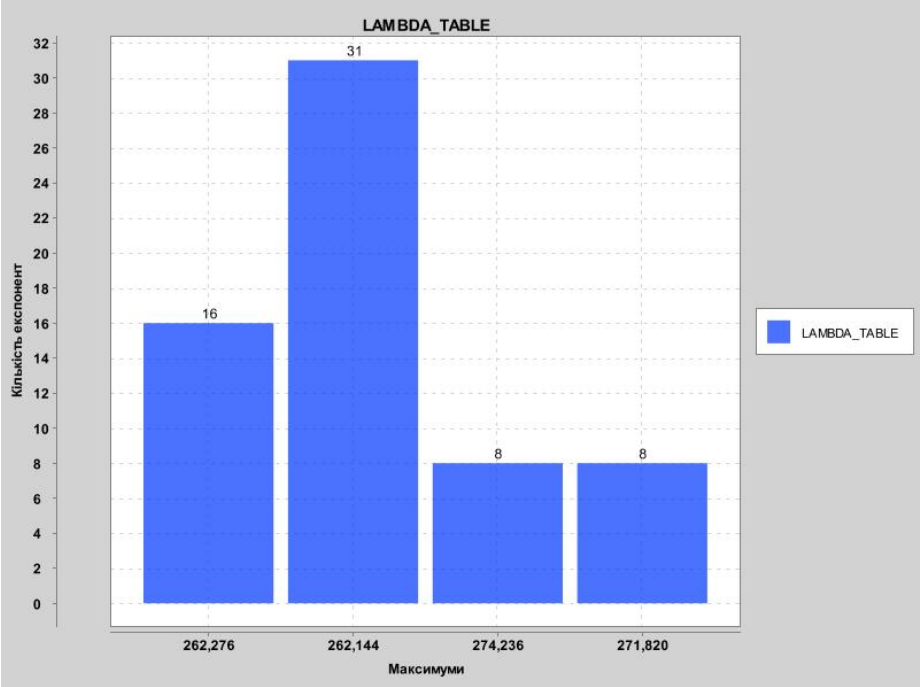

Рисунок 2.14 – Λ^s

Всі результати наведено на рисунках 2.1- 2.14 а також в таблицях 2.1-2.6 Нажаль для параметрів l^s та Λ^s не вдалось обрахувати відповідні статистики для афінного перетворення на вході та на виході експоненційного S-блоку, оскільки ці обрахунки не можливо виконати за розумний час на доступних автору обчислювальних потужностях. Складність таких обчислень $64^2 \cdot (2^8)^6 = 2^{60}$ операцій.

Проаналізувавши отримані дані, бачимо, що застосовуюння афінних перетвореннь на вході або на виході суттєво збільшує множину доступних S-блоків. При цьому частка мінімальних значень максимумів таблиць диференціального розподілу не зменшується. Крім того в деяких випадках з'являється невелика підмножина експонент с кращими диференціальними характеристиками.

Висновки до розділу 2

Таким чином, в цьому розділі наведено і доведено новий вид еквівалентності експоненційних S-блоків, що зберігає диференціальні імовірності $DP_{+,\oplus}$. Доведено можливість розширення множини доступних експоненційних перетворень для використання в криптографії. Також показано, що одночасне застосування афінного перетворення на вході та на виході експоненційного S-блоку не призводить до розширення цієї множини.

У ході проведення обчислень та досліджень програмним способом було знайдено 64 основи для експоненційного перетворення, що мають низькі диференціальні характеристики та максимальний алгебраїчний степінь. Для цих основ було показано, що множину експоненціальних S-блоків можна розширити шляхом афінних перетворень на вході або на виході зі збереженням низької диференціальної імовірності $DP_{+,\oplus}$, а за деяких умов вдалось навіть покращити інші криптографічні характеристики.

ВИСНОВКИ

У ході данної роботи був проведений аналіз роботи по експоненціальним S-блокам Сергія Агієвича та Андрія Афоненко. Саме на роботі цих вчених базувалися проведені мною дослідження. Також було опрацьовано роботи Л. Будагян та К.Карле. Ці роботи присвячені різним видам еквівалентності булевих функцій. Результати цих вчених дали поштовх для наших досліджень в області еквівалентності експоненційних перетворень.

Запропоновано і доведено новий вид еквівалентності експоненційних S-блоків, що зберігає диференціальні імовірності $DP_{+,\oplus}$. Доведено можливість розширення множини доступних експоненційних перетворень для використання в криптографії. Також показано, що одночасне застосування афінного перетворення на вході та на виході експоненційного S-блоку не призводить до розширення цієї множини.

У ході проведення обчислень та досліджень програмним способом було знайдено 64 основи для експоненційного перетворення, що мають низькі диференціальні характеристики та максимальний алгебраїчний степінь. Для цих основ було показано, що множину експоненціальних S-блоків можна розширити шляхом афінних перетворень на вході або на виході зі збереженням низької диференціальної імовірності $DP_{+,\oplus}$, а за деяких умов вдалось навіть покращити інші криптографічні характеристики.

БІБЛІОҐРАФІЯ

1. Логачев О.А. Сальников А.А. Ященко В.В. Булевы функции в теории кодирования и криптологии – М.: МЦНМО, 470с. — 2004.

2. Завадська Л.О. Савчук М.М. Математичні методи захисту інформації: курс лекцій. Частина 1. – Київ: НТУУ «КПІ» – Ч.1. – 128 с. — 2008.

3. В.М. Фомичев. Дискретная математика и криптология. – М.: ДИАЛОГ-МИФИ,– 400с. — 2003.

4. Budaghyan Lilya, Carlet Claude. On CCZ-equivalence and its use in secondary constructions of bent functions. — Cryptology ePrint Archive, Report 2009/042. — 2009. — https://eprint.iacr.org/2009/042.

5. Agievich Sergey, Afonenko Andrey. Exponential S-boxes. — Cryptology ePrint Archive, Report 2004/024. — 2004. — https://eprint.iacr.org/2004/024.

6. Knudsen Lars R. Robshaw Matthew. The Block Cipher Companion. – Springer – 279 pp. – ISBN 978-3-642-17342-4. — 2011.

7. Hawkes Philip, O'Connor Luke, Hawkes Philip. Asymptotic Bounds on Differential Probabilities. — 1998.

8. Alekseychuk A. N., Kovalchuk L. V. Towards a Theory of Security Evaluation for GOST-like Ciphers against Differential and Linear Cryptanalysis. — Cryptology ePrint Archive, Report 2011/489. — 2011. — https://eprint.iacr.org/2011/489.

9. Budaghyan Lilya, Carlet Claude. CCZ-equivalence and Boolean functions. — Cryptology ePrint Archive, Report 2009/063. — 2009. — https://eprint.iacr.org/2009/063.

ДОДАТОК А ТЕКСТИ ПРОГРАМ

```
package ua.kpi.pti.diploma;

import ua.kpi.pti.diploma.charts.BarChartForTables;
import ua.kpi.pti.diploma.charts.CustomBarChart;
import ua.kpi.pti.diploma.tables.*;

import java.util.ArrayList;
import java.util.List;

public class App {
    public static void main(String[] args) {
        AgafonovCriteriaFilter agafonovCriteriaFilter = new AgafonovCriteriaFilter();
        List<TableProvider> tableProvidersPool = new ArrayList<>();
        Type type = Type.EXTENDED;
        List<Integer> alpas = agafonovCriteriaFilter
                .filterByOptimalDifferentialCharacteristics(AgafonovCriteriaFilter.findAllPrimitiveEleme
        alpas = agafonovCriteriaFilter.filterByMaximumAlgebraicDegree(alpas);
        tableProvidersPool.add(new DdtXorXor());
        tableProvidersPool.add(new DdtXorPlus());
        tableProvidersPool.add(new DdtPlusPlus());
        tableProvidersPool.add(new LAT());
        tableProvidersPool.add(new ElTable());
        tableProvidersPool.add(new LambdaTable());
        List<Integer> finalAlpas = alpas;
        tableProvidersPool.forEach(tableProvider -> {
            CustomBarChart chart = new BarChartForTables(tableProvider.getTableName());
            chart.printChart(tableProvider.calculateStatistics(finalAlpas,
                type)).saveChart(type.toString());
        });
    }
```

```
package ua.kpi.pti.diploma;

import org.bouncycastle.pqc.math.linearalgebra.GF2mField;
```

```
4  import ua.kpi.pti.diploma.utils.Utils;
5
6  import java.util.ArrayList;
7  import java.util.List;
8  import java.util.stream.IntStream;
9
10 import static java.lang.Math.pow;
11 import static ua.kpi.pti.diploma.utils.Constants.*;
12
13 public class AgafonovCriteriaFilter {
14     private final int k = (int) pow(P, M - 1);
15
16     public static List<Integer> findAllPrimitiveElementsOfField() {
17         GF2mField gf2mField = new GF2mField(M);
18         ArrayList<Integer> primitiveElements = new ArrayList<>();
19
20         for (int i = 1; i < Q - 1; i++) {
21             int current = i;
22             boolean isPrimitive = Utils.decomposeNumberToPrimeMultipliers(Q - 1).stream()
23                     .allMatch(elem -> gf2mField.exp(current, (Q - 1) / elem) != 1);
24             if (isPrimitive) {
25                 primitiveElements.add(i);
26             }
27         }
28         return primitiveElements;
29     }
30
31     public List<Integer> filterByOptimalDifferentialCharacteristics(List<Integer> primitiveBases) {
       ↪  //means characteristic that described by Agievich and Afonenko
32         List<Integer> filteredExponents = new ArrayList<>();
33         for (Integer alpha : primitiveBases) {
34
35             boolean isAcceptable = IntStream.rangeClosed(1, k - 1)
36                     .allMatch(exponent -> FIELD.add(FIELD.add(FIELD.exp(alpha, k), FIELD.exp(alpha,
                       ↪  exponent)), 1) != 0);
37             if (isAcceptable) {
38                 filteredExponents.add(alpha);
39             }
40
```

```
        }
        return filteredExponents;
    }

    public List<Integer> filterByMaximumAlgebraicDegree(List<Integer> alphas) {
        List<Integer> filteredAlphas = new ArrayList<>();
        for (Integer alpha : alphas) {
            int alphaForBasis = FIELD.mult(alpha, FIELD.inverse(FIELD.add(1, alpha)));
            List<Integer> vectors = new ArrayList<>();
            for (int i = 0; i < M; i++) {
                vectors.add(FIELD.exp(alphaForBasis, (int) pow(2, i)));
            }
            if (Utils.rankIsNotZero(vectors)) {
                filteredAlphas.add(alpha);
            }
        }
        return filteredAlphas;
    }
}
```

```
package ua.kpi.pti.diploma;

public enum Type {
    AFFINE_ON_ENTER,AFFINE_ON_EXIT, USUAL;
}

package ua.kpi.pti.diploma.tables;

import ua.kpi.pti.diploma.Type;
import ua.kpi.pti.diploma.extender.SboxExtender;
import ua.kpi.pti.diploma.tables.threads.TableThread;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import static ua.kpi.pti.diploma.extender.SboxExtender.aList;
```

```
19  import static ua.kpi.pti.diploma.utils.Constants.Q;
20  import static ua.kpi.pti.diploma.utils.Constants.sBoxUsusal;
21
22  public abstract class TableProvider {
23      protected String tableName;
24      protected List<TableThread> threadPool;
25      private SboxExtender ex = new SboxExtender();
26
27      public int[][] getTable(int[] sbox) {
28          int[][] table = new int[Q][Q];
29          calculateWithMultiThreads(getThreadPool(table, sbox));
30          return table;
31      }
32
33      protected abstract void calculate(int[][] table, int sbox[]);
34
35      int maxInTable(int[][] matrix) {
36          int max = 0;
37          for (int i = 1; i < matrix.length; i++) {
38              for (int j = 1; j < matrix[i].length; j++) {
39                  if (matrix[i][j] > max) {
40                      max = matrix[i][j];
41                  }
42              }
43          }
44          return max;
45      }
46
47      void calculateWithMultiThreads(List<TableThread> threadPool) {
48          threadPool.forEach(TableThread::start);
49
50          for (TableThread thread : threadPool) {
51              try {
52                  thread.join();
53              } catch (InterruptedException e) {
54                  e.printStackTrace();
55              }
56          }
57      }
```

```
58
59      private void multithread(Map<Integer, Integer> ststistics, int[] sbox) {
60          int ddt[][] = new int[Q][Q];
61          calculate(ddt, sbox);
62          int max = maxInTable(ddt);
63          ststistics.putIfAbsent(max, 0);
64          ststistics.put(max, ststistics.get(max) + 1);
65      }
66
67      public Map<Integer, Integer> calculateStatistics(List<Integer> basises, Type type) {
68          Map<Integer, Integer> statistics = new ConcurrentHashMap<>();
69          List<Thread> threadPool = new ArrayList<>();
70          ex.getExtendedSBox(type);
71          if (type == Type.USUAL) {
72
73              for (int i = 0; i < basises.size(); i = i + 8) {
74
75                  int finalI = i;
76                  threadPool.add(new Thread(() -> multithread(statistics,
                    ↪  sBoxUsusal[basises.get(finalI)])));
77                  threadPool.add(new Thread(() -> multithread(statistics,
                    ↪  sBoxUsusal[basises.get(finalI + 1)])));
78                  threadPool.add(new Thread(() -> multithread(statistics,
                    ↪  sBoxUsusal[basises.get(finalI + 2)])));
79                  threadPool.add(new Thread(() -> multithread(statistics,
                    ↪  sBoxUsusal[basises.get(finalI + 3)])));
80                  threadPool.add(new Thread(() -> multithread(statistics,
                    ↪  sBoxUsusal[basises.get(finalI + 4)])));
81                  threadPool.add(new Thread(() -> multithread(statistics,
                    ↪  sBoxUsusal[basises.get(finalI + 5)])));
82                  threadPool.add(new Thread(() -> multithread(statistics,
                    ↪  sBoxUsusal[basises.get(finalI + 6)])));
83                  threadPool.add(new Thread(() -> multithread(statistics,
                    ↪  sBoxUsusal[basises.get(finalI + 7)])));
84
85
86              }
87              threadPool.forEach(Thread::start);
88              for (Thread t : threadPool) {
```

```
89                  try {
90                      t.join();
91                  } catch (InterruptedException e) {
92                      e.printStackTrace();
93                  }
94              }
95          }
96          threadPool = new ArrayList<>();
97
98          if (type == Type.AFFINE_ON_ENTER || type == Type.AFFINE_ON_EXIT) {
99              for (int b = 0; b < Q; b++) {
100                 for (int a : aList) {
101                     for (int i = 0; i < basises.size(); i++) {
102                         int finalI = i;
103                         int finalB = b;
104                         threadPool.add(new Thread(() -> multithread(statistics,
                            ↪  ex.getExtendedSBox(type)[finalB][a][basises.get(finalI)])));
105
106                     }
107                 }
108             }
109             for (int j = 0; j < threadPool.size(); j++) {
110                 System.out.println((double) j / threadPool.size() * 100);
111                 List<Thread> pool = new ArrayList<>();
112                 threadPool.get(j).start();
113                 pool.add(threadPool.get(j));
114                 if ((j + 1) % 11 == 0) {
115                     for (Thread p : pool) {
116                         try {
117                             p.join();
118                         } catch (InterruptedException e) {
119                             e.printStackTrace();
120                         }
121                     }
122                 }
123             }
124         }
125         return statistics;
126     }
```

```
127     public String getTableName() {
128         return tableName;
129     }
130     protected abstract List<TableThread> getThreadPool(int[][] table, int[] sbox);
131 }
```

```
1  package ua.kpi.pti.diploma.tables;
2
3  import ua.kpi.pti.diploma.Type;
4  import ua.kpi.pti.diploma.tables.threads.DdtXorXorThread;
5  import ua.kpi.pti.diploma.tables.threads.TableThread;
6  import ua.kpi.pti.diploma.utils.MatrixToCSVPrinter;
7
8  import java.io.IOException;
9  import java.util.ArrayList;
10 import java.util.HashMap;
11 import java.util.List;
12 import java.util.Map;
13
14 import static ua.kpi.pti.diploma.extender.SboxExtender.aList;
15 import static ua.kpi.pti.diploma.utils.Constants.*;
16
17 public class DdtXorXor extends TableProvider {
18     String tableName = "DDT_XOR_XOR";
19
20
21     @Override
22     protected void calculate(int[][] table, int[] sbox) {
23         for (int alpha = 0; alpha < Q; alpha++) {
24             for (int x = 0; x < Q; x++) {
25                 int out = sbox[x ^ alpha] ^ sbox[x];
26                 table[alpha][out] = (table[alpha][out] + 1) &0xFF;
27             }
28         }
29     }
30
31     @Override
32     public List<TableThread> getThreadPool(int[][] table, int[] sbox) {
```

```
33          threadPool = new ArrayList<>();
34
35          for (int i = 0; i < CORES; i++) {
36              threadPool.add(new DdtXorXorThread(table, i * Q / CORES, (i + 1) * Q / CORES, sbox));
37          }
38          return this.threadPool;
39      }
40
41      @Override
42      public String getTableName() {
43          return this.tableName;
44      }
45  }
```

```
1   package ua.kpi.pti.diploma.tables;
2
3   import ua.kpi.pti.diploma.Type;
4   import ua.kpi.pti.diploma.tables.threads.DdtXorPlusThread;
5   import ua.kpi.pti.diploma.tables.threads.TableThread;
6   import ua.kpi.pti.diploma.utils.MatrixToCSVPrinter;
7
8   import java.io.IOException;
9   import java.util.ArrayList;
10  import java.util.HashMap;
11  import java.util.List;
12  import java.util.Map;
13
14  import static ua.kpi.pti.diploma.extender.SboxExtender.aList;
15  import static ua.kpi.pti.diploma.utils.Constants.*;
16
17  public class DdtXorPlus extends TableProvider {
18      String tableName = "DDT_XOR_PLUS";
19
20      @Override
21      protected void calculate(int[][] table, int[] sbox) {
22          for (int alpha = 0; alpha < Q; alpha++) {
23              for (int x = 0; x < Q; x++) {
24                  int out = (sbox[x ^ alpha] - sbox[x] + Q) & 0xFF;
```

```
                table[alpha][out] = (table[alpha][out] + 1) & 0xFF;
            }
        }
    }

    public String getTableName() {
        return tableName;
    }

    @Override
    public List<TableThread> getThreadPool(int[][] table, int[] sbox) {
        threadPool = new ArrayList<>();

        for (int i = 0; i < CORES; i++) {
            threadPool.add(new DdtXorPlusThread(table, i * Q / CORES, (i + 1) * Q / CORES, sbox));
        }
        return this.threadPool;
    }
}
```

```
package ua.kpi.pti.diploma.tables;

import ua.kpi.pti.diploma.Type;
import ua.kpi.pti.diploma.tables.threads.DdtPlusPlusThread;
import ua.kpi.pti.diploma.tables.threads.TableThread;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import static ua.kpi.pti.diploma.extender.SboxExtender.aList;
import static ua.kpi.pti.diploma.utils.Constants.*;

public class DdtPlusPlus extends TableProvider {
    String tableName = "DDT_PLUS_PLUS";


```

```
19      @Override
20      protected void calculate(int[][] table, int[] sbox) {
21          for (int alpha = 0; alpha < Q; alpha++) {
22              for (int x = 0; x < Q; x++) {
23                  int out = (sbox[(x+ alpha + Q) % Q] - sbox[x] + Q) % Q;
24                  table[alpha][out] = (table[alpha][out] + 1) % Q;
25              }
26          }
27      }
28
29      public String getTableName() {
30          return tableName;
31      }
32
33      @Override
34      public List<TableThread> getThreadPool(int[][] table, int[] sbox) {
35          threadPool = new ArrayList<>();
36
37          for (int i = 0; i < CORES; i++) {
38              threadPool.add(new DdtPlusPlusThread(table, i * Q / CORES, (i + 1) * Q / CORES, sbox));
39          }
40          return this.threadPool;
41      }
42  }
```

```
1   package ua.kpi.pti.diploma.tables;
2
3   import ua.kpi.pti.diploma.Type;
4   import ua.kpi.pti.diploma.tables.threads.DdtPlusPlusThread;
5   import ua.kpi.pti.diploma.tables.threads.TableThread;
6
7   import java.util.ArrayList;
8   import java.util.HashMap;
9   import java.util.List;
10  import java.util.Map;
11
12  import static ua.kpi.pti.diploma.extender.SboxExtender.aList;
13  import static ua.kpi.pti.diploma.utils.Constants.*;
```

```
14
15  public class DdtPlusPlus extends TableProvider {
16      String tableName = "DDT_PLUS_PLUS";
17
18
19      @Override
20      protected void calculate(int[][] table, int[] sbox) {
21          for (int alpha = 0; alpha < Q; alpha++) {
22              for (int x = 0; x < Q; x++) {
23                  int out = (sbox[(x+ alpha + Q) % Q] - sbox[x] + Q) % Q;
24                  table[alpha][out] = (table[alpha][out] + 1) % Q;
25              }
26          }
27      }
28
29      public String getTableName() {
30          return tableName;
31      }
32
33      @Override
34      public List<TableThread> getThreadPool(int[][] table, int[] sbox) {
35          threadPool = new ArrayList<>();
36
37          for (int i = 0; i < CORES; i++) {
38              threadPool.add(new DdtPlusPlusThread(table, i * Q / CORES, (i + 1) * Q / CORES, sbox));
39          }
40          return this.threadPool;
41      }
42  }
```

```
1   package ua.kpi.pti.diploma.tables;
2
3   import ua.kpi.pti.diploma.Type;
4   import ua.kpi.pti.diploma.tables.threads.ElThread;
5   import ua.kpi.pti.diploma.tables.threads.TableThread;
6   import ua.kpi.pti.diploma.utils.MatrixToCSVPrinter;
7
8   import java.io.IOException;
```

```
9  import java.util.ArrayList;
10 import java.util.HashMap;
11 import java.util.List;
12 import java.util.Map;
13
14 import static ua.kpi.pti.diploma.extender.SboxExtender.aList;
15 import static ua.kpi.pti.diploma.utils.Constants.*;
16 import static ua.kpi.pti.diploma.utils.Utils.scalarMultiplication;
17
18 public class ElTable extends TableProvider {
19     String tableName = "El_Table";
20
21
22     @Override
23     protected void calculate(int[][] table, int[] sbox) {
24         for (int alpha = 0; alpha < Q; alpha++) {
25             for (int beta = 0; beta < Q; beta++) {
26                 int sum1 = 0;
27
28                 for (int k = 0; k < Q; k++) {
29                     int sum2 = 0;
30
31                     for (int x = 0; x < Q; x++) {
32                         int sBoxOut = sbox[x];
33                         if((scalarMultiplication(alpha, x) ^ scalarMultiplication(beta,
                            ↪  sBoxOut))==1) {
34                             sum2 ++;
35                         }
36                     }
37                     sum1 += (Q - 2*(sum2))*(Q - 2*(sum2));
38                 }
39                 table[alpha][beta] = sum1;
40             }
41         }
42     }
43
44     @Override
45     public List<TableThread> getThreadPool(int[][] table, int[] sbox) {
46         threadPool = new ArrayList<>();
```

```
47
48          for (int i = 0; i < CORES; i++) {
49              threadPool.add(new ElThread(table, i * Q / CORES, (i + 1) * Q / CORES, sbox));
50          }
51
52          return this.threadPool;
53      }
54      public String getTableName() {
55          return this.tableName;
56      }
57  }
```

```
1   package ua.kpi.pti.diploma.tables;
2
3   import ua.kpi.pti.diploma.Type;
4   import ua.kpi.pti.diploma.tables.threads.LambdaThread;
5   import ua.kpi.pti.diploma.tables.threads.TableThread;
6   import ua.kpi.pti.diploma.utils.MatrixToCSVPrinter;
7
8   import java.io.IOException;
9   import java.util.ArrayList;
10  import java.util.HashMap;
11  import java.util.List;
12  import java.util.Map;
13
14  import static java.lang.Math.abs;
15  import static ua.kpi.pti.diploma.extender.SboxExtender.*;
16  import static ua.kpi.pti.diploma.utils.Constants.*;
17  import static ua.kpi.pti.diploma.utils.Utils.scalarMultiplication;
18
19  public class LambdaTable extends TableProvider {
20      String tableName = "LAMBDA_TABLE";
21
22      @Override
23      protected void calculate(int[][] table, int[] sbox) {
24          int n1, n2;
25          int sum, sumK;
26          for (int alpha = 0; alpha < Q; alpha++) {
```

```
27              for (int beta = 0; beta < Q; beta++) {
28                  sumK = 0;
29                  for (int k = 0; k < Q; k++) {
30                      n1 = 0;
31                      n2 = 0;
32                      for (int x = 0; x < Q - k; x++) {
33                          n1 = n1 + (scalarMultiplication(alpha, x) ^ scalarMultiplication(beta,
                             ↪  sbox[(x + k) & 0xFF]));
34                      }
35                      for (int x = Q - k; x < Q; x++) {
36                          n2 = n2 + (scalarMultiplication(alpha, x) ^ scalarMultiplication(beta,
                             ↪  sbox[(x+ k) & 0xFF]));
37                      }
38                      sum = abs(Q - k - 2 * n1) + abs(k - 2 * n2);
39                      sumK = sumK + sum * sum;
40
41                  }
42                  table[alpha][beta] = sumK;
43              }
44          }
45
46      }
47
48      @Override
49      public List<TableThread> getThreadPool(int[][] table, int[] sbox) {
50          threadPool = new ArrayList<>();
51
52          for (int i = 0; i < 7; i++) {
53              threadPool.add(new LambdaThread(table, i * Q / 7, (i + 1) * Q / 7, sbox));
54          }
55
56          return this.threadPool;
57      }
58
59      public String getTableName() {
60          return this.tableName;
61      }
```

```java
package ua.kpi.pti.diploma.tables;

import ua.kpi.pti.diploma.Type;
import ua.kpi.pti.diploma.tables.threads.LatThread;
import ua.kpi.pti.diploma.tables.threads.TableThread;
import ua.kpi.pti.diploma.utils.MatrixToCSVPrinter;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import static ua.kpi.pti.diploma.extender.SboxExtender.aList;
import static ua.kpi.pti.diploma.utils.Constants.*;
import static ua.kpi.pti.diploma.utils.Utils.scalarMultiplication;

public class LAT extends TableProvider {
    String tableName = "LAT";

    @Override
    protected void calculate(int[][] table, int[] sbox) {
        for (int alpha = 0; alpha < Q; alpha++) {
            for (int beta = 0; beta < Q; beta++) {
                int sum = 0;
                for (int x = 0; x < Q; x++) {
                    int sBoxOut = sbox[x];
                    if((scalarMultiplication(alpha, x) ^ scalarMultiplication(beta, sBoxOut))==1) {
                        sum ++;
                    }
                }
                table[alpha][beta] = (Q - 2*(sum))*(Q - 2*(sum));
            }
        }
    }

    public String getTableName() {
        return this.tableName;
```

```
    }

    @Override
    public List<TableThread> getThreadPool(int[][] table, int[] sbox) {
        threadPool = new ArrayList<>();

        for (int i = 0; i < CORES; i++) {
            threadPool.add(new LatThread(table, i * Q / CORES, (i + 1) * Q / CORES, sbox));
        }
        return this.threadPool;
    }
}
```

```
package ua.kpi.pti.diploma.tables.threads;

import static ua.kpi.pti.diploma.utils.Constants.Q;

public class DdtXorXorThread extends TableThread {


    public DdtXorXorThread(int[][] table, int startAlpha, int endAlpha, int[] sboxTable) {
        super(table, startAlpha, endAlpha,sboxTable);
    }

    @Override
    public void run() {
        for (int alpha = startAlpha; alpha < endAlpha; alpha++) {
            for (int x = 0; x < Q; x++) {
                int out = sbox[x ^ alpha] ^ sbox[x];
                table[alpha][out] = (table[alpha][out] + 1) &0xFF;
            }
        }
    }
}
```

```
package ua.kpi.pti.diploma.tables.threads;

import static ua.kpi.pti.diploma.utils.Constants.Q;
```

```
4
5  public class DdtXorPlusThread extends TableThread {
6
7      public DdtXorPlusThread(int[][] lat, int startAlpha, int endAlpha, int[] sboxTable) {
8          super(lat, startAlpha, endAlpha,sboxTable);
9      }
10
11     @Override
12     public void run() {
13         for (int alpha = startAlpha; alpha < endAlpha; alpha++) {
14             for (int x = 0; x < Q; x++) {
15                 int out = (sbox[x ^ alpha] - sbox[x] + Q) & 0xFF;
16                 table[alpha][out] = (table[alpha][out] + 1) & 0xFF;
17             }
18         }
19     }
20 }
```

```
1   package ua.kpi.pti.diploma.tables.threads;
2
3  import static ua.kpi.pti.diploma.utils.Constants.Q;
4
5  public class DdtPlusPlusThread extends TableThread {
6
7      public DdtPlusPlusThread(int[][] ddt, int startAlpha, int endAlpha,int[]sboxTable) {
8          super(ddt, startAlpha, endAlpha, sboxTable);
9   }
10
11     @Override
12     public void run() {
13         for (int alpha = startAlpha; alpha < endAlpha; alpha++) {
14             for (int x = 0; x < Q; x++) {
15                 int out = (sbox[(x+ alpha + Q) % Q] - sbox[x] + Q) % Q;
16                 table[alpha][out] = (table[alpha][out] + 1) % Q;
17             }
18         }
19     }
20 }
```

```java
package ua.kpi.pti.diploma.tables.threads;

import static java.lang.Math.abs;
import static ua.kpi.pti.diploma.utils.Constants.Q;
import static ua.kpi.pti.diploma.utils.Utils.scalarMultiplication;

public class LambdaThread extends TableThread {


    public LambdaThread(int[][] lat, int startAlpha, int endAlpha, int[] sboxTable) {
        super(lat, startAlpha, endAlpha, sboxTable);
    }

    @Override
    public void run() {

        int n1, n2;
        int sum, sumK;
        for (int alpha = startAlpha; alpha < endAlpha; alpha++) {
            for (int beta = 0; beta < Q; beta++) {
                sumK = 0;
                for (int k = 0; k < Q; k++) {
                    n1 = 0;
                    n2 = 0;
                    for (int x = 0; x < Q - k; x++) {
                        n1 = n1 + (scalarMultiplication(alpha, x) ^ scalarMultiplication(beta,
                        ↪  sbox[(x + k) & 0xFF]));
                    }
                    for (int x = Q - k; x < Q; x++) {
                        n2 = n2 + (scalarMultiplication(alpha, x) ^ scalarMultiplication(beta,
                        ↪  sbox[(x+ k) & 0xFF]));
                    }
                    sum = abs(Q - k - 2 * n1) + abs(k - 2 * n2);
                    sumK = sumK + sum * sum;
                }
                table[alpha][beta] = sumK;
            }
        }
```

```
37
38      }
39  }
```

```
1   package ua.kpi.pti.diploma.tables.threads;
2
3   import static ua.kpi.pti.diploma.utils.Constants.Q;
4   import static ua.kpi.pti.diploma.utils.Utils.scalarMultiplication;
5
6   public class LatThread extends TableThread {
7
8
9       public LatThread(int[][] lat, int startAlpha, int endAlpha, int[]sBox) {
10          super(lat, startAlpha, endAlpha, sBox);
11      }
12
13      @Override
14      public void run() {
15          for (int alpha = startAlpha; alpha < endAlpha; alpha++) {
16              for (int beta = 0; beta < Q; beta++) {
17                  int sum = 0;
18                  for (int x = 0; x < Q; x++) {
19                      int sBoxOut = sbox[x];
20                      if((scalarMultiplication(alpha, x) ^ scalarMultiplication(beta, sBoxOut))==1) {
21                          sum ++;
22                      }
23                  }
24                  table[alpha][beta] = (Q - 2*(sum))*(Q - 2*(sum));
25              }
26          }
27      }
28  }
```

```
1   package ua.kpi.pti.diploma.tables.threads;
2
3   public abstract class TableThread extends Thread {
4       protected int[] sbox;
5       protected  int[][] table;
```

```
6       protected  int startAlpha;
7       protected  int endAlpha;
8
9       public TableThread(int[][] table, int startAlpha, int endAlpha,  int[]sBox) {
10          this.table = table;
11          this.startAlpha = startAlpha;
12          this.endAlpha = endAlpha;
13          this.sbox = sBox;
14      }
15  }
```

```
1   package ua.kpi.pti.diploma.utils;
2
3   import org.bouncycastle.pqc.math.linearalgebra.GF2mField;
4
5   public abstract class Constants {
6       public static final int P = 2;//characteristic of field
7       public static final int M = 8; //extension
8       public static final int Q = (int) Math.pow(P, M);
9       public static final GF2mField FIELD = new GF2mField(M);
10      public static final String PATH_TO_XOR_XOR_FOLDER = ".\\DDT_XOR_XOR\\";
11      public static final String PATH_TO_XOR_PLUS_FOLDER = ".\\DDT_XOR_PLUS\\";
12      public static final String PATH_TO_PLUS_PLUS_FOLDER = ".\\DDT_PLUS_PLUS\\";
13      public static final String PATH_TO_LAT = ".\\LAT\\";
14      public static final String PATH_TO_EL_TABLE = ".\\EL_TABLE\\";
15      public static final String PATH_TO_LAMBDA_TABLE = ".\\LAMBDA_TABLE\\";
16      public static final int CORES = Runtime.getRuntime().availableProcessors();
17
18      public static int[][] sBoxUsusal = new int[Q][Q];
19      static {
20          for (int base = 0; base < Q; base++) {
21
22              sBoxUsusal[base][0]=0; //S(0) = 0
23              for (int power = 1; power < Q; power++) {
24                  sBoxUsusal[base][power]=FIELD.exp(base, power);
25              }
26          }
27      }
```

```
28
29      public static int[] AR_WEIGHT = {
30              0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4,
31              1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
32              1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
33              2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
34              1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
35              2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
36              2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
37              3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
38              1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
39              2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
40              2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
41              3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
42              2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
43              3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
44              3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
45              4, 5, 5, 6, 5, 6, 6, 7, 5, 6, 6, 7, 6, 7, 7, 8};
46  }
```

```
1   package ua.kpi.pti.diploma.utils;
2
3   import java.util.HashSet;
4   import java.util.List;
5   import java.util.Set;
6
7   import static ua.kpi.pti.diploma.utils.Constants.AR_WEIGHT;
8
9   public class Utils {
10
11      public static Set<Integer> decomposeNumberToPrimeMultipliers(int number) {
12          Set<Integer> result = new HashSet<Integer>();
13          int div = 2;
14
15          while (number > 1) {
16              while (number % div == 0) {
17                  result.add(div);
18                  number = number / div;
```

```
            }
            div++;
        }
        return result;
    }

    public static boolean rankIsNotZero(List<Integer> vectors) {
        int s = vectors.get(0);
        for (int j = 0; j < vectors.size(); j++) {
            if (j != 0) {
                s = s ^ vectors.get(j);
            }
        }
        return s != 0;
    }

    public static Integer scalarMultiplication(int a, int b) {
        return AR_WEIGHT[a & b] & 1;
    }


}
```

```
package ua.kpi.pti.diploma.extender;

import ua.kpi.pti.diploma.Type;

import java.util.ArrayList;
import java.util.List;


import static org.apache.commons.math3.util.ArithmeticUtils.gcd;
import static ua.kpi.pti.diploma.utils.Constants.*;
import static ua.kpi.pti.diploma.utils.Constants.CORES;

public class SboxExtender {
    public static List<Integer> aList;

```

```
16      static {
17          aList = new ArrayList<>();
18          for (int a = 1; a < Q; a++) {
19              if (gcd(a, Q) == 1 && gcd(a, Q - 1) != 1) {
20                  aList.add(a);
21              }
22          }
23      }
24
25      private static int[][][][] extendedSBox;
26
27      public int[][][][] getExtendedSBox(Type type) {
28          ArrayList<Thread> pool = new ArrayList<>();
29
30          if (extendedSBox == null) {
31              extendedSBox = new int[Q][Q][Q][Q];
32              if (type == Type.AFFINE_ON_ENTER) {
33                  for (int i = 0; i < CORES; i++) {
34                      pool.add(new AffineOnEnterThread(i * Q / CORES, (i + 1) * Q / CORES,
                        ↪  extendedSBox));
35                  }
36              }
37              if (type == Type.AFFINE_ON_EXIT) {
38                  for (int i = 0; i < CORES; i++) {
39                      pool.add(new AfineOnExitThread(i * Q / CORES, (i + 1) * Q / CORES,
                        ↪  extendedSBox));
40                  }
41              }
42              threadWork(pool);
43          }
44          return extendedSBox;
45      }
46
47      private void threadWork(ArrayList<Thread> pool) {
48          pool.forEach(Thread::start);
49
50          for (Thread t : pool) {
51              try {
52                  t.join();
```

```
53              } catch (InterruptedException e) {
54                  e.printStackTrace();
55              }
56          }
57          System.out.println("FINISHED INIT");
58      }
59  }
```

```
1   package ua.kpi.pti.diploma.extender;
2
3   import static ua.kpi.pti.diploma.extender.SboxExtender.aList;
4   import static ua.kpi.pti.diploma.utils.Constants.FIELD;
5   import static ua.kpi.pti.diploma.utils.Constants.Q;
6
7   public class AfineOnExitThread extends Thread{
8       private int[][][][] extendedSBox;
9       private int start;
10      private int end;
11
12       public AfineOnExitThread(int start1, int end1, int[][][][] extendedSBox) {
13
14          this.start = start1;
15          this.end = end1;
16          this.extendedSBox = extendedSBox;
17      }
18
19      @Override
20      public void run() {
21          for (int b = start; b < end; b++) {
22              for (int a : aList) {
23                  for (int base = 0; base < Q; base++) {
24                      for (int power = 0; power < Q; power++) {
25                          extendedSBox[b][a][base][power] = FIELD.mult(a,FIELD.exp(base, power))^b;
26                      }
27                  }
28              }
29          }
30      }
```

```
}
```

```
package ua.kpi.pti.diploma.extender;

import java.util.List;

import static ua.kpi.pti.diploma.extender.SboxExtender.aList;
import static ua.kpi.pti.diploma.utils.Constants.FIELD;
import static ua.kpi.pti.diploma.utils.Constants.Q;

public class AffineOnEnterThread extends Thread {
    private int[][][][] extendedSBox;
    private int start;
    private int end;

    AffineOnEnterThread(int start1, int end1, int[][][][] extendedSBox) {

        this.start = start1;
        this.end = end1;
        this.extendedSBox = extendedSBox;
    }

    @Override
    public void run() {
        for (int b = start; b < end; b++) {
            for (int a:aList) {
                for (int base = 0; base < Q; base++) {
                    for (int power = 0; power < Q; power++) {
                        int x = (a * power + b) & 0xFF;
                        extendedSBox[b][a][base][power] = FIELD.exp(base, x);
                    }
                }
            }
        }
    }
}
```