Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп'ютерних наук

Спеціальність 125 «Кібербезпека»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

Зав.кафедрою БІСТ

Сергій РАССОМАХІН

_____

« » 2022 р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Оптимізація цільових функції у евристичних алгоритмів пошуку високонелінійних S-блоків»

оцінка « »

Голова ЕК

Доценко С.І. _____

Керівник доцент ЗВО, к.т.н.

Полуяненко Микола Олександрович.

Рецензент професор ЗВО, д.т.н.

Кузнецов Олександр Олександрович.

Виконавець : студент групи КБ-61

Попов Юрій Дмитрович

Харків – 2022

РЕФЕРАТ

Дипломну роботу виконано на 74 аркушах, вона містить перелік посилань на використані джерела з 18 найменувань. У роботі наведено 24 рисунки та 27 таблиць.

Метою даної дипломної роботи є вивчення існуючих рішень з формування високонелінійних S-боксів, розробка та дослідження алгоритму для програмного забезпечення формування високонелінійних бієктивних S-боксів.

У роботі проведено аналіз існуючих досліджень та алгоритмів формування S-боксів з використанням різних цінових функцій — GA генетичний алгоритм, цінова функція WHT, цінова функція DDT, цінова функція WCF, цінова функція WHS, цінова функція PCF. Виконано їх порівняння з погляду пошуку S-боксу з нелінійністю 104, за критерієм швидкодії пошуку та меншої кількості порахованих S-боксів. Для розв’язання задачі в роботі вибрано рішення генетичного алгоритму (GA) з різними комбінаціями цінових функцій.

Для обраного рішення побудована модель поетапного проведення дослідження. Розроблено програмне забезпечення, що реалізує обраний метод. Виконано практичне дослідження розробленого алгоритму з безліччю варіантів параметрів та приведено порівняльний результат.

Досліджено можливість використання двох кросоверів, а саме циклічний кросовер та кросовер PMX, у парі з генетичним алгоритмом та обраними оптимальними ціновими функціями. Проведено порівняння отриманих результатів роботи алгоритму без кросовера, та з двома вказаними кросоверами.

Всі результати наведені у вигляді порівняльних таблиць, рисунків та гістограм. Проведені спроби досягти нелінійності у значенні 106, використовуючи найоптимальніші цінові функції х досліджень.

Ключові слова: S-БОКС, НЕЛІНІЙНІСТЬ, ЦІНОВА ФУНКЦІЯ, ГЕНЕТИЧНИЙ, ПОПУЛЯЦІЯ, НАЩАДОК, МУТАЦІЯ, ПАРАМЕТРИ, БІЄКТИВНИЙ.

ABSTRACT

The thesis is completed on 74 sheets; it contains a list of references to used sources from 18 names. The work contains 24 figures and 27 tables.

The purpose of this thesis is to study the existing solutions for the formation of highly nonlinear S-boxes, to develop and research an algorithm for software for the formation of highly nonlinear bijective S-boxes.

The paper analyzes existing research and algorithms for the formation of S-boxes using various price functions – GA genetic algorithm, WHT cost function, DDT cost function, WCF cost function, WHS cost function, PCF cost function. They were compared from the point of view of S-box search with nonlinearity 104, according to the criterion of search speed and a smaller number of counted S-boxes. To solve the problem, the solution of the genetic algorithm (GA) with various combinations of price functions was chosen in the paper.

For the chosen solution, a model of a phased research is built. The software that implements the selected method has been developed. A practical study of the developed algorithm with many options of parameters was performed and a comparative result was given.

The possibility of using two crossovers, namely the cyclic crossover and the PMX crossover, in combination with a genetic algorithm and selected optimal price functions, was investigated. A comparison of the obtained results of the algorithm without a crossover and with the two specified crossovers was made.

Key words: S-BOX, NONLINEARITY, COST FUNCTION, GENETIC, POPULATION, DESCENDANT, MUTATION, PARAMETERS, BIJECTIVE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

S-бокс	–	складова шифрування з симетричними ключами, яка виконує підстановки
Нелінійність	–	термін, який використовується в статистиці для опису ситуації, коли не існує прямолінійних чи прямих зв'язків між незалежною змінною та залежною змінною
GA	–	Genetic Algorithm
GaT	–	Genetic and Tree
WHS	–	Walsh Hadamard Spectrum
WHT	–	Walsh–Hadamard Transform
WCF	–	Walsh-Hadamard Spec-trum in C
PCF	–	Picek's Cost Functions
PCP	–	Post-Crossover Population

ВСТУП

Усі сучасні блокові та потокові шифри мають один або кілька нелінійних елементів. Одним з найбільш використовуваних нелінійних вузлів є вузли підстановки (S-бокси). Існування цілого ряду криптоатак призвело до розробки критеріїв для протистояння таким атакам. Щоб бути криптографічно безпечним, S-бокс має бути звичайним, щоб запобігти видачі статистичної інформації відкритого тексту, коли зашифрований текст відомий. Криптографічно хороші S-бокси також повинні відповідати іншим критеріям. Одним із найважливіших критеріїв є так звана нелінійність. Критерій нелінійності забезпечує певний захист від добре відомих атак, таких як лінійний криптоаналіз та диференційний криптоаналіз. Відкритою задачею є метод, який буде швидко генерувати S-бокси з максимально можливою нелінійністю. Існує багато методів, за допомогою яких можна сформувати S-бокс з нелінійністю до значення 104 починаючи з деякого випадкового стану.

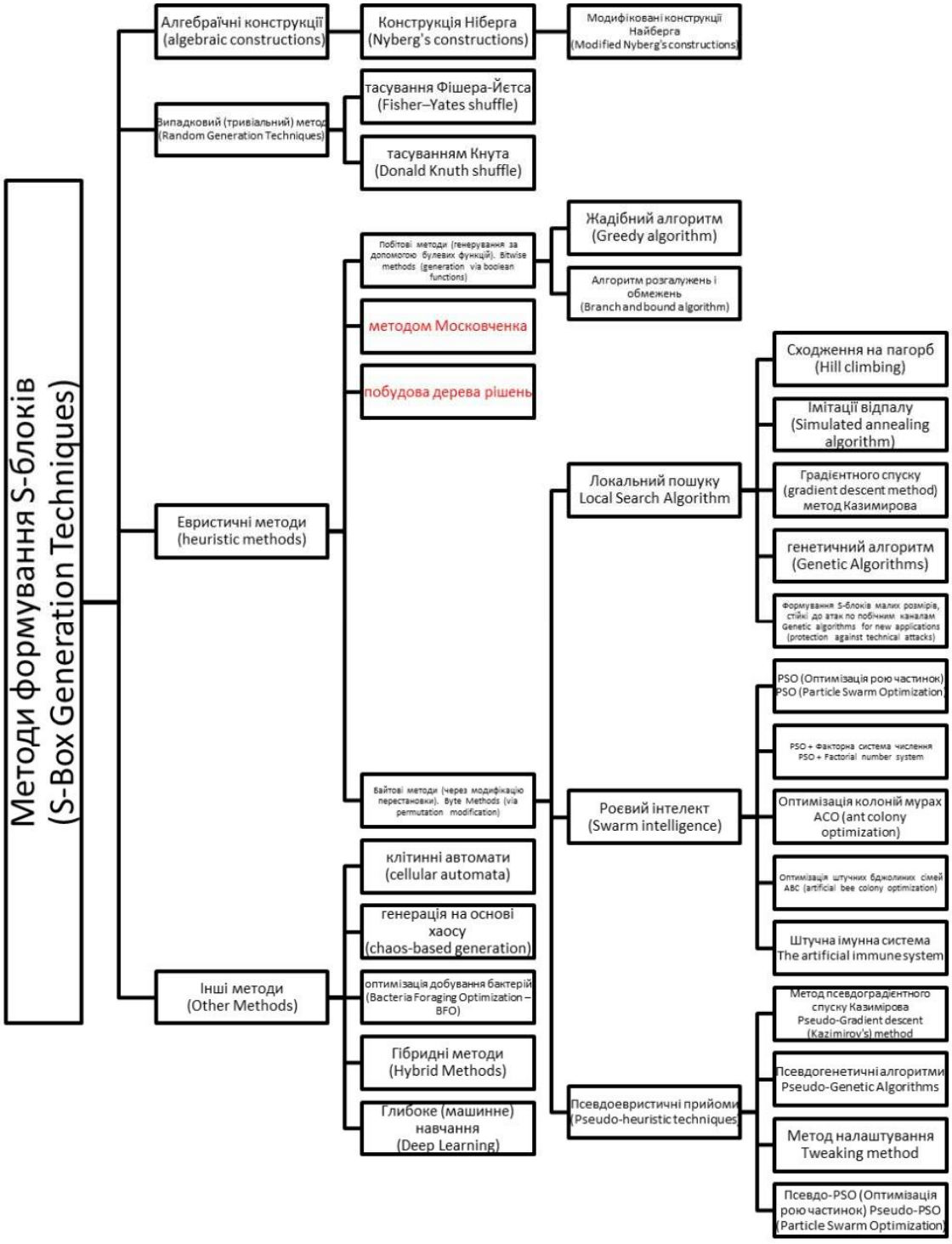

Рисунок 1 – Методи формування S-боксів.

З цих методів слід приділити увагу евристичним методам. У всіх евристичних методах обирається деяка цільова функція, що пов’язана з характеристикою S-боксу (нелінійністю). За допомогою обраного алгоритму намагається понизити значення цільової функції та, відповідно, наближуючи характеристики поточного S-боксу то бажаних значень. Успіх вказаного підходу дуже чутливий до обраної цільової функції, а отже до вибору її параметрів.

Покращення якості евристики (і, результатів, отриманих нею) є важливою метою, оскільки це допоможе підвищити довіру до таких методів. Відповідно до цього розробка евристик, які могли б досягти рішень, отриманих за допомогою алгебраїчних конструкцій, мала б значний вплив як для криптографії. Тому центральне питання цього дослідження полягає в тому, як покращити якість та швидкість отримання результатів, отриманих евристичними методами. Одним із досить поширених способів є дослідження різних евристик у надії знайти найбільш прийнятну.

У даній роботі досліджується можливість застосування досить простого методу пошуку – генетичного алгоритму (GA) разом з ціновими функціями *WHS, WHT, WCF, DDT, PCF*. Наводяться оптимізовані параметри досліджених цінових функцій. Також, наведено дослідження зі зміною параметрів самого алгоритму. Надається отримана середня кількість ітерації для успішного формування випадкових бієктивних S-боксів з нелінійністю 104.

У даній роботі поставлена мета дослідити простий та легкий у реалізації алгоритм пошуку – генетичний алгоритм (GA). В якості цінової функції обрано досить прості та зручні у реалізації функції вартості *WHS*, *WCF*, *WHT*, *DDT*, *PCF*. Проведено оптимізацію параметрів обраних цінових функцій під генетичний алгоритм (GA). Наведено отримані результати щодо можливості формування такою комбінацією високонелінійних S-боксів. Вказано на середньостатистичну кількість ітерації алгоритму пошуку та середній час на виконання одної ітерації,

що у подальшій роботі сприяє порівнянню інших методів та об’єктивному вибору найкращої комбінації алгоритм – цільова функція.

Метою цієї роботи є дослідження вказаних цільових функцій та вплив її параметрів на швидкість (кількість ітерацій) формування випадкових бієктивних S-боксів з нелінійністю 104.

1 ДОСЛІДЖЕННЯ ШЛЯХІВ ФОРМУВАННЯ S-БОКСІВ ОБҐРУНТУВАННЯ ВИБОРУ ГЕНЕТИЧНОГО АЛГОРИТМУ

1.1 Метод пошуку S-боксу

Генетичний алгоритм (англ. genetic algorithm) — це еволюційний алгоритм пошуку, що використовується для вирішення задач оптимізації і моделювання шляхом послідовного підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію.

Особливістю генетичного алгоритму є акцент на використання оператора «схрещення», який виконує операцію рекомбінацію рішень-кандидатів, роль якої аналогічна ролі схрещення в живій природі.

1.2 Опис алгоритму

Задача кодується таким чином, щоб її вирішення могло бути представлено в вигляді масиву подібного до інформації складу хромосоми. Цей масив часто називають саме так «хромосома». Випадковим чином в масиві створюється деяка кількість початкових елементів «осіб», або початкова популяція. Особи оцінюються з використанням функції допасованості (особливий тип цільової функції, який застосовують як порівняльний показник якості для підбивання підсумку того, наскільки близьким є задане конструктивне рішення до досягнення поставлених цілей), в результаті якої кожній особі присвоюється певне значення допасованості, яке визначає можливість виживання особи. Після цього з використанням отриманих значень допасованості вибираються особи, допущені до схрещення (селекція). До осіб застосовується «генетичні оператори» (в більшості випадків це оператор схрещення (crossover) і оператор мутації

(mutation)), створюючи таким чином наступне покоління осіб. Особи наступного покоління також оцінюються застосуванням генетичних операторів і виконується селекція і мутація. Так моделюється еволюційний процес, що продовжується декілька життєвих циклів (поколінь), поки не буде виконано критерій зупинки алгоритму. Таким критерієм може бути:

- Знаходження глобального, або надоптимального вирішення;
- Вичерпання числа поколінь, що відпущені на еволюцію;
- Вичерпання часу, відпущеного на еволюцію.

Генетичні алгоритми можуть використовувати для пошуку рішень в дуже великих і важких просторах пошуку.

1.2.1 Етапи генетичного алгоритму

1) Створення початкової популяції:
2) Обчислення функції допасованості для осіб популяції (оцінювання)
3) Повторювання до виконання критерію зупинки алгоритму:
    a) Вибір індивідів із поточної популяції (селекція)
    b) Схрещення або/та мутація
    c) Обчислення функції допасованості для всіх осіб
    d) Формування нового покоління

1.2.2 Створення початкової популяції

Перед першим кроком необхідно випадковим чином створити деяку початкову популяцію. Навіть якщо популяція виявиться абсолютно неконкурентоздатною, генетичний алгоритм все одно достатньо швидко переведе її в придатну для життя популяцію. Таким чином, на першому кроці можна не старатися зробити надто допасованих осіб, достатньо, щоб вони

відповідали формату осіб популяції, і на них можна було порахувати функцію допасованості. Наслідком першого кроку є популяція H, що налічує N осіб.

1.2.3 Відбір

На етапі відбору необхідно із всієї популяції вибрати її певну долю, яка залишиться в «живих» на цьому етапі популяції. Є декілька способів провести відбір. Ймовірність виживання особи h повинна залежати від значення її допасованості Fitness(h). Сама ж доля відібраних s зазвичай є параметром генетичного алгоритму, і її просто задають заздалегідь. Внаслідок відбору із N осіб популяції H повинні залишитись sN осіб (далі в дослідженняї $s = 10\%$ від початкової популяції), які ввійдуть в наступну популяцію H'. Решта осіб «загине».

Доля відбору базової популяції в дослідженнях становила 10 одиниць потомків, або згідно формули mutations_per_parent *child_per_parent .

Оператори вибору батьків: В даній реалізації використовується метод: basic_selection_method. Даний метод обирає з відсортованого списку тільки кращих представників, згідно цінової функції з кращим результатом. Якщо два поряд розташованих представника мають однакове значення цінової функції, то другий представник відкидається.

Цінова функція вираховується кожний раз, з отриманням нового S-боксу.

1.2.4 Розмноження

Розмноження в генетичних алгоритмах зазвичай статеве — щоб «народити» нащадка, необхідно декілька батьків, зазвичай потрібна участь двох. Розмноження в різних алгоритмах описується по різному — воно, звісно, залежить від формату осіб. Головна вимога до розмноження — щоб нащадок чи

нащадки мали можливість успадкувати риси всіх батьків, «змішавши» їх якимось достатньо розумним чином.

Розмноження або оператор рекомбінації застосовують відразу ж після оператора відбору батьків для отримання нових особин-нащадків. Сенс рекомбінації полягає в тому, що створені нащадки повинні наслідувати генну інформацію від обох батьків. Розрізняють дискретну рекомбінацію і кросинговер.

Приклад операції розмноження: Вибрати $(1-s)\frac{p}{2}$ пар гіпотез із H і провести з ними розмноження, отримавши по два нащадка від кожної пари (якщо розмноження описано так, щоб давати одного нащадка, необхідно вибрати $(1-s)p$ пар), і додати цих нащадків в H'. В результаті H' буде складатися з N осіб.

Особи для розмноження зазвичай вибираються із всієї популяції H, а не із тих, що вижили на першому кроці (хоча останній варіант теж має право на існування). Справа в тому, що головна проблема генетичних алгоритмів — недостача різноманітності (diversity) в особах. Достатньо швидко виділяється єдиний генотип, який являє собою локальний максимум і згодом всі елементи популяції програють йому в відборі, і вся популяція «забивається» копіями цієї особи. Існують різні способи боротьби із таким небажаним ефектом; один з них — вибір для розмноження не з самих «допасованих», а взагалі зі всіх осіб.

Параметри мутації: це перестановка випадкових елементів у самому S-блоці.

```
int pos_1 = random % 256;

int pos_2 = random % 256;

std::swap(new_sbox[pos_1], new_sbox[pos_2]);
```

Рисунок 1.1 – Псевдокод перестановки.

Де, pos_1 – перший випадковий елемент S-боксу та pos_2 – другий випадковій елемент S-боксу

1.2.5 Мутації

До мутацій відноситься все те ж, що і до розмноження: є деяка доля мутантів m, що є параметром генетичного алгоритму, і на кроці мутацій необхідно вибрати mN осіб, а згодом змінити їх згідно з заздалегідь заданими операціями мутації.

1.3 S-бокс

S-бокс (англ. Substitution-box, S-box) — це засаднича складова шифрування з симетричними ключами, яка виконує підстановки. По суті це звичайна таблиця підстановки. У блочних шифрах їх здебільшого використовують для приховування зв’язків між ключем і шифротекстом — властивість плутанини введена Шенноном.

Як правило, S-бокси приймають деяку кількість вхідних бітів – m, і перетворюють їх у деяку кількість вихідних бітів – n, де не обов'язково дорівнює. S-бокс є множиною окремих вихідних булевих функцій, об’єднаних в певному (заданому) порядку. Для S-бокса існує 2^m входів та 2^n можливих виходів.

Щоб бути криптографічно захищеним, S-бокси мають бути бієктивними, щоб запобігти випуску статистичної інформації про відкритий текст по відомому зашифрованому тексту. Криптографічно якісні S-бокси також повинні відповідати іншим критеріям. Одним із найважливіших критеріїв є так звана нелінійність. На сьогодні найбільш популярними є S-бокси 8x8, тобто $m = n = 8$. Такий S-бокс повинен мати нелінійність не менш за 104.

Особу складну задачу становить алгоритм формування S-боксу, який не має у своєму складі будь-яких алгебраїчних конструкції, що у першу чергу

виключає можливість прихованих вразливостей, а по-друге набагато розширяє коло можливих рішень. Таки алгоритми на початку роботи формують деяким випадковим чином бієктивний S-бокс, а потім незначними модифікаціями намагаються його перетворити на криптографічно сильний. До таких алгоритмів відносяться евристичні методи: алгоритм локального пошуку (Local Search Algorithm) або сходження на пагорб (Hill climbing) [3] [4] [5] [6] [7]; метод градієнтного спуску (gradient descent method) [8]; алгоритм імітації відпалу (simulated annealing) [9] [10] генетичний алгоритм (Genetic Algorithm) [1] тощо.

Основне завдання перелічених алгоритмів – зниження (або у деяких випадках збільшення) деякої цільової функції, яка пов’язана з бажаною властивістю S-боксу, що формується. Під час роботи алгоритму виконується наближення характеристик поточного S-боксу до бажаного значення. Успіх вказаного підходу дуже чутливий до обраної цільової функції, а отже до вибору її параметрів. Серед цільових функції можна виділити такі найбільш популярні: функцію вартості Кларка (Clark’s cost functions) [11] та її модифікація; функцію вартості Фрейре-Ечеваррія (Freyre-Echevarría cost functions) [2]; функцію вартості Пічека (Picek’s cost functions) [12] тощо.

На сьогоднішній час задача формування бієктивного S-боксу з нелінійністю більш ніж 104 випадковими методами (які не включають алгебраїчні конструкції) є не вирішеною задачею. Також залишається суперечливим вибір найбільш швидкого евристичного алгоритму для знаходження бієктивного S-боксу з нелінійністю 104. Більшість дослідників, що освітлюють дану тему, наводять результати, які отримані різними методами пошуку та різними цільовими функціями, що робить неможливим порівняння обраних методів між собою.

1.3.1 Визначення S-боксу

S-бокс називають бієктивним, якщо його всі n-компонентні булеві функцій містить в своєму складі всі елементи тривіальної вхідної послідовності $\{0,1,2,\ldots,2^{n-1}\}$, а всі лінійні комбінації компонентних булевих функцій, відповідно, є збалансованими.

Нелінійністю булевої функції f від n змінних (N_f) є мінімальна відстань від f до множини афінних функцій A_n:

$$N_f = dist(f, \mathcal{A}_n) = \min_{g \in \mathcal{A}_n} dist(f, g) \tag{1.1}$$

Через координатні функції наведемо визначення можливого узагальнення на булеві відображення $f = (f_1, f_2, \ldots, f_m)$, виходячи з властивостей сукупності усіх нетривіальних лінійних комбінацій виду

$$l(c, f) = c_1 f_1 \oplus c_2 f_2 \oplus \ldots c_m f_m \tag{1.2}$$

Нелінійність S-боксу $n \times m$, визначається як мінімальна нелінійність з множини нелінійностей координатних булевих функцій та всіх їхніх лінійних комбінацій. У цьому випадку:

$$N_f = \min_{c} N_{l(c,f)}, c \in GF_2^m, c \neq 0 \tag{1.3}$$

Розрахунок нелінійності булевої функцій можна провести застосовуючи перетворення Уолша-Адамара. Це перетворення діє на булевих функціях. При перетворенні Уолша-Адамара елементи 0 і 1 поля GF_2 розглядаються як звичайні цілі числа. Результатом перетворення функції $f(x)$ є цілочисельна функція $WHT_f(a)$, $a \in GF_2^n$, що пов'язана з усіма векторами значень виду

$$h(f,a) = f(x) \oplus \langle a,x \rangle \tag{1.4}$$

Сукупність значень $WHT_f(a)$ ще називають *спектром Уолша-Адамара* (англ. Walsh–Hadamard transform).

У кожному подібному векторі одиниці відповідають місцям неспівпадінь правої частини $f(x)$ з правою частиною відповідної лінійної функції $l_a(x) = \langle a,x \rangle$. Дане перетворення є мірою кореляції між функцією і множиною всіх лінійних функцій.

Нехай вектор $a \in GF_2^n$, розглядається як запис деякого числа у двійковій системі числення. Значення $W_f(a)$ називають *коефіцієнтами Уолша-Адамара* функції f з номером a.

Для довільної булевої функції

$$f, 2^{n/2} \leq \max_a \left| WHT_f(a) \right|, \tag{1.5}$$

звідки випливає:

$$N_f = 2^{n-1} - \frac{1}{2} \max_a \left| WHT_f(a) \right| \leq 2^{n-1} - 2^{n/2-1} \tag{1.6}$$

У випадку, коли $\max_a \left| WHT_f(a) \right| = 2^{n/2}$, цей максимуму є найменшим, при цьому N_f приймає максимальне значення, тобто

$$N_{max} = 2^{n-1} - 2^{\frac{n}{2}-1} \tag{1.7}$$

такі функції називають *бент-функціями*. Оскільки N_f є цілим числом, n – повинно бути обов'язково парне.

У подальшому буде розглянуто S-бокси розміром 8x8 як такі, що є найбільш популярні у прикладних застосунках. Поняття нелінійності легко поширюється на 8x8 S-бокси. У таких випадках нелінійністю S-боксу вважається найгірша (найнижча) нелінійність з усіх 2^8 лінійних комбінацій вихідних функцій.

2 ДОСЛІДЖЕННЯ ГЕНЕТИЧНОГО АЛГОРИТМУ З ВИКОРИСТАННЯМ ВІДОМИХ ЦІЛЬОВИХ ФУНКЦІЙ, ОБҐРУНТУВАННЯ ВИБОРУ ЇХ ОПТИМАЛЬНИХ ПАРАМЕТРІВ

Суть дослідження: знайти найбільш оптимальні параметри і цільову функцію для генетичного алгоритму, щоб спробувати знайти нелінійність S-box у значенні 106. Оптимальними параметрами будуть вважатись ті, з використанням яких буде досягнута нелінійність 104, за найменшу кількість ітерацій – часу.

2.1 Реалізація генетичного алгоритму

1) Виконуємо процедуру генерації s-боксів методом генетичних алгоритмів. Генерується початкова популяція розміру *mutations_per_parent*child_per_parent.*

Метод може працювати як у первинному, так і вторинному режимі. Генерація початкової популяції у первинному режимі полягає у генерації випадкового S-боксу.

Лямбда-вираз для генерації початкової популяції у вторинному режимі. У вторинному режимі популяція складається з мутацій початкового S-боксу. Мутація полягає в обміні двох випадкових позицій в s-боксі:

```
int idx1 = distrib(gen) % 256;
int idx2 = distrib(gen) % 256;
std::swap(new_sbox[idx1], new_sbox[idx2]);
```

Рисунок 2.1 – Фрагмент коду для мутації.

Якщо параметр *input* (покажчик на початковий S-бокс)– не нульовий, то метод викликається як вторинний. У цьому випадку використовується *secondary_initial_generation*. Інакше – *primary_initial_generation* – первинний метод.

2) У головному циклі виконується вибірка кращих представників популяції, і *iterations_count* разів повторюються наступні дії для створення нової популяції:

2.1) Викликається метод селекцій. Дана програмна реалізація підтримує 3 види селекторів:

- *basic_selection_method*. Обирає з відсортованого списку тільки кращих представників. Якщо два представника, що йдуть підряд, мають однакові значення цільової функції, то другий представник відкидається. Ця умова задана через необхідність виходу з локальних мінімумів. Далі в роботі використовується як основний метод для досліджень.
- *rank_selection_method*. Обирає представників випадково з відсортованого за зменшенням цільової функції масиву. Ймовірність, що s-бокс буде обраний складає $\frac{2i}{M(M+1)}$, де i – позиція в масиві, M – число елементів в масиві.
- *roulette_wheel_selection.* Обирає представників випадково з відсортованого за зменшенням цільової функції масиву. Ймовірність, що s-бокс буде обраний складає $\frac{cos\, t_i}{\sum_{j=1}^{M} cos\, t_i}$, де i – позиція в масиві, M – число елементів в масиві, *cost* – значення функції вартості.

Метод селекцій залишає *child_per_parent* представників з популяції

2.2) Викликається метод кроссовера для тих, хто пройшов селекцію. У даній реалізації генетичного алгоритму застосовано два різні методи кросинговеру: *PMX* («частково відображений кросовер») і циклічний кросовер.

- Циклічний кросовер працює наступним чином:

За допомогою генератора випадкових чисел заповнюються S-бокси *parent1* та *parent2*. Випадково вибирається початкова точка циклу.

Елемент *parent1* у початковій точці циклу копіюється в дочірній елемент у тій же позиції: Елемент у тій же позиції в *parent2* є наступним для копіювання в дочірній елемент. Однак він копіюється в ту саму позицію, в якій він зустрічається в батьківському 1. Цей процес триває, доки процес не поверне нас до вихідної початкової точки циклу - іншими словами, коли створено «петля» або «цикл».

Будь-які інші позиції, по яким не пройшов цикл в перший раз, в дочірньому елементі потім заповнюються шляхом копіювання відповідних значень із *parent2*

- Кросовер *PMX* починається заповненням S-боксів *parent1* та *parent2* і випадковим вибором проміжку. Елементи *parent1* у цьому проміжку копіюються в дочірній елемент:

Далі копіюються елементи *parent2*, які ще не були скопійовані в дочірній елемент. Якщо значення *parent2* вже було використано, тоді шукаємо перше не використане.

```
while (index_values[value] == true) {
   for (int j = 0;j < 256;j++) {
      if (parent1[j] == value) {
         value = parent2[j];
         break;
      }
   }
}
```

Рисунок 2.2 – Фрагмент коду для кросоверу PMX.

Усі новоутворені S-бокси додаються в популяцію. Створюємо нову популяцію з кращих представників минулої популяції.

2.3) Тепер реалізуються безпосередні розрахунки за методом генетичних алгоритмів. Над кожним S-боксом в популяції, що утворилася після селекції та кросоверу, проводяться мутації.

Таким чином, додаємо в нову популяцію *mutations_per_parent* мутацій представника за основу береться поточний представник минулого покоління. обираються дві випадкові позиції і міняються місцями:

```
                for (int i = 0;i <
method_data.mutations_per_parent;i++) {
                    SBox mutant = successor;
                    int pos_1 = distrib(gen) % 256;
                    int pos_2 = distrib(gen) % 256;
                    std::swap(mutant[pos_1], mutant[pos_2]);
```

Рисунок 2.3 – Фрагмент коду з додаванням нової популяції.

Процедура реалізує безпосередні розрахунки за методом генетичних алгоритмів у кожному потоці. Тут оновлюються значення функції вартості, порівнюються значення нелінійності, цільової функції, дельта рівномірності, мінімального степеня та алгебраїчного імунітету нового s-боксу. Якщо будь-який з цих параметрів не дорівнює необхідному, то починаємо нову спробу пошуку. Коли s-бокс з потрібними характеристиками знайдено, додаємо його в популяцію.

Отже, над кожним S-боксом проводиться *mutations_per_parent* мутацій, і всі мутанти додаватимуться в популяцію. Було проведено декілька десятків запусків генетичного алгоритму з різними параметрами і цільовими функціями у різних, однакових для схожих функцій, діапазонах. При цьому деякі дослідження показували відмінні результати, а деякі не дали жодного результату.

Таким чином, обравши найбільш оптимальні параметри запуску, і розширивши час роботи програми, може бути можливим досягти нелінійності 106.

Параметри запуску для генетичного алгоритму:

/*кількість потоків*/ threads_count = 6

/*необхідна нелінійність*/ target_nonlinearity=104

/*кількість мутацій кожного нащадку*/ mutations_per_parent=10

/*кількість нащадків кожного родича*/ child_per_parent=10

/*максимальна кількість поколінь*/ iterations_count=15000

/* компаратор */ comparator = sbox_compare104_random

/*метод відбору кращих представників*/ selection_method = basic_selection_method

Дані параметри обрані на основі приблизно 50 тестових запусків з різними ціновими функціями, де дані тестові спроби зазвичай досягали поставленої нелінійності менш ніж за 15000 ітерацій з кількістю мутацій нащадків 10, та кількістю нащадків від кожного родича – 10. Кількість потоків встановлена у значенні 6, так як процесор в персональному комп'ютері має 6 ядер. Подальше збільшення кількості потоків лише погіршує результат.

В таблицях наведених нижче, символ «-» вказує на те, що з даними параметрами не вдалось отримати результату. Перехрещені значення – параметри, що не відповідають дійсності, тобто такі, за яких не можливо запустити пошук.

2.2 Цільова функція WCF

$$\sum_{i=1}^{256*256} \prod_{\substack{j=start \\ j+=step}}^{end} (|WHT[i]| - j) \tag{2.1}$$

де *WHT* – (англ. Walsh–Hadamard transform) спектральні коефіцієнти Уолша–Адамара;

i – цикл за всіма компонентними функціями та їх лінійними комбінаціями;

j – цикл за всіма лінійними функціями;

start – параметр початку з дійсними значеннями.

step – параметр кроку з дійсними значеннями.

end – параметр кінця з дійсними значеннями.

Функція *WCF*[16] була запропонована і досліджена у роботі Alejandro[17]. Порівнявши результати досліджень в нашому випадку функція з генетичним алгоритмом досягла бажаного результату в 77 разів і в 57 разів швидше ніж із алгоритмами LSA та HC відповідно. Ці алгоритми були згадані у роботі Alejandro[17], дані досліджень наведені у Таблиці 2.3.

Таблиця 2.1 – Середньостатистичний час, за який було знайдено цільовий S-бокс при використанні функції WCF.

	wcf.end								
Wcf.step	**32**	**28**	**24**	**20**	**16**	**12**	**8**	**4**	**2**
32	**-**								
28	**-**	**-**							
24	**-**	**-**	**-**						
20	**-**	**-**	**-**	**-**					
16	**-**	**-**	**-**	**-**	**-**				
12	**-**	**-**	**-**	**-**	**-**	**-**			
8	**47,06**	**-**	**202.14**	**-**	**-**	**-**	**-**		
4	**28,46(36,3)**	**30,55**	**29,87**	**52,01**	**112.11**	**-**	**-**	**-**	
2	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**

Таблиця 2.2 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції WCF.

	wcf.end								
Wcf.step	**32**	**28**	**24**	**20**	**16**	**12**	**8**	**4**	**2**
32	**-**								
28	**-**	**-**							
24	**-**	**-**	**-**						
20	**-**	**-**	**-**	**-**					
16	**-**	**-**	**-**	**-**	**-**				
12	**-**	**-**	**-**	**-**	**-**	**-**			
8	**2100**	**>15000**	**9642**	**>15000**	**-**	**-**	**-**		

Продовження Таблиці 2.2 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції WCF.

4	**1159**	**1180**	**1166**	**2000**	**4666**	**>15000**	**-**	**-**		
2	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	

Результат у дужках в таблиці 2.1 вказано, після додаткових досліджень. Зробивше більше досліджень з даним параметром прослідковується збільшення середньостатистичного часу. Така зміна в часі пояснюється тим, що з більшою кількістю експериментів збільшується кількість даних для підрахування середньостатистичного часу і тим, що виконуючи дослідження на персональному комп'ютері, дослідженню могли заважити зайві процеси.

Таблиця 2.3 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції WCF у роботі Alejandro [17].

Метод		Нелінійність
Алгоритм	Функція	104
LSA	WCF	**89460**
HC	WCF	**65933**

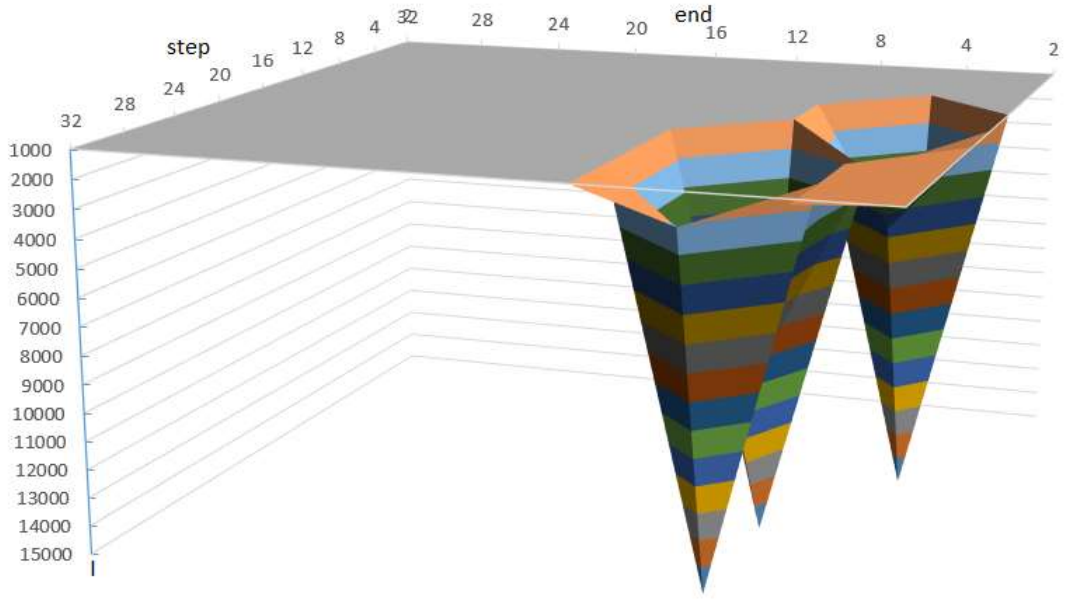

Рисунок 2.4 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції WCF при різних параметрах *wcf.step* та *wcf.end.*

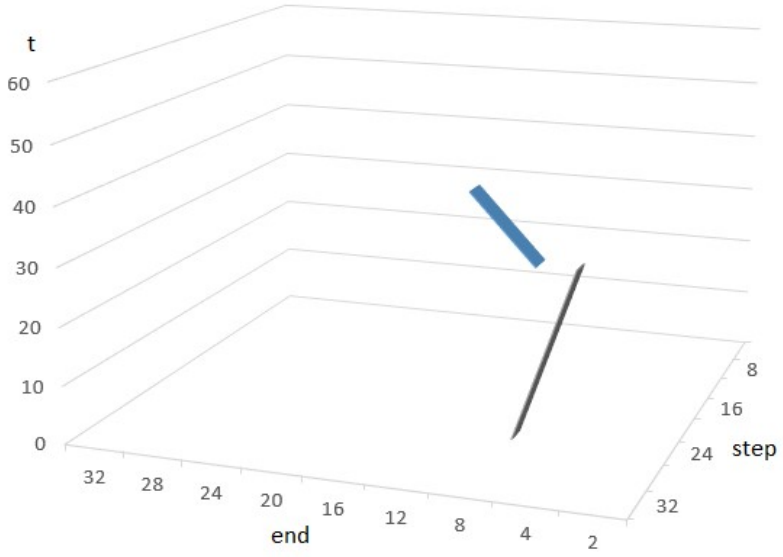

Рисунок 2.5 – Середньостатистичний час який витрачено, до знаходження цільового S-боксу при використанні функції WCF при різних параметрах *wcf.step* та *wcf.end.*

Як бачимо, використовуючи функцію WCF та генетичний алгоритм пошуку, можна дуже швидко знайти біективні S-бокси з нелінійністю $N_f = 104$.

Виходячи з отриманих результатів, оптимальними (тобто найкращими з точки зору найменшої кількості ітерацій до знаходження цільового S-боксу) параметрами цільової функції WCF можна вважати $wcf.start = 0$, $wcf.step = 4$ та $wcf.end = 32/28/24$. При цьому бажаний результат досягається набагато швидше ніж у роботі Кларка та у роботі Тесара [13]. Тесарю вдалося досягти нелінійності у 104, використавши генетичний алгоритм з комбінацією алгоритму дерева та цільовою функцією WHS (параметри $R = 7, X = 21$) приблизно за 3.239 мільйона ітерацій. Генетичний алгоритм та цільова функція WHS (параметри $R = 3, X = 4$) не дала результату, опираючись на роботу Тесара. Таким чином, в даній роботі кількість ітерацій скоротилась приблизно в 2790 разів. На Рисунку 2.4 та 2.5 графічно відображено отримані результати.

При оптимальних значеннях параметрів цільової функції ($wcf.start = 0$, $wcf.step = 4$, $wcf.end = 32$) проведемо додаткові дослідження, результати яких наведено у таблиці 2.4 (щодо витраченого часу) та у таблиці 2.5 (щодо кількості ітерацій). N вказує номер випробування.

Таблиця 2.4 – Середньостатистичний час, за який було знайдено цільовий S-бокс при використанні функції WCF де *N* – номер випробування

	wcf.end = 32								
N	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
wcf.step=4	*35,7*	*33,98*	*44,59*	*37,53*	*32.08*	*33.69*	*42.31*	*46.59*	*40,35*
N	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
wcf.step=4	*31,35*	*36,10*	*31,94*	*37,52*	*38,42*	*34,72*	*34,33*	*35,63*	*34.48*
N	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**
wcf.step=4	*32.53*	*32.07*	*37,14*	*39.95*	*35.95*	*38,35*	*33.39*	*39.80*	*38.16*
N	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
wcf.step=4	*28.46*	*38.63*	*32.73*	*37.95*	*43.39*	*34,79*	*38.89*	*35.55*	*31.74*
N	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**

Продовження Таблиці 2.4 – Середньостатистичний час, за який було знайдено цільовий S-бокс при використанні функції WCF де *N* – номер випробування

	wcf.end =32								
wcf.step=4	*36.59*	*45.65*	*27,44*	*29.67*	*38,19*	*38.20*	*36,32*	*44.70*	*37.70*
N	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**	**54**
wcf.step=4	*34,15*	*33.30*	*23.09*	*37.67*	*36.96*	*34.87*	*31,59*	*36,86*	*38.20*
N	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
wcf.step=4	*25,73*	*52.15*	*38.78*	*26.48*	*37,42*	*25.3*	*40,38*	*33,92*	*30,88*
N	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**
wcf.step=4	*37,14*	*40,03*	*37.36*	*49.54*	*31,11*	*37.42*	*42,58*	*41,94*	*41.74*
N	**73**	**74**	**75**	**76**	**77**	**78**	**79**	**80**	**81**
wcf.step=4	*35.45*	*44,11*	*32,44*	*42.41*	*35,48*	*35.67*	*45,14*	*39,67*	*35.43*
N	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**
wcf.step=4	*44,46*	*38,38*	*29,61*	*30,05*	*36,16*	*43,67*	*38,47*	*31.42*	*31.03*
N	**91**	**92**	**93**	**94**	**95**	**96**	**97**	**98**	**99**
wcf.step=4	*27,39*	*46,81*	*38,70*	*28.70*	*39,1*	*42.04*	*42,44*	*32.40*	*31,85*
N	**100**	**101**	**102**	**103**	**104**	**105**	**106**	**107**	**108**
wcf.step=4	*34.32*	*23,91*	*29,68*	*36,98*	*39.42*	*25.68*	*42.12*	*30,26*	*35.78*
N	**109**	**110**	**111**	**112**	**113**	**114**	**115**	**116**	**117**
wcf.step=4	*21,25*	*39,53*	*37,34*	*35.43*	*28,54*	*37,88*	*24.26*	*37.18*	*31,64*
N	**118**	**119**	**120**	**121**	**122**	**123**	**124**	**125**	**126**
wcf.step=4	*41,79*	*30.63*	*28,34*	*28,34*	*57,98*	*40,16*	*48,46*	*26.15*	*35,58*

Таблиця 2.5 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції WCF

	wcf.end =32								
N	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
wcf.step=4	*1166*	*1200*	*1100*	*1233*	*1100*	*1200*	*1333*	*1000*	*1366*
N	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
wcf.step=4	*800*	*1266*	*1133*	*1166*	*1200*	*1233*	*1100*	*1066*	*1200*
N	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**
wcf.step=4	*966*	*1033*	*1100*	*1033*	*1266*	*1300*	*1200*	*1266*	*1200*
N	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
wcf.step=4	*1000*	*1050*	*1100*	*1133*	*1666*	*1200*	*1266*	*1200*	*1200*
N	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**
wcf.step=4	*1366*	*1400*	*900*	*950*	*1166*	*1100*	*1266*	*1400*	*1300*

Продовження Таблиці 2.5 – Середньостатистичний час, за який було знайдено цільовий S-бокс при використанні функції WCF де *N* – номер випробування

	wcf.end =32								
N	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**	**54**
wcf.step=4	*1200*	*900*	*933*	*1133*	*1200*	*1166*	*1166*	*1333*	*1200*
N	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
wcf.step=4	*866*	*1700*	*1133*	*900*	*1200*	*900*	*1466*	*1266*	*1100*
N	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**
wcf.step=4	*1233*	*1166*	*1000*	*1466*	*800*	*1000*	*1300*	*1300*	*1366*
N	**73**	**74**	**75**	**76**	**77**	**78**	**79**	**80**	**81**
wcf.step=4	*933*	*1166*	*1066*	*1166*	*1066*	*1166*	*1233*	*1233*	*1000*
N	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**
wcf.step=4	*1500*	*1100*	*966*	*966*	*1166*	*1433*	*1233*	*1000*	*1100*
N	**91**	**92**	**93**	**94**	**95**	**96**	**97**	**98**	**99**
wcf.step=4	*1033*	*1300*	*1033*	*1066*	*1033*	*1900*	*1300*	*1500*	*900*
N	**100**	**101**	**102**	**103**	**104**	**105**	**106**	**107**	**108**
wcf.step=4	*1300*	*1100*	*700*	*1300*	*1233*	*1100*	*800*	*1100*	*900*
N	**109**	**110**	**111**	**112**	**113**	**114**	**115**	**116**	**117**
wcf.step=4	*1400*	*1000*	*1300*	*1200*	*666*	*1133*	*1166*	*1233*	*900*
N	**118**	**119**	**120**	**121**	**122**	**123**	**124**	**125**	**126**
wcf.step=4	*1400*	*1400*	*1133*	*1100*	*900*	*1266*	*1200*	*933*	*1366*

Кожне N випробування містить у собі від 3 до 7 запусків програми, таким чином було проведено близько 600 випробувань для отримання результату. При кожному проведеному випробувані було отримано S-бокс з нелінійністю у 104.

Проведемо додаткові дослідження, збільшивши кількість початкової популяції та нащадків. Результати відображено у Таблиці 2.6 у порівнянні з початковими параметрами популяції та кількості нащадків у 100 та 10 відповідно. S-бокс вираховується при кожній мутації нового нащадка, тобто, якщо нащадок має 10 мутацій, то цінова функція відповідно буде обчислена 10 раз.

Також було досліджено можливість формування високонелінійних бієктивних S-боксів за допомогою простого за реалізацією генетичного алгоритму зі застосуванням цільової функції WCF.

В якості оптимальних параметрів функції WCF було визначено $wcf.start = 0$, $wcf.step = 4$ та $wcf.end = 32/28/24$. При оптимальних параметрах та використанні генетичного алгоритму пошуку з ймовірністю близької до 99% може бути сформований бієктивний S-бокс з нелінійністю 104, приклади отриманих S-боксів вказано у Додатку 1.

Середньостатистична кількість ітерацій алгоритму пошуку, при оптимальних параметрах функції WCF, складає 1159, що становить близько 36 секунд, або приблизно 3,248 мс. на одну ітерацію. Враховуючи можливість розпаралелювання процесу пошуку є можливо багатократно (кратність відповідає кількості потоків, що одночасно виконуються) скоротити час пошуку.

Збільшуючи базову популяцію та кількість прямих нащадків вдалося досягти зменшення кількості виконуваних ітерацій, але збільшився час на пошук цільового S-боксу. Якщо порівнювати середній час на виконання одної ітерації найкращі параметри початкової популяції та нащадків становлять 100 до 10.

Таблиця 2.6 – Порівняльна таблиця, до знаходження цільового S-боксу зі збільшенням початкової популяції та нащадків.

Популяція (нащадків)	Середній час пошуку S-боксу (с.)	Середня кількість ітерацій	Середній час на одну ітерацію (мс.)	Середня кількість порахованих S-боксів
50 (5)	32,16	2037	15,79	101850
100 (10)	36,3	1159	31,32	115900
250 (25)	75,42	573	131,67	143250
500 (50)	134,50	529	254,25	264500
1000 (100)	272,87	462	590,63	462000

Зауважмо, що ймовірно, використавши іншу цільову функцію - не можна буде досягти кращого результату, за меншу кількість ітерацій. Тому доцільно залишити ті самі параметри запуску для генетичного алгоритму, одним з яких являється кількість поколінь у значенні 15000.

2.3 Цільова функція power_max_wht

$$\sum_{i=1}^{256}(|WHT[i].max| - x\_param)^{r\_param} \quad (2.2)$$

де *WHT* – (англ. Walsh–Hadamard transform) спектральні коефіцієнти Уолша–Адамара;

i – цикл за всіма компонентними функціями та їх лінійними комбінаціями;

x_param і *r_param* – параметри з дійсними значеннями.

Функція згадується у роботі Тесара[13]. Перетворення Уолша-Адамара зазвичай є основою для інших цінових функцій для обчислення нелінійності. Дане перетворення, це приклад узагальненого класу перетворень Фур'є. Воно виконує ортогональне, симетричне, інволютивне (обернене до самого себе), лінійне перетворення над матрицею розміром 2^m з дійсних чисел. Розглянута цільова функція є модифікацією перетворень Уолша-Адамара.

Таблиця 2.7 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції *power_max_wht*

	R									
X	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**
28	-	-	-	-	-	-	-	-	-	-
24	-	-	-	-	-	-	-	-	-	-
20	-	-	-	-	-	-	-	-	-	-
16	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	-	-	-

Продовження Таблиці 2.7 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції *power_max_wht*

4	-	-	-	-	-	-	-	-	-	-
0	-	-	-	-	-	-	-	-	-	-

Таблиця 2.8 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції *power_max_wht*

	R									
X	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**
28	-	-	-	-	-	-	-	-	-	-
24	-	-	-	-	-	-	-	-	-	-
20	-	-	-	-	-	-	-	-	-	-
16	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-
0	-	-	-	-	-	-	-	-	-	-

Використавши параметри запуску алгоритму, які вказано вище, з цільовою функцією *power_max_wht* не вдалось отримати шуканої нелінійності у значенні 104, використавши діапазон значень, як у таблиці 2.7 та 2.8. Продовжувати пошук з даною функцією збільшивши кількість поколінь (ітерацій), не є доцільним, так як цільова функція WCF вже дала гарний результат при пошуку.

2.4 Цільова функція product_max_wht

$$\sum_{i=1}^{256} \prod_{\substack{j=sta \\ j+=step}}^{end} (|WHT[i].max| - j) \tag{2.3}$$

де *WHT* – спектральні коефіцієнти Уолша–Адамара;

i – цикл за всіма компонентними функціями та їх лінійними комбінаціями;

j – цикл за всіма лінійними функціями;

start – параметр початку з дійсними значеннями.

step – параметр кроку з дійсними значеннями.

end – параметр кінця з дійсними значеннями.

Таблиця 2.9 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції *product_max_wht*

	wht.end								
wht.step	**32**	**28**	**24**	**20**	**16**	**12**	**8**	**4**	**2**
32	**208,87**								
28	**-**	**-**							
24	**-**	**-**	**-**						
20	**-**	**-**	**-**	**-**					
16	**185,02**	**143,24**	**-**	**-**	**-**				
12	**159,47**	**99,25**	**-**	**-**	**-**	**-**			
8	**55,41**	**180,13**	**71,22**	**154,26**	**154.97**	**-**	**-**		
4	**51,14**	**89,53**	**45**	**61,20**	**59,79**	**118,29**	**-**	**-**	
2	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**

Таблиця 2.10 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції *product_max_wht*

	wht.end								
wht.step	**32**	**28**	**24**	**20**	**16**	**12**	**8**	**4**	**2**
32	**12900**								
28	**-**	**-**							
24	**-**	**-**	**-**						
20	**>15000**	**-**	**-**	**-**					
16	**11150**	**8950**	**-**	**-**	**-**				
12	**10233**	**7400**	**-**	**-**	**-**	**-**			
8	**3160**	**10750**	**3966**	**9866**	**10000**	**-**	**-**		
4	**2925**	**5066**	**2466**	**3775**	**3566**	**7266**	**-**	**-**	

Продовження Таблиці 2.10 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції *product_max_wht*

2	-	-	-	-	-	-	-	-	-

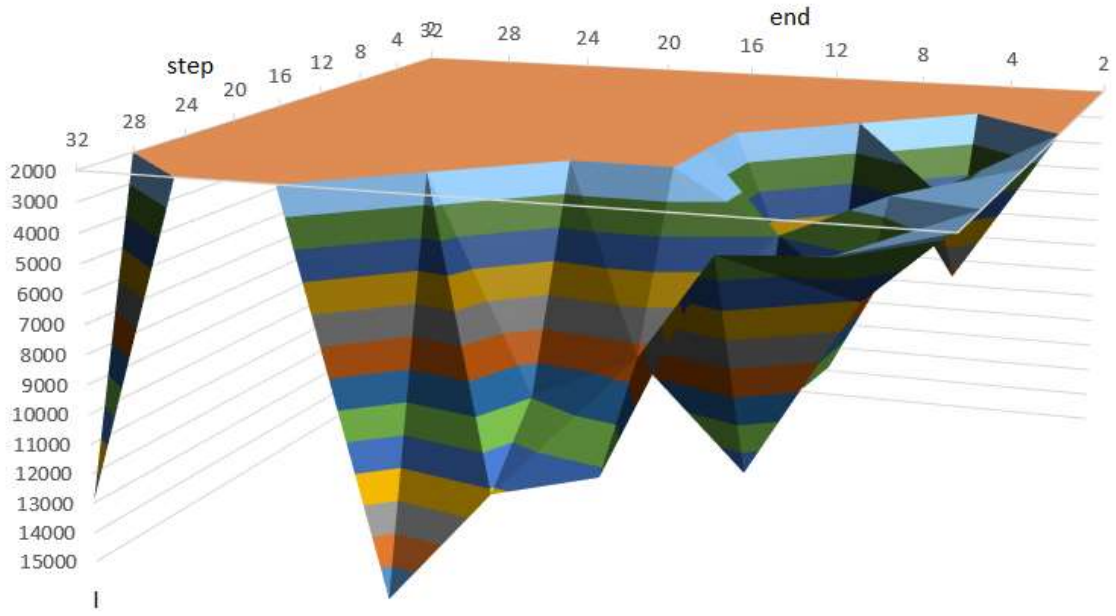

Рисунок 2.6 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції *product_max_wht* при різних параметрах *step* та *end.*

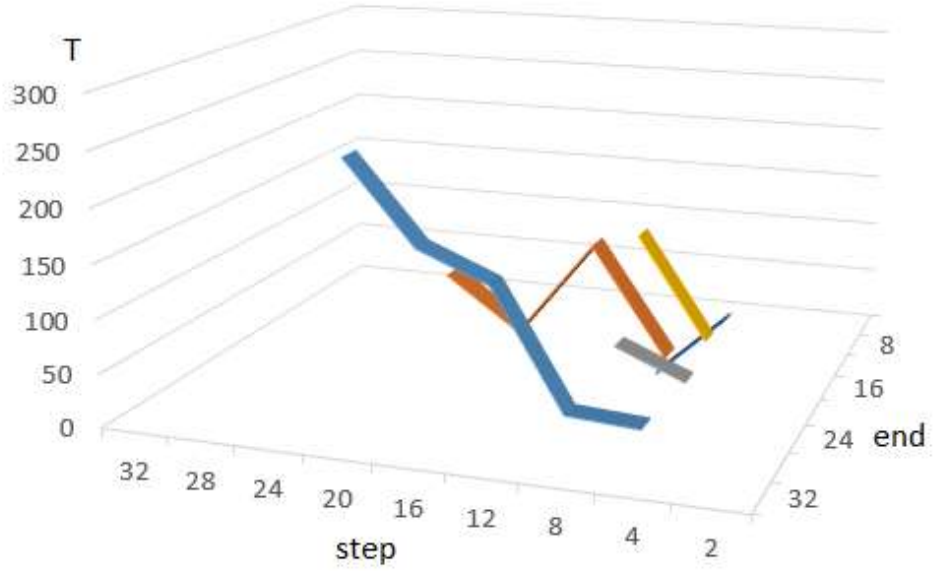

Рисунок 2.7 – Середньостатистичний час який витрачено, до знаходження цільового S-боксу при використанні функції *product_max_wht* при різних параметрах *step* та *end.*

Як бачимо, використовуючи функцію *product_max_wht* можна дуже швидко знайти біективні S-бокси з нелінійністю N_f = 104. Однак, бажаний результат досягнуто за трохи більшу кількість ітерацій ніж у випадку з функцією wcf.

Виходячи з отриманих результатів, оптимальними (тобто найкращими з точки зору найменшої кількості ітерацій до знаходження цільового S-боксу) параметрами цільової функції *WHT* можна вважати *wht.step* = 4 та *wht.end* = 32/24. При чому, у середньому, буде достатньо виконати 2466 ітерацій алгоритмом пошуку для досягнення мети. Крім того, є область значень сусідніх до оптимальних, де середня кількість ітерацій дещо більші, ніж при оптимальних значеннях. На Рисунках 2.6 та 2.7 показано графічно отримані результати.

Проведемо додаткові дослідження, збільшивши кількість початкової популяції та нащадків. Результати відображено у Таблиці 2.11 у порівнянні з початковими параметрами популяції та кількості нащадків у 100 та 10 відповідно.

Було досліджено можливість формування високонелінійних бієктивних S-боксів за допомогою простого за реалізацією генетичного алгоритму зі застосуванням цільової функції WHT.

В якості оптимальних параметрів функції WHT було визначено *wht.step* = 4 та *wht.end* = 32. При оптимальних параметрах та використанні генетичного алгоритму пошуку з ймовірністю близької до 99% може бути сформований бієктивний S-бокс з нелінійністю 104.

Середньостатистична кількість ітерацій алгоритму пошуку, при оптимальних параметрах функції WHT, складає 2466, що становить близько 45 секунд, або приблизно 1,8248 мс. на одну ітерацію.

Збільшуючи базову популяцію та кількість прямих нащадків вдалося досягти зменшення кількості виконуваних ітерацій, проте збільшився час на пошук цільового S-боксу. Якщо порівнювати середній час на виконання одної ітерації найкращі параметри початкової популяції та нащадків становлять 100 до 10.

Таблиця 2.11 – Порівняльна таблиця, до знаходження цільового S-боксу зі збільшенням початкової популяції та нащадків

Популяція (нащадків)	Середній час пошуку S-боксу (с.)	Середня кількість ітерацій	Середній час на одну ітерацію (мс.)	Середня кількість порахованих S-боксів
50 (5)	46,54	4614	10,086	230700
100 (10)	45	2466	18,248	246600
250 (25)	89,91	2028	44,334	505000
500 (50)	92,148	1142	80,69	571000
1000 (100)	157,13	933	168,41	933000

2.5 Цільова функція product_ddt

$$\sum_{i=1}^{256*256} \prod_{\substack{j=start \\ j+=step}}^{end} (|DDT[i]| - j) \tag{2.4}$$

i – цикл за всіма компонентними функціями та їх лінійними комбінаціями;

j – цикл за всіма лінійними функціями;

start – параметр початку з дійсними значеннями.

step – параметр кроку з дійсними значеннями.

end – параметр кінця з дійсними значеннями.

Цільова функція DDT розглянута у даній роботі, як така, яку можливо застосувати у комбінації з іншими функціями для повернення результату нелінійності назад і продовження пошуку з новим результатом.

Таблиця 2.12 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції product_ddt.

	product_ddt.end								
ddt.step	**32**	**28**	**24**	**20**	**16**	**12**	**8**	**4**	**2**
32	-								
28	-	-							
24	-	-	-						
20	-	-	-	-					
16	-	-	-	-	-				
12	-	-	-	-	-	-			
8	-	-	-	-	-	-	-		
4	-	-	-	-	-	-	-	-	
2	-	-	-	-	-	-	-	-	-

Таблиця 2.13 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції product_ddt.

	product_ddt.end								
ddt.step	**32**	**28**	**24**	**20**	**16**	**12**	**8**	**4**	**2**
32	-								
28	-	-							
24	-	-	-						
20	-	-	-	-					
16	-	-	-	-	-				
12	-	-	-	-	-	-			
8	-	-	-	-	-	-	-		
4	-	-	-	-	-	-	-	-	
2	-	-	-	-	-	-	-	-	-

Дана цільова функція направлена на зменшення параметру нелінійності. Досягти нелінійності більше ніж у 102 на початку роботи програми не можливо, проте дана функція можлива у використанні в комбінації з іншою функцією, для того, щоб повернути значення нелінійності до бажаного і продовжити пошук з використанням іншої функції. Використання даної функції в комбінації з іншими можливо покращить результат.

2.6 Цільова функція WHS

Цільова функція *WHS* була запропонована Кларком у роботі [11] та має наступний вигляд:

$$WHS = \sum_{b=1}^{255} \sum_{i=0}^{255} ||WHT[b,i]| - X|^R \tag{2.5}$$

де *WHT* – (англ. Walsh–Hadamard transform) спектральні коефіцієнти Уолша–Адамара;

i – цикл за всіма компонентними функціями та їх лінійними комбінаціями;

b – цикл за всіма лінійними функціями;

X і *R* – параметри з дійсними значеннями.

У даному розділі в якості цільової функції розглядається функція WHS, яка була запропонована Кларком у роботі [11]. Ця цільова функція ураховує весь спектр Уолша Адамара, а не лише його частину, як, наприклад функція WCF. Кларк у своїй роботі зазначає, що важко передбачити, якими мають бути найкращі значення параметрів *X* та *R*, і необхідні значні експерименти. Однак, як стверджує Кларк у своїй роботі, на той час вони дали деякі виняткові результати, які фактично дорівнюють найкращим результатам теоретиків для функцій з 8-ми входами. Однак, йому не вдалось знайти S-бокси з нелінійністю вище 102. Кларк використовував для пошуку алгоритм імітації відпалу (англ. – simulated annealing) та параметри функціі WHS $R = 3$ та $X = -4, -3, -2, -1, 0, 1, 2, 3, 4$.

Аналізуючи роботу Полуяненка [15] та Тесара [13], використаємо дану функцію, але з генетичним алгоритмом, використаємо оптимальні параметри роботи програми описані вище.

Результати досліджень Полуяненка [15] наведено у таблиці 2.16. Символом «–» позначено випадки, коли алгоритмом пошуку було знайдено цільовий S-бокс менш ніж у 50% іспитах та отримані результати не можуть характеризувати необхідну для пошуку кількість ітерації.

Результати досліджень Тесара наведено у таблиці 2.17.

Тесар, у своїй роботі [13] використовував цільову функцію WHS у методі пошуку з використанням генетичного алгоритму із пошуком повного дерева (GaT – Genetic and Tree). Він дослідив значення R з інтервалу [3, 25] і цілим значенням X з інтервалу [-25, 25] (всього 1173 пари). За результатами свого дослідження Тесар вважає параметри $R = 7$ і $X = 21$ найкращими.

У роботі [18] досліджується значення $R = 2; 2.5; 3$ і цілим значенням X з інтервалу [-16, 16] з кроком 2. Краща нелінійність для окремої булевої функції визначено $R = 3$ та $R = 3$.

Як бачимо, оптимально наведені параметри в різних роботах для функції WHS значно відрізняється.

Таблиця 2.14 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції WHS.

	R									
X	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**
28	-	-	-	-	-	-	-	-	-	-
24	-	-	-	-	-	-	-	-	-	-
20	-	-	-	-	-	-	-	-	-	-
16	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-
0	-	-	-	-	-	-	-	-	-	-

Таблиця 2.15 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції WHS.

	R									
X	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**
28	-	-	-	-	-	-	-	-	-	-
24	-	-	-	-	-	-	-	-	-	-
20	-	-	-	-	-	-	-	-	-	-
16	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-
0	-	-	-	-	-	-	-	-	-	-

Таблиця 2.16 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції WHS згідно з роботи Полуяненка. [15]

	R									
X	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**
28	–	–	–	–	–	–	–	–	–	–
24	–	–	–	–	–	–	–	–	–	–
20	163 762	132 661	120 302	86 567	82 720	97 632	135 248	–	–	–
16	135 713	108 804	80 965	77 219	60 552	*52 332*	*54 553*	63 123	76 407	–
12	97 179	74 352	64 079	*55 569*	*53 835*	*52 829*	61 261	61 182	74 028	180 759
8	75 484	64 675	*54 808*	*55 230*	*54 412*	*58 811*	63 599	88 677	131 640	–
4	62 673	*56 544*	*51 477*	*51 278*	*52 746*	62 225	67 845	119 959	200 386	–
0	*58 345*	*54 004*	*50 265*	*54 607*	*59 391*	68 247	75 202	130 057	–	–

Таблиця 2.17 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції WHS згідно з роботи Тесара. [13]

Метод \ Нелінійність	98	100	102	104
Випадковий пошук	695	12,5 M	NRP	NRP
Сходження на гору	NA	2500	NRP	NRP
GaT з CF=нелінійність	171	29648	NRP	NRP
GaT з CF=$WHS_{3,4}$	217	2714	79385	NRP
GaT з CF=$WHS_{7,21}$	64	522	9859	3,239 M

Середні значення критерію в Таблиці 2.16 (M — це мільйон, NA — не доступний, а NRP означає, що метод не може досягти цього значення.).

Було проведено декілька десятків запусків генетичного алгоритму з різними параметрами і цільовими функціями у діапазоні, що були запропоновані

Полуяненком у його роботі [15]. При цьому жодного разу не було досягнуто нелінійність 104. Це не означає, що неможливо досягнути цього результату, використовуючи цю функцію з генетичним алгоритмом, однак треба або значно більша кількість ітерацій, або застосовувати інший алгоритм пошуку.

Порівнявши роботу Тесара, та зроблений аналіз, можна закріпити отриманий результат, а саме: для досягнення бажаної нелінійності слід розширити кількість поколінь для генетичного алгоритму, Тесар використав до 9 мільйонів поколінь. Проте в цьому не має ніякого сенсу, адже функція WCF дає кращі результати.

2.7 Цільова функція power_ddt

$$\sum_{i=1}^{256*256} (|DDT[i].max| - x\_param)^{r\_param} \tag{2.6}$$

i – цикл за всіма компонентними функціями та їх лінійними комбінаціями;

x_param і r_param – параметри з дійсними значеннями.

Модифікація цінової функції DDT розглянута у даній роботі, як така, яку можливо застосувати у комбінації з іншими функціями для повернення результату нелінійності назад і продовження пошуку з новим результатом.

Таблиця 2.18 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції power_ddt.

	R									
X	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**
40										
36										
32										
28										

Продовження Таблиці 2.18 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції power_ddt

	R									
X	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**
24										
20										
16										
12										
8	-	-	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-
0										

Таблиця 2.19 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції power_ddt.

	R									
X	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**
40										
36										
32										
28										
24										
20										
16										
12	-				-					
8	-	-	-	-	-	-	-	-	-	-
4	-	-	-	-	**>15000**	-	-	-	-	-
0										

Представлена цінова функція є модифікацією згаданої вище функції product_ddt. Вона також спрямована на зменшення значення нелінійності, однак після деякого кроку назад, починається збільшення нелінійності. Результату нелінійності у 104 вдалося досягти, але довелось вийти за рамки визначеної кількості поколінь.

2.8 Цільова функція PCF

Функції вартості Пічека (Picek's cost functions) [12]. Автор у своїй роботі, за допомогою запропонованої функції, намагався зменшити кількість коефіцієнтів, які мають максимальне абсолютне значення. Як тільки кількість максимальних коефіцієнтів абсолютного значення зменшується до нуля, нелінійність S-боксу збільшується. Як зазначено в роботі [12], S-блоки спочатку порівнюються за значенням нелінійності (чим більше, тим краще), і якщо значення нелінійності дорівнює шуканому значенню, тоді встановлюється, що кращим S-боксом є той, у якого менше значення функції вартості.

$$\sum_{i=1}^{N} 2^{-i} H(sbox)_{l-i} \tag{2.7}$$

де sbox – сам S-бокс, H – вектор з нульовим індексом.

Таблиця 2.20 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції PCF .

N	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Time		**-**	**-**	**-**	**-**	**212,53**	**91,39**	**88,86**	**100,76**	**42,18**
N	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
Time	**46,92**	**39,35**	**40,77**	**39,83**	**42,31**	**37,47**	**36,58**			

Таблиця 2.21 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції PCF.

N	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Iterat		**-**	**-**	**-**	**-**	**15000**	**6450**	**5566**	**5700**	**1566**
N	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
Iterat	**1900**	**1366**	**1466**	**1400**	**1575**	**1233**	**1133**			

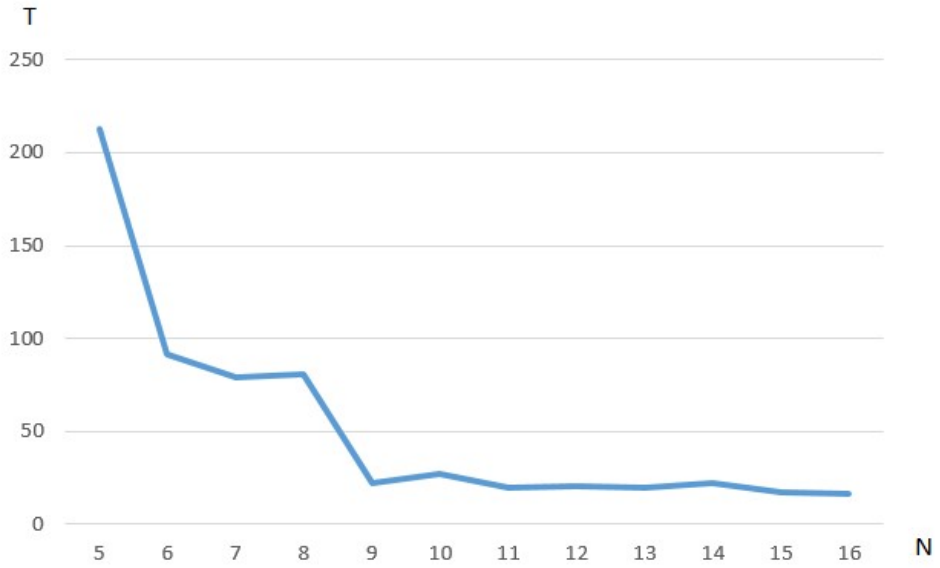

Рисунок 2.8 – Середньостатистичний час який витрачено, до знаходження цільового S-боксу при використанні функції PCF при різних параметрах *N*.

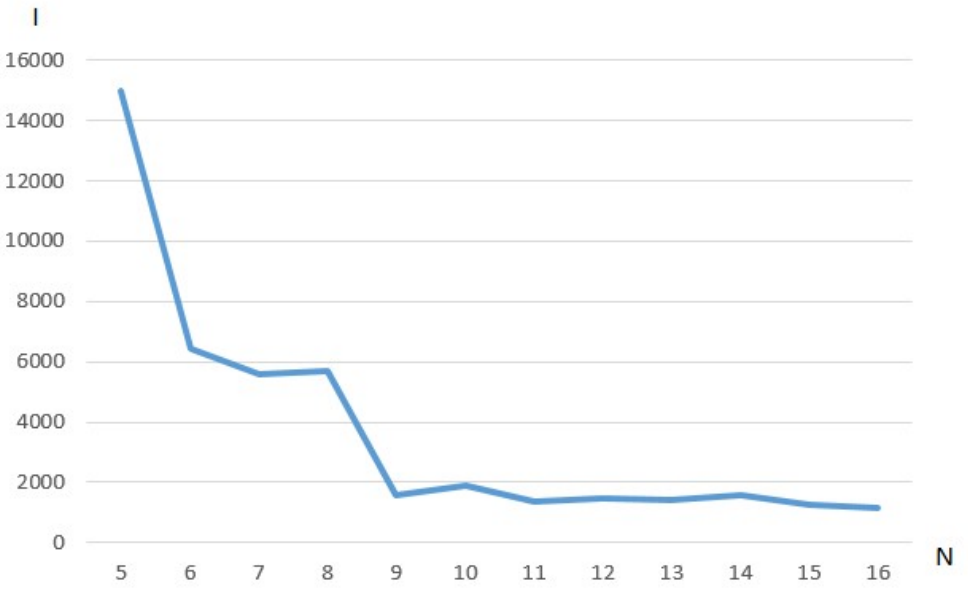

Рисунок 2.9 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції PCF при різних параметрах *N*.

Графічні відображення результату показано на Рисунку 2.8 та 2.9. Зі збільшенням параметру *N* зменшується час та кількість ітерацій за які знаходиться шуканий S-бокс з потрібною нелінійністю. Однак подальше збільшення параметру більше 16 не дає результату зовсім.

В порівнянні з роботою Пичека, а саме з роботою генетичного алгоритму, вдалося отримати результат у 104 набагато швидше. Результати роботи за Пичеком наведено у Таблиці 2.22. Як стверджує автор, такий результат було досягнуто з параметрами *N*=1 та *N*=10. В даній роботі при N=1,2,3,4 не було отримано жодного результату.

Таблиця 2.22 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції PCF за Пичеком.

Алгоритм/Нелінійність	98	100	102	104
GA	473	6184	24963	741371

В порівнянні з дослідженнями Alejandro[17], а саме з роботою генетичного алгоритму, так само вдалося отримати результат у 104 набагато швидше. Результати роботи за Alejandro наведено у Таблиці 2.23.

Таблиця 2.23 – Середньостатистична кількість ітерацій, які було виконано, до знаходження цільового S-боксу при використанні функції PCF за Alejandro.

Метод		Нелінійність		
Алгоритм	Функція	100	102	104
GA	PCF	6148	24963	741371
GaT	PCF	751	4519	167451

Проведемо додаткові дослідження, збільшивши кількість початкової популяції та нащадків. Результати відображено у Таблиці 2.24 у порівнянні з початковими параметрами популяції та кількості нащадків у 100 та 10 відповідно.

Було досліджено можливість формування високонелінійних бієктивних S-боксів за допомогою простого за реалізацією генетичного алгоритму зі застосуванням цільової функції PCF.

В якості оптимальних параметрів функції PCF було визначено $N = 16$. При оптимальних параметрах та використанні генетичного алгоритму пошуку з ймовірністю близької до 99% може бути сформований бієктивний S-бокс з нелінійністю 104.

Середньостатистична кількість ітерацій алгоритму пошуку, при оптимальних параметрах функції ЗCF, складає 1133, що становить близько 36 секунд, або приблизно 32,28 мс. на одну ітерацію.

Таблиця 2.24 – Порівняльна таблиця, до знаходження цільового S-боксу зі збільшенням початкової популяції та нащадків.

Популяція (нащадків)	Середній час пошуку S-боксу (с.)	Середня кількість ітерацій	Середній час на одну ітерацію (мс.)	Середня кількість порахованих S-боксів
50 (5)	26,67	2757	9,6735	137850
100 (10)	36,58	1133	32,286	113300
250 (25)	50,57	1142	44,2819	285500
500 (50)	56,79	742	76,5363	371000
1000 (100)	91,636	571	160,483	571000

Збільшуючи базову популяцію та кількість прямих нащадків вдалося досягти зменшення кількості виконуваних ітерацій, проте збільшився час на пошук цільового S-боксу, за виключенням, коли параметри популяції та нащадків

становили 250 до 25. Погіршення результату за таких параметрів може пояснюватись тим, що роботі алгоритму міг завадити деякий зайвий процес у системі, так як дослідження проводяться на персональному комп'ютері. Якщо порівнювати середній час на виконання одної ітерації найкращі параметри початкової популяції та нащадків становлять 100 до 10, згідно кількості порахованих S-боксів.

2.9 Порівняння

Для проведення порівняння досліджених цінових функцій побудовано гістограму середньостатистичної кількості ітерацій (Рисунок 2.10) та гістограму середньостатистичного часу на виконання ітерацій (Рисунок 2.11). Стовпці гістограми, що знаходяться у значенні 0 означають, що дана цінова функція не досягла шуканого значення, а стовпці, що перевищують значення у 14000 – досягають шуканого значення за більш ніж 14000 ітерацій.

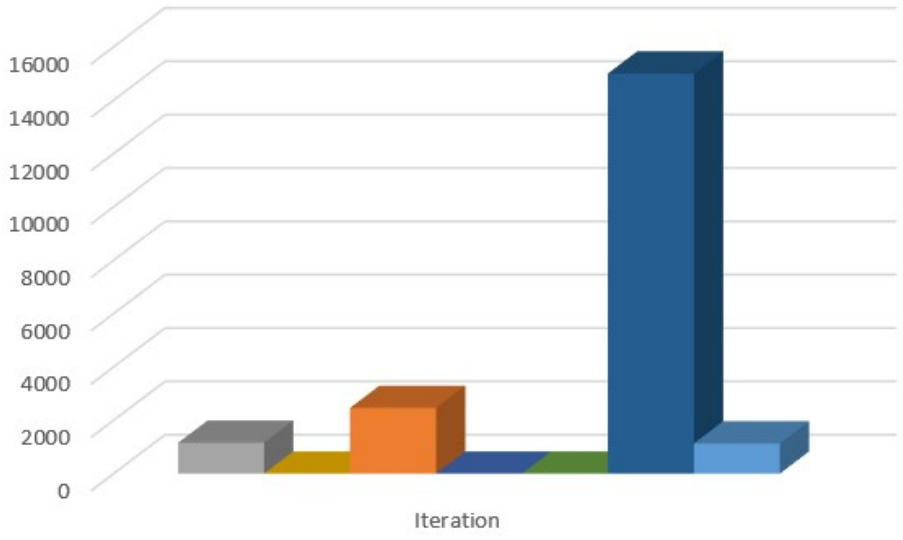

Рисунок 2.10 – Порівняння середньостатистичної кількісті ітерацій, які було виконано, до знаходження цільового S-боксу при використанні розглянутих функцій при оптимальних параметрах для кожної функції.

Рисунок 2.11 - Порівняння середньостатистичного часу, за який було виконано, до знаходження цільового S-боксу при використанні цінових функцій *WCF, WHT, PCF* при оптимальних параметрах для кожної функції.

На Рисунку 2.10 наведено гістограму із середніх значень ітерацій з використанням оптимальних параметрів, які необхідні для досягнення нелінійності у значенні 104. Таким чином, найкращі результати були отримані з функціями:

1) *WCF*, параметри: *start* = 0, *step* = 4, *end* = 32/28/24 (сірий колір);

2) *product_max_wht*, параметри: *start* = 0, *step* = 4, *end* = 32/24 (помаранчевий колір);

3) *PCF*, параметри: *N* = 11/15/16 (голубий колір);

4) *power_max_wht, product_ddt, WHS*: не досягли шуканого результату при вказаних параметрах.

5) *power_ddt*: може досягти шуканого значення, проте це виходить за рамки досліджень.

Найкращий результат між цими трьома функціями показали функції *WCF* та *PCF*. Отримані результати незначною мірою відрізняються між собою, як у кількості виконаних ітерацій, так і в часі. Дані функції у подальших дослідженнях доцільно використовувати, щоб досягти нелінійності в 106.

При параметрах досліджених вище, при яких середньостатистична загальна кількість ітерації менш 2500 спостерігається висока ймовірність знаходження цільового S-блоку. Так, з більш ніж 600 випробувань, лише у одному випадку алгоритм пошуку було зупинено при досягненні граничного значення max_frozen_outer_loops.

2.10 Швидкість пошуку

ОС – Windows 10 Pro 10.0.19044

Тип бази – x64

Процесор – Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz, 2208 МГц, ядер: 6, логічних процесів: 12

Оперативна пам’ять: 32 ГБ

Компілятор: VC++

Налаштування оптимізації (вимкнені /0d):

Розгортання функцій підстановки – за замовчуванням

Ввімкнення функцій підстановки – ні

Пріоритет розміру або швидкості – ні

Волокно-безпечна оптимізація – ні

Оптимізація всієї програми – ні

За таких параметрів системи та за оптимальних параметрів алгоритму середній час на виконання одної ітерації пошуку становитиме приблизно 0,018248 – 0,03132 секунд, або 18,248 – 31,32 мс.

2.11 Додаткові дослідження

Використовуючи оптимальні параметри кожних досліджених цінових функцій було проведено близько 50 запусків алгоритму з кожною із функцій. Метою пошуку ставилась нелінійність у 106. Кожна з використаних функцій не дала результату. Було виконано більше 2,5 мільйона ітерацій у кожному дослідженні. Результат наведено у Таблиці 2.25. NRP – означає, що з цими параметрами не вдалося досягти шуканого значення.

Таблиця 2.25 – Середньостатистична кількість ітерацій до знаходження цільового S-боксу.

	Нелінійність	
Цінова функція	104	106
WCF	**1159**	**NRP**
WHT	**2466**	**NRP**
PCF	**1133**	**NRP**

Зауважмо, що ймовірно іншими алгоритмами, чи ціновими функціями можна досягти бажаного результату. Однак інший алгоритм потребує унікальних налаштувань під обрану цільову функцію та може бути не оптимальним, у зв’язку з чим, буде важко проводити порівняння ефективності (кількості ітерацій) різних цільових функцій.

3 ДОСЛІДЖЕННЯ ВИКОРИСТАННЯ КРОСОВЕРІВ У ГЕНЕТИЧНОМУ АЛГОРИТМІ

3.1 Функція кросовера

Функція кросовера, $cross(p_1, p_2)$, приймає два «батьківських» рішення-кандидати p_1 і p_2 як вхідні дані та виводить «дочірнє» рішення-кандидат o_1 який певним чином поєднує риси обох p_1 і p_2. Зауважимо, що $o_1 = cross(p_1, p_2)$, не обов'язково дорівнює $o_2 = cross(p_2, p_1)$.

Кілька різних алгоритмів кросоверу були розроблені для еволюції біективних функцій (або, будь-якої сутності, яку можна представити як перестановку на наборі цілих чисел), і вважається надзвичайно важливим вибрати хороший алгоритм кросинговеру для проблемної області. У розділі цієї дипломної роботи ми намагаємося розвинути біективні функції над кінцевим полем$GF(2^n)$, порівнюється два різні методи кросоверу; *PMX* («частково відображений кросовер») і циклічний кросовер [14]. Ці два методи кросоверу були обрані через їхню увагу до параметру x у зіставленні з кожним виходом замість того порядку, в якому ці виходи з'являлися.

3.1.1 Циклічний кросовер

Даний кросовер працює наступним чином.

PARENT 1: a b c **d** e f g h i j

PARENT 2: c f a **j** h d i g b e

Рисунок 3.1 – Пара батьків.

Випадково вибрана початкова точка циклу виділена жирним шрифтом, пусті значення в дочірньому елементі позначені як «?».

Елемент *Parent 1* у початковій точці циклу копіюється в дочірній елемент у тій же позиції (Рисунок 3.2).

CHILD: ? ? ? d ? ? ? ? ? ?

Рисунок 3.2 – Дочірній елемент після першого кроку.

Елемент у тій же позиції в *Parent 2* є наступним для копіювання в дочірній елемент, однак він копіюється в ту саму позицію, в якій він зустрічається в *Parent* 1 (Рисунок 3.3).

CHILD: ? ? ? d ? ? ? ? ? j

Рисунок 3.3 – Дочірній елемент після другого кроку.

Цей процес триває, доки процес не поверне нас до вихідної початкової точки циклу - іншими словами, коли створена «петля» або «цикл». В цьому випадку:

(d, j)→(j, e)→(e, h)→(h, g)→(g, i)→(i, b)→(b, f)→(f, d)→(d, j) знову.

CHILD: ? b ? d e f g h i j

Рисунок 3.4 – Дочірній елемент на передостанньому кроці.

Будь-які інші вакантні позиції в дочірньому елементі потім заповнюються шляхом копіювання відповідних значень із *Parent* 2 (Рисунок 3.5).

CHILD: c b a d e f g h i j

Рисунок 3.5 – Кінцевий дочірній елемент.

3.1.2 Кросовер PMX

Цей кросовер починається випадковим вибором двох «точок перетину», як показано вертикальними лініями на Рисунку 3.6. Елементи *Parent 1* між цими точками копіюються в дочірній елемент.

PARENT 1: a b | c d e f | g h i j

PARENT 2: c f | a j h d | i g b e

CHILD: ? ? | c d e f | ? ? ? ?

Рисунок 3.6 – Перший крок створення дочірнього елементу.

Далі будь-які елементи батьківського елемента 2 (*Parent 2*), які ще не були скопійовані в дочірній елемент, копіюються в нього (Рисунок 3.7).

PARENT 1: a b | c d e f | g h i j

PARENT 2: c f | a j h d | i g b e

CHILD: ? ? | c d e f | i g b ?

Рисунок 3.7 – Наступний крок створення дочірнього елементу.

C уже скопійовано з батьківського елемента 1 (*Parent 1*), тому елемент у позиції [0] не може дорівнювати *C*. Ми бачимо, що елемент батьківського елемента 2 (*Parent 2*) у тій же позиції, що й C у батьківському елементі 1, - *A*, і скопіюємо його в позицію [0]. Подібним чином кінцевий елемент не може дорівнювати *E*, тому ми розміщуємо *H* у цій позиції, батьківський елемент 2 у ту саму позицію, що й *E* батьківського елемента 1 (Рисунок 3.9).

PARENT 1: a b | c d e f | g h i j

PARENT 2: c f | a j h d | i g b e

CHILD: a ? | c d e f | i g b h

Рисунок 3.9 – Наступний етап формування дочірнього елементу.

Остаточна нерозподілена позиція складніша. Ми не можемо скопіювати *F*, оскільки він вже присутній у дитини. Ми шукаємо *F* у *Parent* 1 і знаходимо *D* у відповідній позиції *Parent* 2. На жаль, *D* також було скопійовано в дочірній елемент! Далі ми шукаємо *D* у Батьківському 1 і знаходимо, що *J* присутній у тій самій позиції *Parent* 2 і не був скопійований у дочірній, що дозволяє нам завершити процес (Рисунок 3.10).

PARENT 1: a b | c d e f | g h i j

PARENT 2: c f | a j h d | i g b e

CHILD: ? ? | c d e f | i g b ?

Рисунок 3.10 – Завершення формування дочірнього елементу.

3.1.3 Параметри кросоверів

Який би метод кросоверу ми не вибрали, задіяні наступні два параметри:

• *no_of_children*: Коли p_1 і p_2 вибираються з P_i, це визначає, чи буде застосована функція перехресного переходу лише для додавання $o_1 = cross(p_1, p_2)$, до *PCP*, або чи $o_2 = cross(p_2, p_1)$, також буде обчислено та додано.

Якщо функція кросинговеру не застосована, це визначає, що один p_1 або обидва p_1 і p_2 додаються до *PCP*.

• *crossover_probability*: Коли p_1 і p_2 вибираються з P_i під час фази кросоверу це визначає ймовірність застосування функції кросоверу - тобто чи o_1(і o_2, залежно від попереднього параметра), або p_1 і можливо p_2, додаються до *PCP* в цьому поколінні.

Нам також потрібна «функція мутації», *mutate(c)*, взявши варіант рішення з *PCP* як вхідні дані, роблячи невелику випадкову зміну («мутацію») певної форми та повертаючи результат (який замінює оригінал у *PCP*). У наших експериментах, функція мутації робить один хід, як визначено тією ж методологією локального пошуку, що використовується для симуляції відпалу та підйому на пагорб; у випадку еволюції біективних функцій це означає, що два елементи таблиці істинності міняються місцями. Для цього співвідносяться два параметри:

• максимальна кількість можливих мутацій(*max_possible_mutations*): під час «фази мутації» алгоритму це визначає максимальну кількість мутацій, які можуть бути застосовані до будь-якого окремого кандидата.

• ймовірність мутації(*mutation_probability*): кожна потенційна мутація (до кількості, визначеної критерієм вище) виникає випадково з цією ймовірністю, незалежно від інших потенційних мутацій.

Мутація додає аспект дослідження в меметичний пошук, дозволяючи йому уникнути локальних оптимумів.

Третій етап, підйом на пагорб(*hill-climbing*), майже ідентичний алгоритму підйому на пагорб, згаданому раніше. Зауважимо, що оскільки меметичні алгоритми використовують функцію придатності замість цінової функції, алгоритм має бути налаштованим відповідно до цього. На цьому етапі учасники *PCP* всі піднімаються на пагорб до локальних оптимумів щодо зазначеної функції придатності.

Далі, фаза «відбору», яка сама по собі поділяється на різні підфази. Виконавець може вирішити відсортувати елементи *PCP* за їх значеннями придатності для ефективності на початку фази відбору, якщо так, - це сортування є першою підфазою.

Після того, як сортування виконано (чи ні), наступною підфазою є підфаза «елітарності». Якщо параметр *рівень елітарності*(*elitism_level*) має ненульове значення, тоді *elitism_level* учасник з P_i з найвищими значеннями придатності копіюються безпосередньо в P_{i+1}. Якщо це призводить до повної популяції (що не бажано!), етап відбору завершується. Якщо ні, нам потрібно використовувати метод вибору, щоб продовжувати вибирати елементи з *PCP* щоб додати до P_{i+1}.

3.2 Дослідження використання функцій кросоверу

Використовуючи циклічний кросовер, та кросовер PMX в даних дослідженнях не вдалося досягти результатів, які були досягнуті без кросоверу. В Таблиці 3.11 та 3.12 наведено отриманий результат, де і – кількість ітерацій до знаходження цільового S-боксу з нелінійністю у 104, та середній час на одну ітерацію. NRP – означає, що з цими параметрами не вдалося досягти шуканого значення.

Таблиця 3.1 – Порівняльна таблиця, знайдених оптимальних параметрів цінових функцій, з використанням циклічного кросоверу.

	None (i/сер. час на ітерацію (мс.))	Cycle crossover (i / сер. час на одну ітерацію (мс.))							
child_count	-	1	2	3	4	5	6	8	10
WCF (*start* = 0, *step* = 4, *end* = 32)	1159 / 31,57	NRP / 55,66	NRP / 70,94	NRP / 47,74	NRP / 48,19	NRP / 48,01	NRP / 48,10	NRP / 47,68	NRP / 47,49
WCF (*start* = 0, *step* = 4, *end* = 28)	1180 / 25,8898	NRP / 46,53	NRP / 68,60	NRP / 46,63	NRP / 46,64	NRP / 46,84	NRP / 46,94	NRP / 47,31	NRP / 46,90
WCF (*start* = 0, *step* = 4, *end* = 24)	1166 / 25,6174	NRP / 45,45	NRP / 66,41	NRP / 45,55	NRP / 45,79	NRP / 45,58	NRP / 45,95	NRP / 47,27	NRP / 47,48
WHT (*start* = 0, *step* = 4, *end* = 32)	2925 / 17,4837	NRP / 30,40	NRP / 45,29	NRP / 29,95	NRP / 30,19	NRP / 30,31	NRP / 31,05	NRP / 29,87	NRP / 30,16
WHT (*start* = 0, *step* = 4, *end* = 24)	2466 / 18,2481	NRP / 31,33	NRP / 45,53	NRP / 31,12	NRP / 30,87	NRP / 31,03	NRP / 30,69	NRP / 31,12	NRP / 30,83
PCF (*N* = 11)	1366 / 28,80	NRP / 33,10	NRP / 48,76	NRP / 34,04	NRP / 33,41	NRP / 33,81	NRP / 34,42	NRP / 33,14	NRP / 33,23

Продовження Таблиці 3.1 – Порівняльна таблиця, знайдених оптимальних параметрів цінових функцій, з використанням циклічного кросоверу.

	None (i/сер. час на ітерацію (мс.))	Cycle crossover (i / сер. час на одну ітерацію (мс.))							
child_count	-	1	2	3	4	5	6	8	10
PCF (N = 15)	1233 / 30,389	NRP / 33,37	NRP / 49,16	NRP / 32,96	NRP / 33,35	NRP / 33,61	NRP / 32,97	NRP / 33,17	NRP / 33,65
PCF (N = 16)	1133 / 32,2859	NRP / 34,80	NRP / 48,88	NRP / 34,86	NRP / 34,37	NRP / 34,53	NRP / 34,71	NRP / 34,40	NRP / 34,23

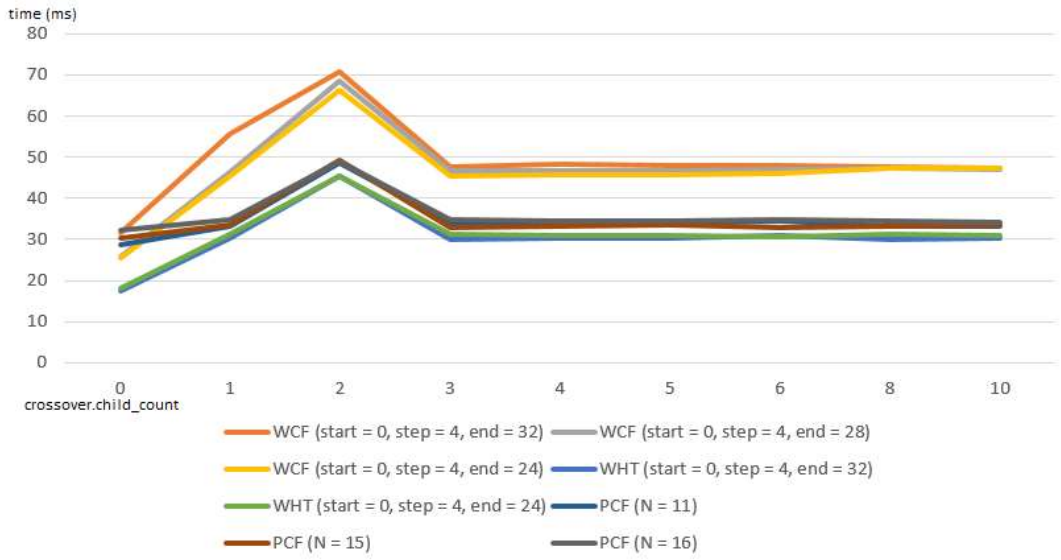

Рисунок 3.11 – Графік зміни часу від змін параметрів циклічного кросоверу.

Таблиця 3.2 – Порівняльна таблиця, знайдених оптимальних параметрів цінових функцій, з використанням PMX кросоверу.

	None (i/time)	PMX crossover (i/time)							
child_count	-	1	2	3	4	5	6	8	10
WCF (*start* = 0, *step* = 4, *end* = 32)	1159 / 31,57	NRP / 50,76	NRP / 77,49	NRP / 49,92	NRP / 50,84	NRP / 49,21	NRP / 48,99	NRP / 49,38	NRP / 49,16
WCF (*start* = 0, *step* = 4, *end* = 28)	1180 / 25,8898	NRP / 46,93	NRP / 68,15	NRP / 46,14	NRP / 46,65	NRP / 46,81	NRP / 46,15	NRP / 47,76	NRP / 46,76
WCF (*start* = 0, *step* = 4, *end* = 24)	1166 / 25,6174	NRP / 44,67	NRP / 65,80	NRP / 44,52	NRP / 44,41	NRP / 44,18	NRP / 46,15	NRP / 45,34	NRP / 46,45
WHT (*start* = 0, *step* = 4, *end* = 32)	2925 / 17,4837	NRP / 30,78	NRP / 43,91	NRP / 29,71	NRP / 30,64	NRP / 29,93	NRP / 29,77	NRP / 29,81	NRP / 30,65
WHT (*start* = 0, *step* = 4, *end* = 24)	2466 / 18,2481	NRP / 30,65	NRP / 45,47	NRP / 29,96	NRP / 30,89	NRP / 30,53	NRP / 30,93	NRP / 31,13	NRP / 31,09
PCF (*N* = 11)	1366 / 28,80	NRP / 32,47	NRP / 47,86	NRP / 33,01	NRP / 32,61	NRP / 32,23	NRP / 32,24	NRP / 32,09	NRP / 32,57
PCF (*N* = 15)	1233 / 30,389	NRP / 32,41	NRP / 48,90	NRP / 32,27	NRP / 32,43	NRP / 32,15	NRP / 31,99	NRP / 33,01	NRP / 32,72

Продовження Таблиці 3.2 – Порівняльна таблиця, знайдених оптимальних параметрів цінових функцій, з використанням PMX кросоверу.

	None (i/time)	PMX crossover (i/time)							
child_count	-	1	2	3	4	5	6	8	10
PCF (N = 16)	1133 / 32,2859	NRP / 33,34	NRP / 49,24	NRP / 32,60	NRP / 32,45	NRP / 32,11	NRP / 32,31	NRP / 32,66	NRP / 32,61

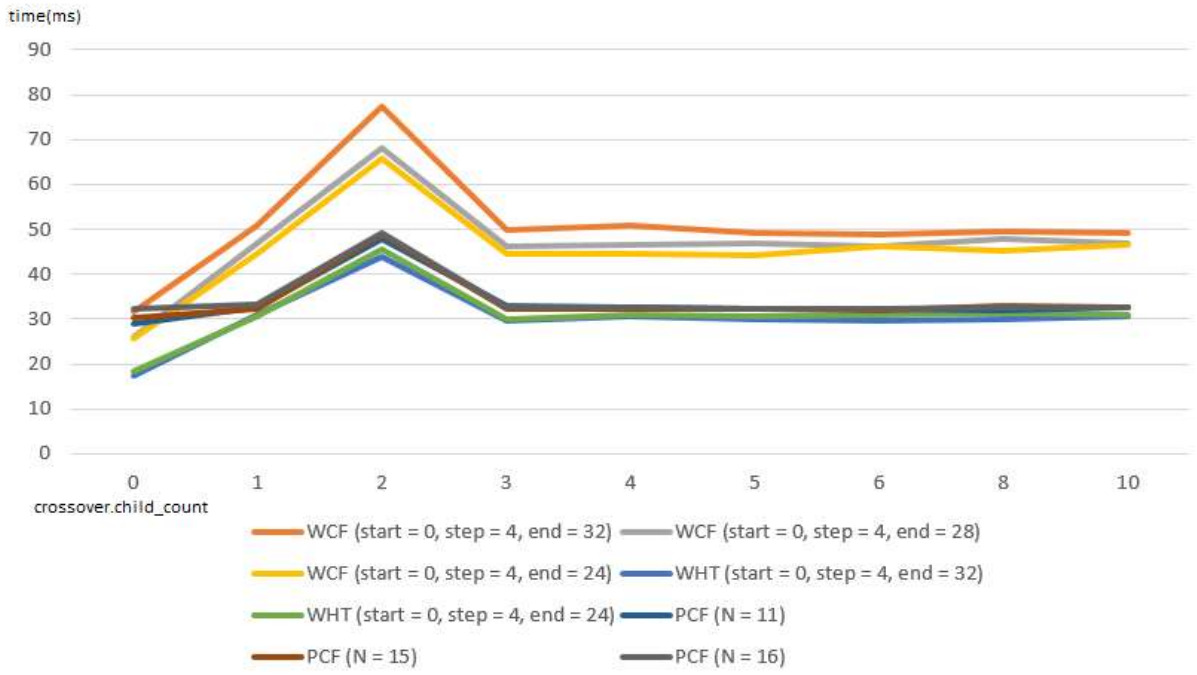

Рисунок 3.12 – Графік зміни часу від змін параметрів PMX кросоверу.

Як видно з отриманих результатів, використання циклічного кросоверу та кросоверу PMX не вдалося знайти такий S-бокс, щоб його нелінійність дорівнювала 104. Також використання кросоверів збільшило час виконання

однієї ітерації. Найбільший стрибок у часі відбувся, коли параметр child_count дорівнював 2. У мілісекундах для цінової функції PMX незначне в той час, як для функцій WCF та WHT збільшення досить значне. Незначне збільшення часу виконання однієї ітерації при великій кількості ітерацій може дуже значно збільшити час на пошук потрібної нелінійності, тому можна зробити висновок, що в даній реалізації і дослідженнях, використання функцій кросоверу не доцільне.

Тож реалізація генетичного алгоритму (GA) з методом отримання нащадків шляхом випадкової кількості перестановок, що описаний вище, дає кращий результат.

ВИСНОВКИ

У цій роботі було досліджено можливість формування високонелінійних S-боксів за допомогою простого за реалізацією генетичного алгоритму зі застосуванням цільових функцій *WHT*, *DDT*, *PCF*, *WHS, WCF* та проведено порівняння функцій з іншими роботами та між собою. Визначено оптимальні параметри для кожної досліджуваної функції, наведено таблиці та рисунки з результатами досліджень. Досліджена можливість використання функцій кросоверу.

В якості оптимальних параметрів функції *wcf* було визначено *start* = 0, *step* = 4, *end* = 32/28/24; функції *product_max_wht* – *start* = 0, *step* = 4, *end* = 32/24; функції *pcf* – *N* = 11/15/16. При оптимальних параметрах та використанні генетичного алгоритму пошуку з ймовірністю близької до 99% може бути сформований бієктивний S-бокс з нелінійністю 104.

Середньостатистична кількість ітерацій алгоритму пошуку, при оптимальних параметрах функції *wcf*, складає 1159, що становить близько 36 секунд.

Середньостатистична кількість ітерацій алгоритму пошуку, при оптимальних параметрах функції *product_max_wht*, складає 2466, що становить близько 45 секунд.

Середньостатистична кількість ітерацій алгоритму пошуку, при оптимальних параметрах функції *pcf*, складає 1133, що становить близько 36 секунд. Враховуючи можливість розпаралелювання процесу пошуку можливо багатократно (кратність відповідає кількості потоків, що одночасно виконуються) скоротити час пошуку.

Змінюючи кількість нащадків для кожних досліджуваних функцій вдалося зменшити середню кількість ітерацій до знаходження цільової нелінійності, проте збільшився час на виконання однієї ітерації, а отже і час на пошук цільового S-боксу з шуканою нелінійністю.

Так, параметри з кількості батьків та нащадків у значенні 100 до 10 відповідно, з ціновою функцією PCF, вважаються оптимальними, спираючись на кількість обчислених S-боксів.

Для цінових функцій WCF та WHT, оптимальні значення батьків та нащадків становитиме 50 до 5 відповідно. Згідно з результатів дослідження, шукана нелінійність знаходиться за більш короткий час, та з меншої кількості обчислених S-боксів.

Використовуючи функції кросоверу для генетичного алгоритму GA, не вдалось покращити результат, знизити час чи зменшити кількість ітерацій для пошуку. Таким чином випадкові перестановки у S-боксі кращий варіант ніж функції крососверу описаних у розділі 3.

Всі дослідження були виконані на домашньому персональному комп'ютері, параметри якого були згадані. Враховуючи те, що існують набагато потужніші комп'ютери, то, можливо, отримати ще більш кращі результати. Основне навантаження під час обчислень бере на себе центральний процесор.

В роботі досягнута мета, а саме, досліджено генетичний алгоритм (GA) пошуку. Досліджено наступні цінові функції: *WHS*, *WCF*, *WHT*, *DDT*, *PCF*; та вплив їх параметрів на швидкість (кількість ітерацій) формування випадкових бієктивних S-боксів з нелінійністю 104. Знайдено найбільш оптимізовані параметри обраних цінових функцій під генетичний алгоритм (GA). Наведено отримані результати щодо можливості формування такою комбінацією високонелінійних S-боксів. Вказано на середньостатистичну кількість ітерації алгоритму пошуку та середній час на виконання одної ітерації, що у подальшій

роботі сприяє порівнянню інших методів та об’єктивному вибору найкращої комбінації алгоритм – цільова функція.

Щодо знаходження нелінійності у значенні 106 не вдалось досягти результатів за приблизно 2,5 мільйону ітерацій. Можливо більша кількість ітерацій, чи модифікації алгоритму, або цінових функцій приведе до покращення результату. Для 106 необхідно оцінити кількість коефіцієнтів Уолша-Адамара зі значенням 48. Якщо їх кількість вийде довести до нуля, то, можливо, це і буде нелінійність 106.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ivanov G, Nikolov N, Nikova S. Reversed genetic algorithms for generation of bijective s-boxes with good cryptographic properties. Cryptogr Commun 8, 2016. P. 247–276.
2. Freyre-Echevarría A. et al. An External Parameter Independent Novel Cost Function for Evolving Bijective Substitution-Boxes: 11 // Symmetry. Multidisciplinary Digital Publishing Institute, 2020. Vol. 12, № 11. 1896 p. URL: https://doi.org/10.3390/sym12111896 (дата звернення: 05.09.2022).
3. Battiti R, Mauro B, Franco M. Reactive Search and Intelligent Optimization, 2008. Springer Verlag. ISBN 978-0-387-09623-0. Archived from the original on 2012-03-16.
4. Burnett L.D. Heuristic Optimization of Boolean Functions and Substitution Boxes for Cryptography: phd. Queensland University of Technology, 2005.
5. Millan W., Clark A. Smart Hill Climbing Finds Better Boolean Functions. 1997.
6. Kavut S., Yücel M.D. Improved Cost Function in the Design of Boolean Functions Satisfying Multiple Criteria // Progress in Cryptology - INDOCRYPT 2003 / ed. Johansson T., Maitra S. Berlin, Heidelberg: Springer, 2003. P. 121–134.
7. Evolving Nonlinear S-Boxes With Improved Theoretical Resilience to Power Attacks / Freyre-Echevarría A. and others. IEEE Access 8:202728–202737. 2020.

8. Kazymyrov O., Kazymyrova V., Oliynykov R. A Method For Generation Of High-Nonlinear S-Boxes Based On Gradient Descent: 578. 2013.

9. Clark J.A., Jacob J.L., Stepney S. The design of S-boxes by simulated annealing. New Gener. Comput. 2005, 23, 219–231. URL: https://www.researchgate.net/publication/225133656_The_Design_of_S-Boxes_by_Simulated_Annealing (дата звернення: 09.09.2022)

10. Daniel Delahaye, Supatcha Chaimatanan, Marcel Mongeau. Simulated annealing: From basics to applications. Gendreau, Michel; Potvin, Jean-Yves. Handbook of Metaheuristics, 272, Springer, pp.1-35.ISBN 978-3-319-91085-7, 2019, International Series in Operations Research & Management Science (ISOR), 978-3-319-91086-4. URL: https://hal-enac.archives-ouvertes.fr/hal-01887543 (дата звернення 16.09.2022).

11. Clark, J.A.; Jacob, J.L.; Stepney, S. The design of S-boxes by simulated annealing. New Gener. Comput. 2005, 23, 219–231. URL: https://www.researchgate.net/publication/225133656_The_Design_of_S-Boxes_by_Simulated_Annealing (дата звернення 18.09.2022).

12. Picek, S.; Cupic, M.; Rotim, L. A new cost function for evolution of s-boxes. Evol. Comput. 2016, 24, 695–718. URL: http://morgoth.zemris.fer.hr/people/Marko.Cupic/files/2016-p1343-single.pdf (дата звернення 18.09.2022).

13. Tesař, P. A new method for generating high non-linearity S-boxes. Radioengineering 2010, 19, 23–26. URL: https://www.researchgate.net/publication/229026133_A_New_Method_for_Generating_High_Non-linearity_S-Boxes (дата звернення 18.09.2022).

14. A comparison of genetic sequencing operators. In R.K. Belew and L.B. Booker, editors, Proceedings of the 4th International Conference on Genetic

Algorithms (ICGA 1991) / Mathias K. and others. Morgan Kaufmann, July 1991. P. 69–76.

15. Полуяненко М.О. Цільова функція WHS. Оптимізація параметрів цільової функції WHS для формування високонелінійних S-блоків.

16. Попов Ю.Д. Формування високонелілейних S-блоків. Методом генетичного алгоритму з цільовою функцією WCF.

17. Alejandro F.-E., Ismel M.-D. A new cost function to improve nonlinearity of bijective S-boxes. August 2020. URL: https://www.researchgate.net/publication/343699912_A_new_cost_function_to_improve_nonlinearity_of_bijective_S-boxes (дата звернення 21.10.2022).

18. Clark J.A., Jeremy J.L., Stepney S. Searching for cost functions. In Proceedings of the 2004 IEEE Congress on Evolutionary Computation. 2004. P. 1517-1524 URL: https://www-users.cs.york.ac.uk/~susan/bib/ss/nonstd/cec04b.pdf (дата звернення 03.11.2022).

ДОДАТОК 1

Приклади отриманих S-боксів:

1)

0xBA, 0x08, 0x9D, 0x16, 0xFF, 0xF3, 0xCD, 0x73, 0xB1, 0x30, 0xDC, 0x7C, 0x61, 0xAE, 0xB0, 0x72,
0x76, 0x07, 0x77, 0xE1, 0x83, 0x5C, 0x8B, 0x4D, 0x37, 0x28, 0xA8, 0xF1, 0xF8, 0xFB, 0x69, 0x4F,
0xE7, 0x8A, 0x88, 0xF4, 0x1E, 0x64, 0xD3, 0x5D, 0x80, 0x54, 0xF7, 0xA1, 0xC1, 0x3C, 0x34, 0x5B,
0x66, 0xB7, 0x3E, 0xAF, 0xBB, 0x87, 0xE4, 0x99, 0x71, 0xE8, 0xE6, 0x36, 0x62, 0xD4, 0x22, 0x46,
0xD9, 0x12, 0xC6, 0xC4, 0xE0, 0xA0, 0x94, 0x79, 0xB9, 0x58, 0xDE, 0xC0, 0x6A, 0x7E, 0x4C, 0x41,
0x0F, 0x49, 0xEB, 0x8F, 0x70, 0x38, 0x3F, 0xCF, 0x39, 0x06, 0x82, 0x5A, 0xFD, 0x9E, 0x85, 0x3B,
0xD7, 0xA7, 0x7B, 0x6F, 0x9A, 0x35, 0x91, 0x0B, 0xCB, 0xBF, 0xAA, 0xD1, 0x40, 0x90, 0xEA, 0xF6,
0xD0, 0xA6, 0x3D, 0x2D, 0x97, 0x2F, 0x45, 0xCC, 0x74, 0xCE, 0x84, 0x0E, 0x7A, 0xF9, 0x60, 0xE5,
0x95, 0xDB, 0xA4, 0x4E, 0xFC, 0x86, 0x1F, 0x52, 0x47, 0xC9, 0x5F, 0x02, 0x17, 0x4A, 0x68, 0x33,
0x67, 0x7D, 0x1B, 0xE9, 0xFE, 0x53, 0x26, 0xA3, 0x8D, 0x98, 0x20, 0x6C, 0x9B, 0x7F, 0x89, 0xDF,
0xB4, 0xB5, 0xB8, 0xDA, 0xED, 0x03, 0x5E, 0x51, 0x92, 0x09, 0xEC, 0x27, 0xD2, 0x59, 0x0D, 0x6B,
0xAC, 0x2E, 0x11, 0xA2, 0xDD, 0x3A, 0xA9, 0x01, 0x1C, 0xC8, 0xEF, 0x57, 0xFA, 0xBC, 0x05, 0x1D,
0xF5, 0xA5, 0x00, 0xC5, 0x93, 0x55, 0x43, 0x25, 0x15, 0xBE, 0x50, 0x23, 0x24, 0xC3, 0x29, 0x78,
0xC7, 0x18, 0x8C, 0xF0, 0x2C, 0xAD, 0x10, 0xD5, 0x0C, 0x56, 0xB2, 0xCA, 0xE2, 0x9F, 0x1A, 0x8E,
0x75, 0x14, 0xF2, 0xBD, 0x21, 0x42, 0x9C, 0xB3, 0x4B, 0x0A, 0x81, 0xEE, 0xD6, 0xE3, 0x04, 0x6E,
0x13, 0x63, 0xC2, 0x19, 0x65, 0xD8, 0xB6, 0x6D, 0x2A, 0x96, 0x48, 0xAB, 0x31, 0x44, 0x32, 0x2B,

2)

0xB2, 0x56, 0x94, 0xB3, 0x8A, 0x24, 0x42, 0x3B, 0x51, 0x72, 0x2A, 0x04, 0x99, 0x9C, 0xE8, 0x79,
0xE4, 0x00, 0x80, 0x22, 0x38, 0xA1, 0x9D, 0xCF, 0x70, 0xB9, 0x1F, 0xB4, 0x2E, 0xDD, 0x3C, 0x7B,
0x21, 0x8B, 0x89, 0x18, 0x60, 0xD8, 0x5B, 0x77, 0x3F, 0xBF, 0x03, 0xF8, 0xDB, 0xD7, 0x95, 0x76,
0x4D, 0xC8, 0x98, 0xE6, 0x35, 0x66, 0xAB, 0x7F, 0x23, 0xFF, 0x6A, 0xAF, 0x65, 0xE5, 0xC9, 0x7D,
0xE9, 0x09, 0xF6, 0xE7, 0x08, 0x47, 0x26, 0x9B, 0x81, 0x6B, 0x9E, 0xF4, 0x49, 0xF1, 0xA4, 0x7E,
0xEC, 0xDC, 0x88, 0xB7, 0x4E, 0xEA, 0x12, 0x45, 0x6C, 0xD6, 0x63, 0xA3, 0x0F, 0xEF, 0xA0, 0xD5,
0x53, 0xAA, 0xA7, 0x55, 0x0B, 0xD3, 0xED, 0x69, 0x1E, 0x74, 0x64, 0x4A, 0x10, 0xBA, 0x1B, 0x8C,
0x9A, 0xF9, 0x2C, 0xE3, 0x84, 0xFB, 0x91, 0x37, 0xC1, 0x8F, 0x13, 0xBB, 0x40, 0x28, 0x50, 0x54,
0xD9, 0x78, 0xC7, 0x0D, 0x68, 0x3D, 0x14, 0x48, 0xA9, 0x5F, 0x58, 0x3E, 0x0E, 0xAE, 0x41, 0x8E,
0x52, 0xBD, 0x07, 0xC4, 0x67, 0x86, 0x4B, 0xB6, 0x71, 0x59, 0x17, 0xB8, 0xC3, 0x1D, 0x39, 0xF2,
0xB1, 0xA2, 0xCC, 0x2B, 0x92, 0x90, 0xDF, 0xCD, 0xFE, 0xFD, 0x61, 0x5D, 0xBE, 0x0A, 0xC5, 0xE2,
0xD0, 0xAC, 0xA8, 0xE0, 0xB0, 0x31, 0xC6, 0xF3, 0x87, 0x62, 0x73, 0x5E, 0x96, 0xFC, 0x25, 0x33,
0x36, 0x46, 0xF5, 0x6F, 0xDA, 0xB5, 0x16, 0x11, 0x27, 0xC2, 0x44, 0xD2, 0x2F, 0x85, 0x1A, 0xA6,
0x57, 0x29, 0xCB, 0x5A, 0x20, 0x43, 0x2D, 0x3A, 0xCA, 0x6D, 0x1C, 0x4F, 0xD4, 0xCE, 0x5C, 0x32,
0x01, 0x4C, 0xE1, 0xF7, 0x93, 0x19, 0x6E, 0xFA, 0x02, 0x75, 0xDE, 0xEE, 0xA5, 0x7A, 0xAD, 0xEB,
0x8D, 0x05, 0x82, 0x06, 0x0C, 0x30, 0x15, 0x34, 0x7C, 0x83, 0x9F, 0xC0, 0xD1, 0x97, 0xF0, 0xBC,

3)

0x91, 0xA0, 0x75, 0x1D, 0x2D, 0x5E, 0x4C, 0x9B, 0xA8, 0xE5, 0xE6, 0x1C, 0x85, 0x9A, 0x64, 0xCF,
0x0A, 0xA7, 0xF3, 0x9C, 0xAC, 0x6D, 0x5A, 0x49, 0xBD, 0x76, 0xE9, 0xDE, 0x71, 0xBE, 0xAF, 0xF2,
0x83, 0x10, 0x09, 0xD3, 0x86, 0xC3, 0xB1, 0xDD, 0x4F, 0xF7, 0x47, 0xB5, 0xDB, 0x67, 0x08, 0xBC,
0xAB, 0x73, 0x31, 0x36, 0x97, 0x92, 0x39, 0x7E, 0x8B, 0xC8, 0x05, 0xDC, 0x65, 0x53, 0xF9, 0xE4,
0x87, 0x2B, 0xEA, 0x66, 0x9F, 0x41, 0x69, 0xA1, 0xF0, 0x1E, 0xFE, 0x96, 0xD1, 0x01, 0xC0, 0x95,
0x7C, 0xCD, 0x34, 0xF8, 0x72, 0x6F, 0xF1, 0x17, 0x8F, 0xB9, 0x4E, 0x0D, 0x44, 0xEB, 0x27, 0x7B,
0x23, 0x6B, 0x12, 0x0C, 0x6E, 0xD5, 0x29, 0xF5, 0x51, 0x18, 0xA2, 0x2E, 0x32, 0xEF, 0x93, 0x19,
0xBB, 0xAE, 0x45, 0xED, 0x5D, 0x89, 0x48, 0x20, 0xE0, 0x52, 0x43, 0x63, 0x2F, 0x1A, 0xCE, 0x5C,
0xFD, 0x42, 0xBA, 0x3E, 0x90, 0x8E, 0x7D, 0x59, 0x80, 0x70, 0x2C, 0x84, 0x6C, 0xB6, 0x25, 0xC7,
0xB3, 0xB2, 0x07, 0xF4, 0xD6, 0x3C, 0x9E, 0x02, 0xCC, 0xEC, 0x62, 0xAD, 0x13, 0x0E, 0x0B, 0x8C,
0xCA, 0x55, 0xC1, 0xD0, 0xAA, 0xD2, 0x37, 0xD4, 0xC2, 0xFF, 0xA4, 0x79, 0xE8, 0x38, 0x8D, 0xE3,
0x11, 0x81, 0x78, 0xCB, 0x56, 0x40, 0x0F, 0x04, 0x7A, 0xC4, 0xFA, 0x4B, 0x77, 0xE1, 0x26, 0x82,
0x14, 0x94, 0xF6, 0xA3, 0x16, 0xD9, 0xE7, 0x61, 0x6A, 0xB8, 0x06, 0xC6, 0x68, 0xA6, 0xFB, 0x9D,
0xDA, 0xEE, 0x3D, 0x24, 0x30, 0xD8, 0x1F, 0x58, 0x00, 0xC5, 0x98, 0xFC, 0x74, 0xDF, 0xB0, 0x8A,
0x57, 0x54, 0x33, 0x35, 0xC9, 0x15, 0xA5, 0xBF, 0x22, 0x88, 0xD7, 0xB7, 0xE2, 0x5B, 0x7F, 0x4D,
0x5F, 0x50, 0x28, 0x60, 0x4A, 0x46, 0x99, 0x3A, 0x03, 0x3B, 0x2A, 0xB4, 0x3F, 0x21, 0xA9, 0x1B,

4)

0xC8, 0x96, 0xA4, 0x4B, 0x06, 0xD5, 0x3D, 0x9A, 0xCD, 0x0D, 0xEA, 0xC9, 0xF0, 0x0A, 0x56, 0x40,
0x4E, 0xE8, 0xB1, 0x35, 0x66, 0xA6, 0x74, 0x85, 0x08, 0x3A, 0x53, 0x0B, 0x54, 0x39, 0xE7, 0x3C,
0x91, 0x50, 0x55, 0x2D, 0xF8, 0x93, 0x22, 0xEB, 0xDD, 0x99, 0xF7, 0x95, 0x42, 0xCC, 0xCF, 0xB6,
0xC1, 0x17, 0x1D, 0xDE, 0xC4, 0x4A, 0xC2, 0xE3, 0xA0, 0xF3, 0xED, 0xE2, 0x7B, 0x81, 0xF2, 0x31,
0x12, 0x24, 0xCB, 0xE9, 0x05, 0x82, 0x20, 0x18, 0xF1, 0x8F, 0x8E, 0x6E, 0x94, 0x36, 0xA3, 0x0F,
0xC6, 0xD2, 0x9B, 0xAD, 0x5C, 0x58, 0xB3, 0x7C, 0x11, 0xE6, 0x6B, 0x02, 0x25, 0xC7, 0x34, 0xFF,
0xAA, 0x13, 0x69, 0x0C, 0x5F, 0x76, 0x9D, 0x48, 0x87, 0x4D, 0x77, 0xDC, 0x27, 0x4F, 0xFC, 0x7E,
0x8B, 0x52, 0x41, 0x6F, 0xF4, 0xA7, 0x2C, 0x59, 0xD9, 0x28, 0xB2, 0x23, 0xBE, 0xF9, 0x32, 0x49,
0x8C, 0x2A, 0x9C, 0x6C, 0x1C, 0xE5, 0x72, 0x73, 0x79, 0xAE, 0xFE, 0xEE, 0x97, 0x3B, 0x45, 0x1F,
0xDA, 0x84, 0x71, 0x60, 0xF6, 0xCE, 0xB5, 0xEF, 0x2B, 0xC5, 0x44, 0x5D, 0xD4, 0x88, 0x7A, 0xAB,
0x64, 0x47, 0x86, 0x61, 0x38, 0x9F, 0x5A, 0x65, 0x5B, 0xA1, 0x46, 0xC3, 0xFD, 0x33, 0x3F, 0x43,
0x8A, 0xCA, 0xD6, 0x8D, 0xBB, 0xA9, 0x78, 0x1A, 0x03, 0xD3, 0x98, 0xBA, 0xD7, 0x83, 0x15, 0x26,
0xD1, 0xE0, 0x4C, 0xDF, 0x1B, 0xA2, 0x01, 0x89, 0x04, 0x6D, 0x68, 0x9E, 0x5E, 0xC0, 0x29, 0xF5,
0x92, 0xBD, 0xD8, 0x57, 0x3E, 0xB0, 0x7F, 0x07, 0xFA, 0xD0, 0xE4, 0x37, 0x70, 0x10, 0xA5, 0x0E,
0xAF, 0x21, 0x90, 0xB7, 0xB4, 0x16, 0x63, 0xEC, 0x2E, 0x19, 0xFB, 0x67, 0x80, 0x2F, 0xAC, 0xB9,
0x09, 0x6A, 0x14, 0x7D, 0xE1, 0xB8, 0x62, 0x1E, 0x51, 0x00, 0xA8, 0xBF, 0x30, 0xDB, 0xBC, 0x75,

5)

0x79, 0x1F, 0xA2, 0x37, 0xE0, 0x2F, 0x86, 0xC4, 0x40, 0x13, 0x30, 0x8F, 0x9C, 0x4B, 0xEB, 0xB6,
0x70, 0x43, 0x0E, 0x2D, 0x60, 0xD7, 0xC5, 0xA7, 0x81, 0x54, 0xAC, 0x4C, 0x4A, 0x12, 0xD9, 0x0B,
0xB9, 0x87, 0x2C, 0x31, 0x1E, 0x56, 0x29, 0x8E, 0xFC, 0x3B, 0xC0, 0x4D, 0xF1, 0x44, 0x5F, 0xB5,
0x20, 0x00, 0xF8, 0x75, 0xC6, 0xF4, 0x46, 0x83, 0xE8, 0x9E, 0x88, 0xA1, 0x3E, 0xB4, 0x80, 0xBA,
0xC1, 0xC7, 0x90, 0xB8, 0x5D, 0x53, 0x61, 0x71, 0x78, 0x76, 0xD4, 0x21, 0x6A, 0x3F, 0x0C, 0xA8,
0x01, 0x08, 0xE2, 0x1B, 0xE5, 0x1A, 0x7A, 0xA3, 0x5B, 0x66, 0x17, 0xD2, 0x28, 0x1D, 0x3A, 0x23,
0xCB, 0x6B, 0x4F, 0x9B, 0xCF, 0x2E, 0xA5, 0xC9, 0xD0, 0x62, 0x26, 0xD6, 0xDA, 0x36, 0x6D, 0x2A,
0xFD, 0xDF, 0x59, 0xF5, 0x4E, 0x6C, 0xFF, 0x33, 0xCE, 0x5E, 0xE1, 0x8D, 0xAF, 0xDB, 0xBE, 0x39,
0x99, 0x0D, 0xFB, 0x97, 0x38, 0x49, 0x93, 0xD3, 0x63, 0xBB, 0x58, 0x6E, 0x68, 0xCA, 0x14, 0x05,
0x7E, 0xD8, 0xB7, 0x18, 0xDE, 0x84, 0x25, 0x04, 0xCC, 0x55, 0xB0, 0xAB, 0x9D, 0x57, 0xEE, 0x8C,

0x09, 0x72, 0x11, 0xF9, 0xB1, 0x9A, 0x34, 0x96, 0x50, 0xBF, 0xA6, 0xFA, 0x95, 0x48, 0x22, 0x3D,
0xAA, 0xFE, 0x52, 0x85, 0x1C, 0x45, 0xAD, 0x77, 0xC2, 0xBC, 0xDD, 0x02, 0x8B, 0x91, 0x5C, 0x9F,
0x94, 0xE6, 0x03, 0xA4, 0x8A, 0x47, 0x41, 0xE9, 0x32, 0x7D, 0xED, 0x2B, 0x51, 0xB3, 0xDC, 0xEF,
0xF0, 0x24, 0x3C, 0xD1, 0xEA, 0xC3, 0x74, 0x35, 0x67, 0xF2, 0xD5, 0xF6, 0xBD, 0xE4, 0x15, 0x7F,
0x19, 0x82, 0x98, 0x16, 0x0A, 0xA9, 0x7B, 0x0F, 0x07, 0xC8, 0x27, 0xCD, 0xA0, 0x69, 0xF7, 0x06,
0xE7, 0xE3, 0xF3, 0x65, 0x10, 0x6F, 0x64, 0x5A, 0x7C, 0xAE, 0x89, 0xB2, 0x92, 0xEC, 0x73, 0x42,

6)

0x99, 0xDF, 0x20, 0x9D, 0x17, 0x4F, 0x8B, 0x30, 0x33, 0x8A, 0x5D, 0xE0, 0x59, 0xAB, 0x4E, 0xBB,
0x3D, 0x97, 0xCA, 0xC4, 0xA8, 0x13, 0x9B, 0x23, 0x60, 0x40, 0xE9, 0xFC, 0x18, 0x75, 0xE3, 0x6B,
0x86, 0x01, 0xA9, 0x04, 0xFF, 0x94, 0x19, 0x1D, 0x39, 0xC3, 0x62, 0x0B, 0xB0, 0x74, 0xD5, 0xD4,
0x9F, 0xDB, 0x31, 0xE6, 0xFA, 0xD2, 0x96, 0x1C, 0xB3, 0x4B, 0x7D, 0x8E, 0x50, 0x77, 0x7A, 0x91,
0x16, 0x32, 0xA7, 0xBF, 0xC7, 0x69, 0xAC, 0x1F, 0x2F, 0x35, 0x9C, 0x07, 0x47, 0xE8, 0x27, 0xA6,
0xA3, 0xEC, 0x2E, 0x06, 0xA2, 0x58, 0x78, 0xB1, 0x9E, 0x89, 0xBD, 0xDE, 0xD1, 0xF7, 0x0C, 0xE1,
0xDD, 0x11, 0x81, 0xB9, 0x83, 0xA0, 0xEF, 0x71, 0x08, 0xD9, 0x38, 0x45, 0x6E, 0x7C, 0x1A, 0xD3,
0x34, 0x8D, 0xC8, 0xAF, 0xD0, 0x8F, 0xB8, 0x7E, 0x2B, 0xF8, 0x3A, 0xCE, 0x7B, 0x3B, 0xC6, 0x28,
0xD6, 0xEE, 0x5E, 0xED, 0x5C, 0x8C, 0xA4, 0xD7, 0x3C, 0xE5, 0xF1, 0x02, 0xE7, 0x25, 0xA1, 0xC0,
0xBC, 0x2D, 0xAA, 0x46, 0x95, 0x98, 0xE4, 0x5B, 0x61, 0x6D, 0xC9, 0x0F, 0xCC, 0x10, 0x0D, 0x1E,
0xB7, 0xD8, 0x63, 0xC5, 0xC1, 0xF6, 0xF0, 0x51, 0x88, 0x43, 0x56, 0x84, 0x65, 0x6A, 0x66, 0xBE,
0x53, 0x6F, 0x37, 0x55, 0x92, 0x05, 0x4D, 0xB5, 0x29, 0xFD, 0x24, 0xEA, 0xB4, 0xDA, 0xDC, 0xE2,
0xF5, 0xC2, 0x44, 0x5A, 0x82, 0xFE, 0x76, 0x42, 0x4C, 0x9A, 0x54, 0xF9, 0x90, 0x85, 0x80, 0xCD,
0x4A, 0x2C, 0x14, 0xB6, 0x15, 0x87, 0xAE, 0x79, 0x12, 0x3F, 0x64, 0x00, 0x36, 0x41, 0x68, 0xEB,
0x0A, 0xAD, 0x93, 0x7F, 0xA5, 0x57, 0xFB, 0x26, 0x49, 0x52, 0xF4, 0x5F, 0x03, 0xF3, 0xBA, 0x09,
0xCF, 0x70, 0xCB, 0x48, 0xB2, 0x21, 0x3E, 0x1B, 0x67, 0xF2, 0x6C, 0x22, 0x2A, 0x73, 0x72, 0x0E,

7)

0x3C, 0x1D, 0xE6, 0xD0, 0x4C, 0x61, 0xCF, 0x08, 0x8D, 0x79, 0x01, 0x38, 0xE5, 0xD5, 0xE7, 0x76,
0x7F, 0x87, 0x6F, 0xFD, 0xC1, 0x04, 0x1F, 0x73, 0xC5, 0x11, 0x68, 0x1E, 0x28, 0x82, 0xE8, 0x0B,
0xCB, 0x65, 0x2D, 0xD1, 0xC8, 0xE9, 0x07, 0x35, 0xB2, 0x93, 0xF2, 0xC4, 0xD8, 0xD6, 0xD9, 0x51,
0xF4, 0x2B, 0x21, 0xF1, 0x66, 0x6B, 0xA1, 0x49, 0x15, 0x81, 0xF6, 0x8A, 0xAB, 0xA7, 0x84, 0xB6,
0xF5, 0x13, 0x0C, 0x62, 0xD3, 0xF9, 0x60, 0x57, 0x4F, 0xD4, 0xEB, 0xD2, 0x3A, 0x1C, 0x6E, 0xB9,
0x86, 0x9F, 0x2C, 0xCE, 0xA6, 0x31, 0x50, 0xB4, 0x92, 0x4E, 0xB8, 0xEF, 0x33, 0x36, 0x7A, 0x72,
0x5E, 0x34, 0x59, 0xEC, 0x26, 0xF0, 0xA9, 0xCC, 0xB1, 0x7E, 0x98, 0x14, 0x96, 0xE3, 0x39, 0x9C,
0x67, 0x4D, 0x69, 0x23, 0x30, 0x8F, 0xBC, 0xEE, 0x25, 0x10, 0x02, 0x9D, 0x5C, 0x0D, 0x2F, 0xA2,
0x43, 0xBE, 0x06, 0x29, 0x95, 0x89, 0x9A, 0x1A, 0xFB, 0xAD, 0xAF, 0x99, 0xCD, 0xEA, 0xE0, 0x20,
0x5D, 0x54, 0x71, 0x37, 0x19, 0x18, 0x4B, 0x64, 0xDC, 0xB0, 0x6D, 0x63, 0xFF, 0x2E, 0xD7, 0x03,
0x2A, 0xDB, 0xE4, 0x78, 0x4A, 0x55, 0x16, 0x97, 0xF8, 0xC0, 0xBB, 0xA8, 0x05, 0xC9, 0x53, 0x22,
0x3B, 0xFC, 0xDA, 0x47, 0x8C, 0x56, 0xA3, 0x40, 0x3D, 0x7D, 0x44, 0x83, 0xC6, 0x94, 0xE2, 0xBD,
0x0A, 0x88, 0x77, 0x5F, 0xED, 0x75, 0x12, 0xBA, 0xF3, 0x5B, 0x6A, 0xDD, 0xE1, 0x00, 0x90, 0x5A,
0x80, 0x58, 0xC2, 0x3E, 0xBF, 0x41, 0x6C, 0xB7, 0x17, 0x9B, 0x3F, 0x7B, 0xAC, 0x24, 0x48, 0xC3,
0x74, 0x85, 0xB5, 0xA0, 0x09, 0x1B, 0xCA, 0xF7, 0x8B, 0xDF, 0xC7, 0x0F, 0x52, 0x45, 0x91, 0xA5,
0x9E, 0x27, 0x32, 0xFA, 0x8E, 0xDE, 0x42, 0xA4, 0xB3, 0x46, 0xFE, 0x70, 0x0E, 0xAA, 0x7C, 0xAE,

8)

0x59, 0x51, 0xFC, 0x73, 0x5E, 0x50, 0x64, 0xB4, 0xE2, 0xB1, 0x60, 0xBB, 0x79, 0xB7, 0x41, 0x1A,
0x22, 0x34, 0x25, 0x04, 0x3B, 0x37, 0x2C, 0xE9, 0x89, 0x1F, 0x58, 0xF9, 0xB0, 0x92, 0x31, 0xDB,
0xA5, 0x5C, 0x67, 0x4E, 0x08, 0x27, 0x16, 0xE7, 0x44, 0x1B, 0x9A, 0xEA, 0x6A, 0x7F, 0x2F, 0xAA,
0x53, 0x63, 0xEF, 0x3A, 0x5D, 0x32, 0x01, 0x8E, 0x14, 0x75, 0x80, 0x61, 0x8B, 0x4A, 0x6C, 0x0A,

0xE8, 0x6F, 0xAF, 0x40, 0x95, 0x77, 0xE3, 0x29, 0x56, 0x70, 0x96, 0xA9, 0x43, 0xD8, 0x84, 0xFE,
0x49, 0xB3, 0x62, 0x05, 0xAB, 0xFD, 0x91, 0xC3, 0xCE, 0x0C, 0x8F, 0x03, 0xD5, 0xF8, 0x9D, 0x45,
0x52, 0xF6, 0x4B, 0x9E, 0xE6, 0x81, 0x74, 0x9B, 0x7C, 0x68, 0xFF, 0x11, 0x5F, 0xC4, 0x5B, 0x9C,
0x3F, 0x7B, 0x33, 0x38, 0xBC, 0x1C, 0x97, 0xF3, 0x4F, 0xC9, 0xA1, 0xD2, 0x88, 0x8C, 0xC8, 0x4D,
0xD9, 0xA7, 0x2D, 0xF7, 0x54, 0xC6, 0xFB, 0x98, 0xCA, 0x9F, 0xB8, 0x12, 0x7A, 0x7E, 0xCC, 0xC0,
0x4C, 0xD7, 0x2E, 0xD0, 0xF1, 0xF0, 0x06, 0x35, 0x78, 0x36, 0xD3, 0x6D, 0xC7, 0xEC, 0xED, 0xCF,
0xE1, 0xBA, 0xF4, 0x13, 0xDF, 0xDD, 0x0E, 0x8A, 0xA6, 0x48, 0xEE, 0x90, 0x76, 0x5A, 0xA8, 0xA0,
0x26, 0x20, 0x47, 0x71, 0xAC, 0x6E, 0xDA, 0x17, 0x87, 0xD4, 0xE4, 0xD1, 0xBD, 0x23, 0x3D, 0xCB,
0x21, 0x02, 0x55, 0x0B, 0xA2, 0x18, 0x2A, 0x57, 0x19, 0x00, 0x39, 0xB5, 0x42, 0xFA, 0x24, 0xF2,
0xBE, 0x99, 0x1E, 0xC2, 0x1D, 0xE0, 0xB9, 0x28, 0xCD, 0xDC, 0x7D, 0x65, 0xE5, 0xF5, 0x69, 0x86,
0xAE, 0x30, 0x66, 0x8D, 0x2B, 0x93, 0x07, 0x10, 0xC1, 0x83, 0xBF, 0x09, 0xDE, 0xB6, 0xA3, 0x85,
0x46, 0x72, 0x0D, 0x94, 0x15, 0xB2, 0x3C, 0x82, 0x3E, 0xC5, 0x0F, 0xD6, 0xA4, 0xEB, 0x6B, 0xAD,

9)

0x4B, 0x70, 0x1F, 0x2D, 0x8F, 0x4C, 0x61, 0x85, 0xAA, 0x72, 0x58, 0x93, 0x11, 0xCB, 0x49, 0xCF,
0x88, 0x5E, 0x31, 0x34, 0x7E, 0x65, 0x08, 0x66, 0x39, 0x1D, 0x0F, 0xEC, 0x82, 0xAE, 0x54, 0xA0,
0x91, 0x69, 0xBA, 0x80, 0x84, 0xC8, 0x4A, 0x98, 0xDD, 0xE1, 0xF4, 0x57, 0x0B, 0x9C, 0x7B, 0xD1,
0x09, 0xBB, 0x03, 0x73, 0x94, 0x8B, 0xD7, 0xFF, 0x04, 0x14, 0x4F, 0xF7, 0x13, 0x50, 0x6B, 0x6E,
0x45, 0x62, 0x76, 0x3C, 0xB1, 0xBE, 0x7C, 0xF0, 0x81, 0x52, 0x0A, 0xE5, 0x3B, 0xCC, 0x23, 0xD4,
0xD0, 0xB2, 0xA3, 0xCA, 0xC4, 0x3F, 0xE9, 0x9F, 0x25, 0xA2, 0xF2, 0x5D, 0x51, 0xE0, 0xC5, 0x87,
0x5A, 0x3E, 0x8A, 0xB4, 0x33, 0x9D, 0x2C, 0x22, 0xB5, 0x89, 0xA5, 0x7D, 0xF3, 0xEE, 0x36, 0x79,
0xA6, 0xFC, 0x18, 0x53, 0x32, 0x35, 0xBF, 0x15, 0x9B, 0x8E, 0xAD, 0x64, 0x55, 0xAB, 0xED, 0x21,
0x1B, 0xD5, 0x60, 0x47, 0x01, 0xDC, 0xCD, 0x9A, 0x5F, 0xC0, 0xD3, 0x17, 0xD2, 0x2A, 0x20, 0x0C,
0x92, 0xF1, 0x67, 0xE4, 0xBD, 0xF8, 0xB7, 0x27, 0xF9, 0x3A, 0x71, 0x9E, 0x12, 0xDE, 0xC2, 0x38,
0x78, 0x48, 0x86, 0xE8, 0xDF, 0xA1, 0x99, 0xAF, 0xB8, 0xFA, 0x77, 0x6C, 0x97, 0xAC, 0xFD, 0xB3,
0x6A, 0x00, 0x1C, 0xEA, 0xF6, 0xB0, 0x19, 0xA4, 0xC6, 0x30, 0x0E, 0x7A, 0x74, 0x4D, 0xC1, 0xE2,
0x8C, 0x43, 0x29, 0x6D, 0xE7, 0xD8, 0x3D, 0xB9, 0x02, 0x1A, 0xF5, 0x16, 0x06, 0x2F, 0x2B, 0x75,
0x05, 0x0D, 0x5B, 0xE3, 0x26, 0xFE, 0x41, 0xFB, 0x95, 0x07, 0x46, 0x59, 0x56, 0x42, 0x24, 0x2E,
0xD6, 0xE6, 0x8D, 0x63, 0x68, 0xEF, 0xBC, 0xEB, 0x5C, 0xC7, 0xCE, 0x90, 0x6F, 0x96, 0x44, 0xC9,
0xD9, 0x40, 0x7F, 0x28, 0xA7, 0xDB, 0xDA, 0x37, 0xA8, 0x4E, 0xB6, 0x1E, 0xA9, 0x83, 0xC3, 0x10,

10)

0xBD, 0x0A, 0x84, 0xAD, 0x00, 0xF0, 0x95, 0x27, 0xFC, 0xA4, 0xF1, 0x93, 0x7F, 0xBC, 0x55, 0x30,
0x41, 0xFF, 0xB7, 0xCA, 0xD9, 0xA8, 0xE7, 0x9C, 0x48, 0xF8, 0xC3, 0xAB, 0xCE, 0x14, 0x79, 0x4A,
0x0F, 0x90, 0x76, 0x1D, 0x2C, 0x63, 0x49, 0x51, 0x3A, 0x12, 0x16, 0xAF, 0x1A, 0xA6, 0x62, 0xCC,
0x59, 0xB5, 0x33, 0x02, 0x65, 0xF6, 0xA5, 0x31, 0x70, 0x19, 0x9B, 0x52, 0xB6, 0x5C, 0xEE, 0x2D,
0x6E, 0xF9, 0x8F, 0x1E, 0x9E, 0xA3, 0x29, 0x2B, 0xCB, 0x7D, 0x83, 0x54, 0x2E, 0x8A, 0x08, 0x0D,
0xE5, 0x5B, 0xAE, 0x68, 0x58, 0x39, 0xD5, 0xDC, 0x47, 0xF5, 0x26, 0x25, 0x4D, 0x73, 0x94, 0x0E,
0x72, 0x44, 0xD8, 0xB1, 0xBE, 0x4F, 0x98, 0x78, 0xE9, 0x01, 0x91, 0xCD, 0xE8, 0x7C, 0x53, 0xC8,
0x6D, 0x3C, 0x38, 0x75, 0xD1, 0x57, 0xBF, 0xAC, 0x23, 0x1B, 0xE4, 0x4C, 0xA2, 0xDE, 0xA9, 0xD6,
0x9D, 0x3D, 0xD7, 0xEA, 0x03, 0x1C, 0xC0, 0x09, 0xA1, 0x60, 0x18, 0x32, 0x07, 0xED, 0x77, 0xAA,
0x6B, 0x20, 0x6C, 0x22, 0x3E, 0x5D, 0x86, 0xEB, 0x13, 0x10, 0x1F, 0xB9, 0x7A, 0x64, 0x96, 0x87,
0x34, 0x35, 0xE1, 0xC7, 0xC1, 0xF4, 0x6F, 0x46, 0xB4, 0xC6, 0x11, 0xDF, 0xDD, 0x99, 0x40, 0x8B,
0xFA, 0x15, 0xBB, 0x8C, 0x7E, 0x45, 0x4B, 0xEC, 0x85, 0x0B, 0x3F, 0x9F, 0x2A, 0x17, 0xC9, 0x43,
0xF3, 0x88, 0xFE, 0xD3, 0x28, 0x80, 0x06, 0x56, 0xE6, 0x61, 0x37, 0x82, 0xB8, 0x5F, 0x5A, 0x4E,
0xFB, 0xE2, 0x92, 0x66, 0x42, 0x21, 0x5E, 0xA7, 0x67, 0xC5, 0xD4, 0x2F, 0xBA, 0x0C, 0xC4, 0xE0,
0x3B, 0x24, 0x81, 0x50, 0x7B, 0xB2, 0x8E, 0xEF, 0x6A, 0x71, 0xF2, 0xB3, 0x8D, 0xD0, 0xDB, 0xC2,
0x89, 0xF7, 0x05, 0xD2, 0xE3, 0xDA, 0x04, 0x9A, 0xA0, 0x74, 0x97, 0xFD, 0xCF, 0x69, 0xB0, 0x36,