УДК 004.056.5 **DOI:10.30837/rt.2021.3.206.06**

О.О. КУЗНЕЦОВ, д-р техн. наук, М.О. ПОЛУЯНЕНКО, канд. техн. наук,
С.Л. БЕРДНИК, канд. техн. наук, С.О. КАНДІЙ, Ю.О. ЗАИЧЕНКО

ОПТИМІЗАЦІЯ ПАРАМЕТРІВ АЛГОРИТМУ ЛОКАЛЬНОГО ПОШУКУ ДЛЯ ГЕНЕРАЦІЇ НЕЛІНІЙНИХ ПІДСТАНОВОК

1. Вступ

Проєктування симетричних шифрів передбачає застосування різних криптопримітивів, зокрема і нелінійних таблиць заміни (званих також S-блоками, нелінійними підстановками, вузлами ускладнення, тощо) [1, 2]. Від криптографічних показників S-блоків залежить ефективність симетричних шифрів, зокрема їх стійкість до різних криптоаналітичних атак [3 – 5]. Отже, генерація нелінійних таблиць заміни з потрібними криптографічними показниками є безумовно актуальною та важливою науковою задачею [6 – 9].

В роботі розглядаються алгоритми локального пошуку та проводяться експериментальні дослідження їх ефективності для генерації S-блоків.

Локальний пошук відноситься до евристичних технік вирішення оптимізаційних задач та об’єднує групу обчислювальних алгоритмів, зокрема, сходження на пагорб, імітації відпалу, генетичні алгоритми та ін. [10 – 12]. Локальний пошук використовують для вирішення різних проблем, для яких можна сформулювати критерій пошуку серед безлічі можливих рішень (наприклад, у вигляді спеціальної функції вартості). По суті, всі алгоритми локального пошуку застосовують локальні зміни початкового стану до тих пір, поки не буде знайдено рішення, яке вважається оптимальним, або поки не закінчиться обмеження за часом (кількістю ітерацій) [13].

Розглядаються алгоритми локального пошуку, які застосовують спеціальну функцію вартості для інформованого пошуку підходящих рішень (S-блоків з необхідними криптографічними показниками) у просторі можливих станів (всіх можливих таблиць заміни) [6, 14, 15]. Тобто на кожному кроці через розрахунок функції вартості алгоритм оцінює можливі альтернативи та приймає рішення стосовно подальшого переборного пошуку. В статті застосовується цільова функція з робіт [16, 17] та досліджується її поведінка та вплив на ефективність алгоритму локального пошуку. Проводяться багаточисельні експерименти та евристично обираються оптимальні параметри алгоритму, наводяться оцінки його результативності. Фактично вдалося оптимізувати алгоритм локального пошуку S-блоків з цільовою нелінійністю 104 і отримати середній час генерації (при багаторазовому запуску алгоритму) 33,2 секунди (на ПК з тактовою частотою процесору 3,49 GHz, AMD Rizen 9 3950 X 16, RAM 128 GB, Windows 10). При однократному запуску алгоритму ймовірність знаходження цільового S-блоку становить 21,5 %. Це, на думку авторів, є одним із найкращих результатів для швидкої генерації нелінійних вузлів симетричних шифрів.

2. Пов’язані роботи

Техніки евристичного пошуку нелінійних підстановок розглядалися в багатьох роботах. Наприклад, в ранніх роботах [18–20] досліджувалися евристичні техніки генерації криптографічних булевих функцій, які згодом було поширено на пошук бієктивних S-блоків [6, 7, 14, 21, 22]. Цей напрямок досліджень виявився дуже продуктивним, оскільки вдавалося поступово підвищувати криптографічні показники згенерованих таблиць заміни. Зокрема, у роботах [7, 17, 22 – 24] ті ін. досліджено алгоритми імітації відпалу. Роботи [6, 21, 23, 25, 26] та ін. присвячено вивченню техніки сходження на пагорб. В [27 – 30] досліджено генетичні алгоритми. Найчастіше ставилася задача генерації бієктивних 8×8 S-блоків з найвищою нелінійністю, і поступово вдавалося підвищувати цей показник. Наприклад, в [14, 19, 21] досліджено ефективність випадкової генерації (досягнуто нелінійність 98) та техніки схо-

дження на пагорб (сформовано S-блоки з нелінійністю 100). В роботах [16, 17] застосовано техніку імітації відпалу та досягнуто нелінійність 102. В [15, 27] розглянуто генетичні алгоритми, які у комбінації із іншими методами оптимізації дозволили формувати підстановки з нелінійністю 104. На сьогодні дослідники акцентують свою увагу на функціях вартості [15, 31, 32]. Дійсно, зміна параметрів або форми функції вартості може значно вплинути на ефективність локального пошуку, і це наочно продемонстровано, наприклад, у роботах [15, 31].

В цій роботі розглянуто найпростіший варіант алгоритму локального пошуку і просту форму функцію вартості з ранніх робіт, наприклад, з [7, 16]. Метою роботи є оптимізація параметрів цього варіанту локального пошуку таким чином, щоб швидко формувати S-блоки з високою нелінійністю, тобто коли навіть для простих і добре відомих алгоритмів і функцій вартості вдається досягти показників, які є порівняні із кращими відомими на сьогодні результатами.

3. Опис алгоритму

Алгоритм локального пошуку (англ. – Local Search Algorithm, або також відомий у літературі як алгоритм Монте – Карло) – це ітераційний алгоритм, який починає свій пошук з можливої точки, випадково обраної в просторі станів. Потім послідовно застосовується механізм генерації для пошуку кращого рішення (з точки зору значення цільової функції), досліджуючи сусідство поточного рішення. Якщо знайдено краще рішення, воно стає поточним рішенням. Алгоритм закінчується, коли не вдається знайти покращення, а поточне рішення розглядається як приблизне рішення задачі оптимізації.

Алгоритм локального пошуку оптимізує цільову функцію, досліджуючи сусідні точки рішення відносно поточної точки в просторі рішень. У наступних визначеннях розглянемо (S, f) приклад комбінаторної задачі оптимізації (де S – набір можливих рішень; f – цільова функція, яку слід мінімізувати).

Нехай N – програма, яка визначає для кожного розв'язку $i \in S$ підмножину $S_i \subset S$ рішень, «близьких» (близькість визначається користувачем) відповідно до задачі i. Вважаємо, що N – структура сусідства, що визначена (S, f).

Механізм, що породжує наступний стан, – це засіб для вибору рішення j у будь-якому сусідстві S_i поточного рішення i. Конкретна методика, яка застосовується для зміни рішення, залежить від задачі, яка вирішується, та подання рішення. Наприклад, якщо рішення представлено у вигляді двійкового рядка фіксованої довжини, тоді підходящим механізмом пропонування можливих нових рішень може бути доповнення одного (або декількох) випадково вибраних значень у бітовому рядку.

Алгоритм локального пошуку (для задачі мінімізації) можна представити узагальнити наступним псевдокодом 1:

Псевдокод 1. Алгоритм пошуку локального мінімуму

1. Вибрати початкове рішення i;
2. Генерувати рішення j із сусідства S_i поточного рішення i;
3. Якщо ($f(j) < f(i)$), то j стає поточним рішенням;
4. Якщо ($f(j) \geq f(i)$) для всіх $j \in S_i$, то закінчити;
5. Перейди до кроку 2.

Розв'язок $i^* \in S$ називається *локальним оптимумом* відносно N для (S, f), якщо $f(i^*) \leq f(j)$ для всіх $j \in S_{i^*}$.

Структура сусідства N *називається точною*, якщо для кожного локального оптимуму щодо N, $i^* \in S$, i^* також є глобальним оптимумом (S, f).

В роботі запрограмований алгоритм локального пошуку, який одночасно виконував пошук в декількох потоках, працюючих паралельно. Кількість потоків вказується у вхідному параметрі thread_count (в нашому випадку thread_count = 30). Алгоритм локального пошуку

починає свою дію з рішення, яке встановлене випадково. Воно ж встановлюється як поточне рішення i. На кожній ітерації циклу утворюється декілька нових рішень j, які генеруються за заданими операторами мутації. Оператор мутації, в свою чергу, обирає випадковим чином k (використовувалось $k=2$) різних позицій у поданому рішенні і надалі переставляє елементи у обраних позиціях.

Всі ітерації алгоритму локального пошуку виконуються у внутрішньому циклі. Ітерації внутрішнього циклу вложено в зовнішній цикл. Зовнішній цикл не є обов'язковим для роботи алгоритму, він був введений для відстеження поточного стану процесу пошуку та оптимізації вибору його параметрів.

Нове рішення порівнюється з поточним. В разі отримання кращого, ніж поточне, рішення воно встановлюється як поточне. В якості цільового S-блоку було обрано S-блок з нелінійністю $N_f = 104$.

4. Трек цільової функції

В якості цільової функції було обрано функцію вартості:

$$WHS = \sum_{i=1}^{255} \left| \left| \max\left(WHT\right) \right| - X \right|^R , \tag{1}$$

де $\max(WHT)$ – максимальне значення спектральних коефіцієнтів Уолша – Адамара S-блоку (WHT – англ. Walsh – Hadamard transform – коефіцієнти Уолша – Адамара);

- X=36, R=4 – обрані параметри для функції вартості.

На рис. 1 наведено приклад роботи алгоритму з формування треку функції вартості (WHS) для перших шести зовнішніх циклів (max_outer_loops=6), десяті внутрішніх циклів (max_inner_loops=10) та чотирьох незалежних (працюючих асинхронно) потоків у кожному циклі (threads_count=4). Для наочності зміни нелінійності S-блоку для кожної функції вартості неведено поточне значення нелінійності S-блоку (N_f). Знаком «+» позначено поточний вибір кращих значень функції вартості.

Першим виконується зовнішній цикл з індексом 0 (відповідні строки з ліво на право позначені «[0] =»). Перше значення, яке обирається за найкраще, є cost=24573952 у четвертому потоці (Tread = 4) при першій ітерації. При цьому нелінійність початкового S-блоку дорівнює $N_f = 88$. Наступне значення обертається також при першій ітерації, але у другому потоці (Tread = 2) та дорівнює: cost=24315904. Першу ітерацію внутрішнього циклу закінчено. При другій ітерації внутрішнього циклу найкраще значення приймає WHS=22753280 у першому потоці, при цьому нелінійність відповідного S-боксу вже становить $N_f = 90$. Наступним краще значення при третій ітерації має третій потік з WHS=21907456 та відповідно нелінійність $N_f = 92$, і так далі. Перший цикл закінчено з кращим значенням WHS=17156352 (Tread = 3) та $N_f = 92$.

При другому зовнішньому циклі (відповідні строки позначені «[1] =») найкраще значення знаходиться у третьому потоці з WHS=16667392 при першій ітерації, а також при другій – WHS=16410112, де значення нелінійності покращується до $N_f = 94$. І так далі.

Наведені ітераційні цикли закінчуються найкращим значенням WHS=7531264 та нелінійністю $N_f = 100$ (Tread = 3, друга внутрішня ітерація). Нагадаємо, що потоки виконуються незалежно, тому попереднє найкраще значення WHS=7968000 було знайдено при четвертій ітерації другого потоку, який фактично (в часовому просторі) виконаний раніше.

```
Thread = 1
[0] = 24657664 Nf = 88   22753280 Nf = 90 + 23070720 Nf = 90   22593280 Nf = 90   22968576 Nf = 92   21449472 Nf = 92   21255936 Nf = 92   19173120 Nf = 92   18697984 Nf = 92   18793216 Nf = 92
[1] = 18027008 Nf = 92   18005504 Nf = 92   16793856 Nf = 92   15598080 Nf = 94 + 15716352 Nf = 94   16421888 Nf = 94   16390912 Nf = 96   14831360 Nf = 96   13995776 Nf = 96   13843968 Nf = 96
[2] = 13491968 Nf = 96   12124416 Nf = 96 + 13575168 Nf = 96   11696128 Nf = 96 + 12137728 Nf = 96   12347648 Nf = 98   11511040 Nf = 98 + 12165120 Nf = 96   11317248 Nf = 96   11478784 Nf = 96
[3] = 11712000 Nf = 96   12338944 Nf = 96   11095296 Nf = 98   10488832 Nf = 98   10444288 Nf = 98   10187008 Nf = 98   10444288 Nf = 98   10750464 Nf = 98   10460416 Nf = 98   10411776 Nf = 98
[4] = 10547712 Nf = 98    9615360 Nf = 98    9725184 Nf = 96    9268736 Nf = 98   10171136 Nf = 98    9630720 Nf = 98    9520896 Nf = 98   10310144 Nf = 98    9314816 Nf = 98    8844288 Nf = 98
[5] =  8568832 Nf = 98    9028352 Nf = 98    8864000 Nf = 98    8786944 Nf = 98    8770816 Nf = 98    8971520 Nf = 98    8061184 Nf = 98    9039616 Nf = 98    9148672 Nf = 98    7842560 Nf = 100
Thread = 2
[0] = 24315904 Nf = 88 + 23898880 Nf = 88   23206400 Nf = 90   22245120 Nf = 90   22760704 Nf = 92   22665728 Nf = 92   20834560 Nf = 92   18776832 Nf = 94   17977856 Nf = 92 + 17735680 Nf = 92 +
[1] = 19432960 Nf = 92   17823744 Nf = 92   16498944 Nf = 94   18161408 Nf = 94   14856448 Nf = 96 + 14267648 Nf = 98 + 13939968 Nf = 96 + 13594624 Nf = 96 + 13083648 Nf = 96 + 12351744 Nf = 96 +
[2] = 12496384 Nf = 96   13166080 Nf = 96   13451264 Nf = 94   11650816 Nf = 98 + 12574720 Nf = 98   11817728 Nf = 96   12244736 Nf = 96   12432896 Nf = 98   11992320 Nf = 96   12736512 Nf = 96
[3] = 10489856 Nf = 98 + 10110720 Nf = 98 + 10356480 Nf = 96   10467840 Nf = 98   10352128 Nf = 98   11083008 Nf = 96    9971200 Nf = 98 +  9626368 Nf = 98 +  9175040 Nf = 98 +  9773824 Nf = 98
[4] =  9661184 Nf = 96    9307136 Nf = 98    9175552 Nf = 98    9824000 Nf = 98    9406208 Nf = 98    9231360 Nf = 98    8359936 Nf = 100 + 9008128 Nf = 98    8836608 Nf = 98    8626944 Nf = 98
[5] =  8660480 Nf = 98    8963328 Nf = 98    8995072 Nf = 98    7968000 Nf = 100 + 8084736 Nf = 100   8303616 Nf = 98    8035840 Nf = 98    7788032 Nf = 98    7102976 Nf = 98 +  8151552 Nf = 98
Thread = 3
[0] = 27599872 Nf = 86   25770240 Nf = 86   21907456 Nf = 92 + 24334336 Nf = 90   20675584 Nf = 92   19764992 Nf = 92 + 18215936 Nf = 92 + 18451456 Nf = 92   19691776 Nf = 90   17156352 Nf = 92 +
[1] = 16667392 Nf = 92 + 16410112 Nf = 94 + 16765952 Nf = 92   15900928 Nf = 94   15229184 Nf = 94   15523584 Nf = 94   14140672 Nf = 96   14247680 Nf = 96   14633984 Nf = 96   13194240 Nf = 96
[2] = 13859840 Nf = 96   12294912 Nf = 96   12169728 Nf = 98   11537408 Nf = 96 + 11919616 Nf = 96   11451392 Nf = 96 + 11375360 Nf = 96   11492352 Nf = 98   11019264 Nf = 98 + 11715840 Nf = 96
[3] = 11386880 Nf = 98   11131136 Nf = 96   10740992 Nf = 98   11609600 Nf = 96   11675904 Nf = 98   10878976 Nf = 96   10360832 Nf = 96   10788608 Nf = 98   10004736 Nf = 98    9824768 Nf = 98
[4] =  9257728 Nf = 98   10234112 Nf = 96    8558080 Nf = 98 +  8600576 Nf = 98    9052672 Nf = 98    8968192 Nf = 98    8140288 Nf = 100 + 8516096 Nf = 98    8473600 Nf = 98    8650752 Nf = 98
[5] =  8844544 Nf = 98    7531264 Nf = 100 + 7816704 Nf = 98    7820544 Nf = 98    8189952 Nf = 98    8210176 Nf = 100   7491840 Nf = 98    7312128 Nf = 98    8351232 Nf = 98    7273728 Nf = 98
Thread = 4
[0] = 24573952 Nf = 88 + 25537280 Nf = 88   24875264 Nf = 88   24933376 Nf = 88   23527168 Nf = 90   20508928 Nf = 92 + 19581184 Nf = 92 + 18914048 Nf = 92   18319616 Nf = 92   18215936 Nf = 92
[1] = 17441792 Nf = 92   17569280 Nf = 92   16595200 Nf = 92   16707584 Nf = 94   17074944 Nf = 94   14479872 Nf = 96   15001088 Nf = 96   13867008 Nf = 96   14000384 Nf = 96   12122624 Nf = 96 +
[2] = 13317376 Nf = 94   12077824 Nf = 98   12715520 Nf = 96   12618752 Nf = 98   11909888 Nf = 96   11212544 Nf = 98 + 12107776 Nf = 96   12372736 Nf = 98   11607296 Nf = 98   10898944 Nf = 98 +
[3] = 11386880 Nf = 98   11131136 Nf = 96   10740992 Nf = 98   11609600 Nf = 96   11675904 Nf = 98   10878976 Nf = 96   10360832 Nf = 96   10788608 Nf = 98   10004736 Nf = 98    9824768 Nf = 98
[4] =  9552896 Nf = 98    9323776 Nf = 98    9490688 Nf = 98    9609472 Nf = 98    9658880 Nf = 98    8787712 Nf = 98    8607232 Nf = 98    9565952 Nf = 98    9072128 Nf = 98    8256256 Nf = 98
[5] =  8723200 Nf = 98    9068544 Nf = 98    8913408 Nf = 98    7851520 Nf = 98    7603200 Nf = 98    8335872 Nf = 98    7756544 Nf = 96    7708416 Nf = 98    7272448 Nf = 98    7477760 Nf = 98
```

Рис. 1. Значення перших чотирьох циклів при формуванні функції вартості.
Знаком «+» позначено поточний вибір кращих значень функції вартості

Таким чином, можна побудувати так званий трек функції вартості, що буде відповідати поточному кращому значенню функції WHS. Наведемо зазначений трек для зовнішнього циклу, тобто найкраще значення WHS, яке буде у кінці кожного зовнішнього циклу: 17156352 ($N_f = 92$); 12122624 ($N_f = 96$); 10898944 ($N_f = 98$); 9175040 ($N_f = 98$); 8140288 ($N_f = 100$); 7531264 ($N_f = 100$).

Розглянемо трек функції вартості наприкінці кожного з внутрішніх циклів (табл. 1). У таблицю заносилися значення, якщо хоча б один раз у потоці було знайдено найкращу поточну функцію вартості (вона може не бути кращою серед інших потоків). Символом «–» відзначено зовнішній цикл, при якому жодного покращення не було знайдено. Пошук завершено на 33 ітерації зовнішнього циклу, коли знайдено S-блок з нелінійністю $N_f = 104$.

Таблиця 1

Приклад зміни значення найкращої функції вартості (cost) та нелінійності (N_f) для зовнішніх циклів та окремих потоків

Зовнішній цикл, номер	Thread = 1		Thread = 2		Thread = 3		Thread = 4	
	WHS	N_f	*WHS*	N_f	*WHS*	N_f	*WHS*	N_f
0.	3869696	100	4185088	102	3728896	100	3553536	100
1.	3536128	100	3527168	102	3413248	102	3496448	100
2.	3039744	102	–		3212032	102	3138304	102
3.	–		–		3015680	102	–	
4.	–		–		2941952	102	2929664	102
5.	–		–		–		2897664	102
6.	–		–		–		–	
7.	–		–		2832384	102	–	
8.	2803968	102	–		–		2814464	102
9.	2769920	102	–		–		2774784	102
10.	–		–		–		2695680	102
11.	2657280	102	–		–		–	
12.	–		–		–		–	
13.	2636800	102	2614784	102	2607104	102	–	
14.	–		–		–		–	
15.	–		2606080	102	–		–	
16.	2577408	102	–		–		2541824	102
17.	–		–		–		–	
18.	–		–		–		–	
19.	–		–		–		–	
20.	2408448	102	2485248	102	–		–	
21.	–		–		–		–	
22.	–		–		–		–	
23.	–		–		–		–	
24.	–		–		–		–	
25.	–		–		–		–	
26.	–		–		–		–	
27.	–		–		–		–	
28.	–		–		–		2395648	102
29.	–		–		2370304	102	–	
30.	–		–		–		–	
31.	–		–		–		–	
32.	–		–		–		–	
33.	2321408	104	2143488	102	2312704	102	–	

Завдяки випадковості зміни двох елементів S-блоку виконуються як зондування сусідніх станів на можливість покращення функції вартості у кожному з потоків. В разі можливості такого покращення воно виконується, і нова ітерація у кожному потоці приймає за поточний стан проведені зміни у S-блоку.

Із збільшенням номеру зовнішнього циклу функція вартості наближується до «дна» поточного локального мінімуму, що зменшує ймовірність покращити значення функції вартості при випадковій мутації. Тому є сенс обмежитися деякою максимальною кількістю поспіль зовнішніх циклів (max_frozen_outer_loops), при яких не виконано жодного покращення функції вартості, а також деякою максимальною кількістю зовнішніх циклів (max_outer_loops). При досягненні значення max_frozen_outer_loops або max_outer_loops вважається, що стан знаходиться у локальному мінімумі функції вартості. Тому пошук припиняється, формується новий стан S-блоку та процедура пошуку цільових параметрів S-блоку починається знову.

Для вибору оптимального значення max_outer_loops та max_frozen_outer_loops розглянемо їх вплив на результати роботи алгоритму пошуку.

5. Оптимізація параметрів локального пошуку

5.1. Оптимізація кількості зовнішніх циклів (max_outer_loops)

Для визначення максимальної кількості зовнішніх циклів (max_outer_loops) було проведено пошук S-блоків з цільовою нелінійністю $N_f = 104$ алгоритмом локального пошуку. Пошук виконувався за наступними параметрами:

- threads_count=30;
- max_outer_loops=50
- max_inner_loops=1000
- X=36;
- R=4.

Загалом було проведено 1500 запусків алгоритму локального пошуку, з яких було знайдено 346 (23 %) S-блоків з цільовою нелінійністю 104. Кількість зовнішніх циклів, протягом яких було знайдено покращення функції вартості, наведена на рис. 2 (гістограми суцільного кольору), для порівняння окремо наведено стовпчики (зі штрихованим заповненням) лише для випадку, коли у результаті роботи алгоритму було досягнуто цільову нелінійність. З 1500 лише в двох випадках було виконано 25 ітерацій зовнішнього циклу. Таким чином, при обраних параметрах, доцільно обмежитися максимальним значенням кількості зовнішніх циклів max_outer_loops=25.

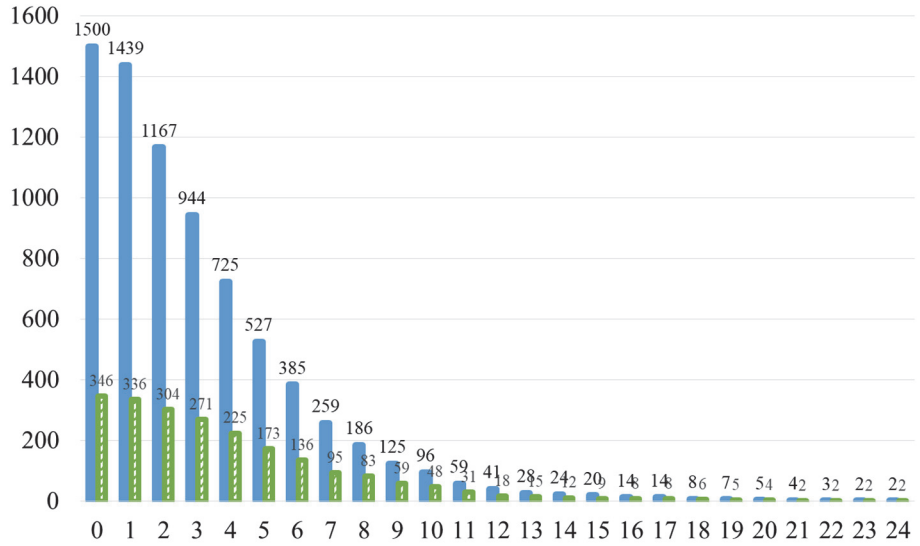

Рис. 2. Кількість зовнішніх циклів, протягом яких було знайдено покращення функції вартості

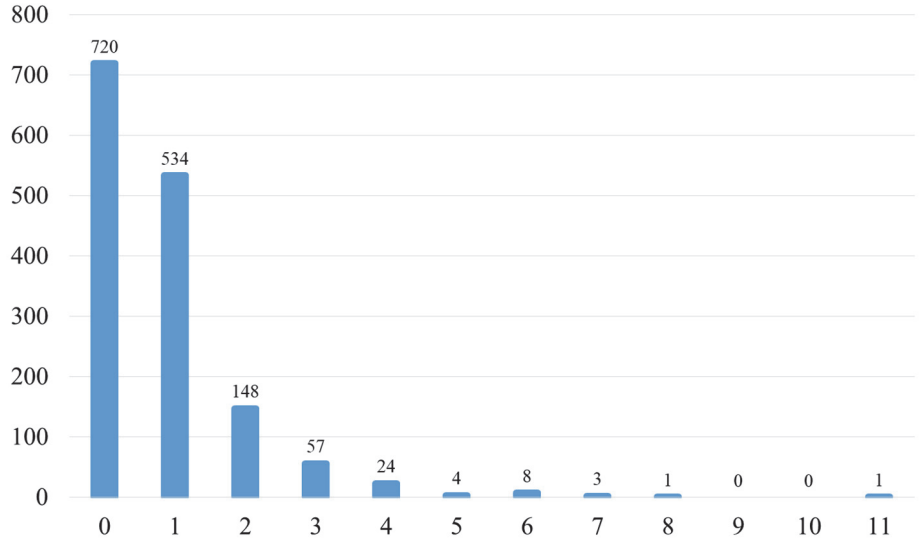

Рис. 3. Розподіл кількості поспіль виконаних зовнішніх циклів,
при яких не знайдено жодного покращення, однак потім покращення мали місце

На рис. 3 наведено гістограму розподілу кількості поспіль зовнішніх циклів, при яких не знайдено жодного покращення, однак потім покращення мали місце (для наочного пояснення цього випадку можна звернутися до комірок позначених символом «–» табл. 1). Бачимо, що після дев'ятої ітерації зовнішнього циклу менш ніж 9 % зовнішніх циклів знаходять покращення функції вартості. Нуль відповідає випадку, якщо у кожному зовнішньому циклі були знайдені рішення, які покращують цільову функцію. Якщо обмежитися пятьма поспіль відсутніми покращеннями, то з 1500 проведених випробувань будуть помилково відкинуті 15 випробувань (4 – випробування з 5 поспіль нерезультативними випробуваннями, 8 – з 6 поспіль нерезультативними випробуваннями, 3 – з 7 та по одному з 9 та 11), що становить 1 %.

Таким чином, при заданих параметрах (threads_count=30, max_inner_loops=1000, X=36, R=4) будимо вважати за кращі:

- max_outer_loops=25;
- max_frozen_outer_loops=5.

Ще раз зазначимо, при інших параметрах (наприклад, зменшення threads_count та / або max_inner_loops) необхідно вибирати інші кращі значення max_outer_loops та max_frozen_outer_loops (в наведеному прикладі необхідно буде їх збільшити, так як буде зменшена кількість сусідніх значень, які тестуються). При max_frozen_outer_loops=5 та max_inner_loops=1 000 маємо загальне обмеження у 5000 тестів, при яких відсутнє жодне покращення значення цільової функції.

Узагальнена картина зміни треку функції вартості з кожним новим кроком ітерації зовнішнього циклу наведена на рис. 4. Суцільною лінією позначено усереднене значення WHS за 1500 випробуваннями. Пунктир – усереднене значення WHS за 346 випробуваннями, за яким знайдено S-блоків з $N_f = 104$. Як бачимо, середнє значення функцій вартості, які приходять до цільових показників нелінійності має характерний для інших трек. Крапкова лінія – середнє відхилення від усередненого значення функції вартості за всіма1500 випробуваннями. Середнє відхилення має невелике значення, що вказує на добру узгодженість різних випробувань з середньо статистичними результатами. Лінія крапка-тире – абсолютне максимальне відхилення від середнього значення. Значення функції вартості, у випадках, де було знайдено S-блок з цільовою нелінійністю, не перевищувало $2,6 \cdot 10^6$. За допомогою алгоритму локального пошуку можна досить швидко знайти S-блок з функцією вартості нижчі ніж $2 \cdot 10^6$, що добре буде поєднуватися з іншими методами пошуку S-блоків з цільовими характеристиками.

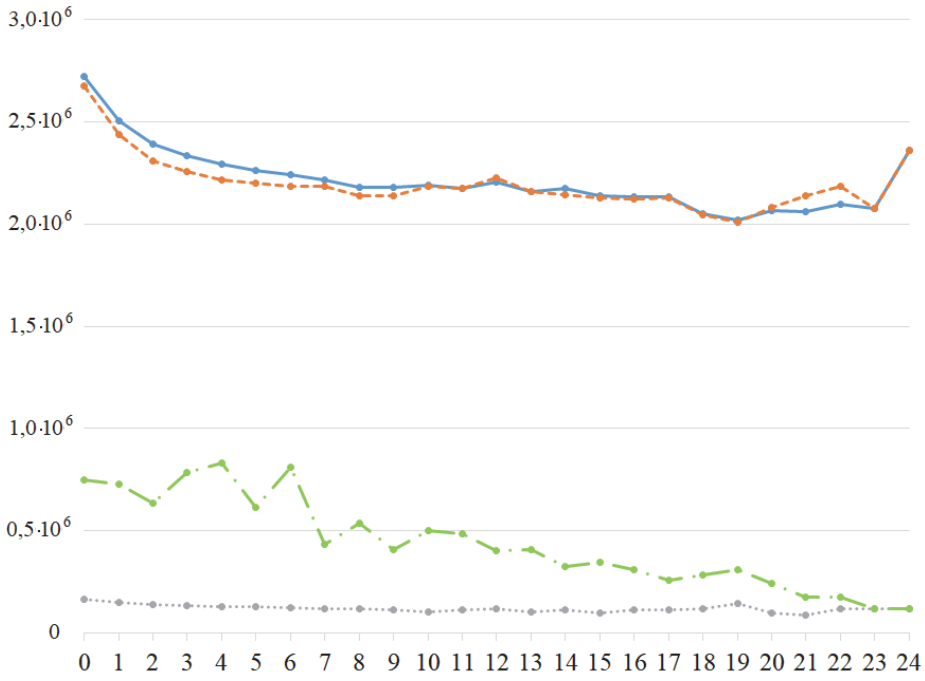

Рис. 4. Трек зміни середнього значення найкращої функції вартості

5.2. Оптимізація кількості внутрішніх циклів (max_inner_loops)

Наступним кроком встановимо вплив кількості внутрішніх циклів (параметр max_inner_loops). Тестування проводилися на двох обчислювальних машинах з багатоядерними процесорами:

- з тактовою частотою процесору 2,59 GHz, Intel Core i9-7980 XE, RAM 64 GB, Window 10 (на ПК № 1);
- з тактовою частотою процесору 3,49 GHz, AMD Rizen 9 3950 X 16, RAM 128 GB, Windows 10 (на ПК № 2).

На кожній машині запускалось по 30 окремих потоків, тобто threads_count = 30, кількість зовнішніх циклів була max_outer_loops=50, параметри цільової функції *WHS* залишалися фіксованими: $X = 36$; $R = 4$. Параметр max_inner_loops змінювався у наступному діапазоні:

- max_inner_loops від 200 до 600 з шагом 10 (на машині № 1);
- max_inner_loops від 610 до 1000 з шагом 10 (на машині № 2).

Тестування проводилось по 11 груп, у кожній групі виконувалось 100 запусків алгоритму пошуку. У кожній групі іспитів всі параметри залишалися незмінними. Таким чином, для кожного параметра проводилось 1 100 запусків. Всього було проведено 89 100 запусків алгоритму за приблизно 367 годин (астрономічного часу) на кожній машині. На рис. 5 наведено кількість знайдених цільових S-блоків в залежності від кількості внутрішніх циклів для кожної з 11 груп. Як бачимо з отриманих результатів при max_inner_loops менш 250 цільові S-блоки майже не знайдено. В інтервалі від 250 до 650 йде майже лінійних ріст кількості знайдених цільових S-блоків, а з 650 та вище значного приросту кількості знайдених цільових S-блоків не спостерігається.

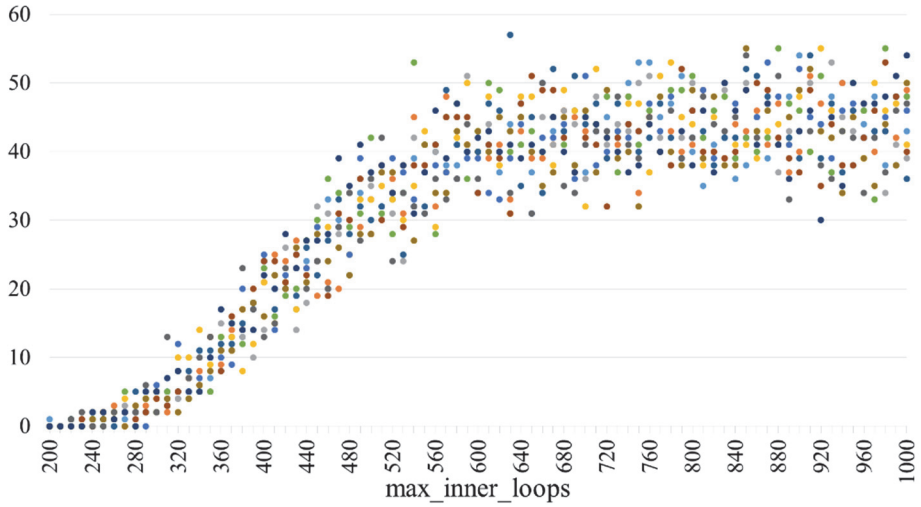

Рис. 5. Кількість знайдених цільових S-блоків в залежності від кількості внутрішніх циклів

Якщо треба знайти максимальну кількість цільових S-блоків, то слід брати велике значення max_inner_loops у кожному тестуванні. Однак, якщо збільшується кількість внутрішніх циклів, то пропорційно зростає час на кожне тестування. На рис. 6 наведено середній (за 100 запусків) час, що було затрачено на кожне виконання алгоритму локального пошуку у кожній групі. Нагадаємо, що тестування проведено на двох різних обчислювальних машинах, тому результати затраченого часу, які наведені на першій половині графіку, кількісно не співпадають із затраченим часом наведеним на другій половині графіку. Це пояснюється різною обчислювальною потужністю цих ПК. Але якісно відповідні залежності є продовженням одна одної.

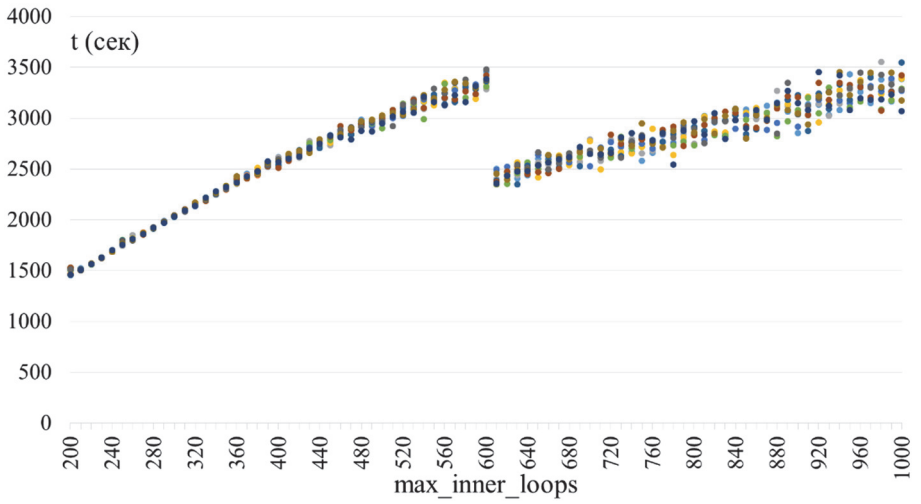

Рис. 6. Час, що було затрачено для тестування у кожній групі

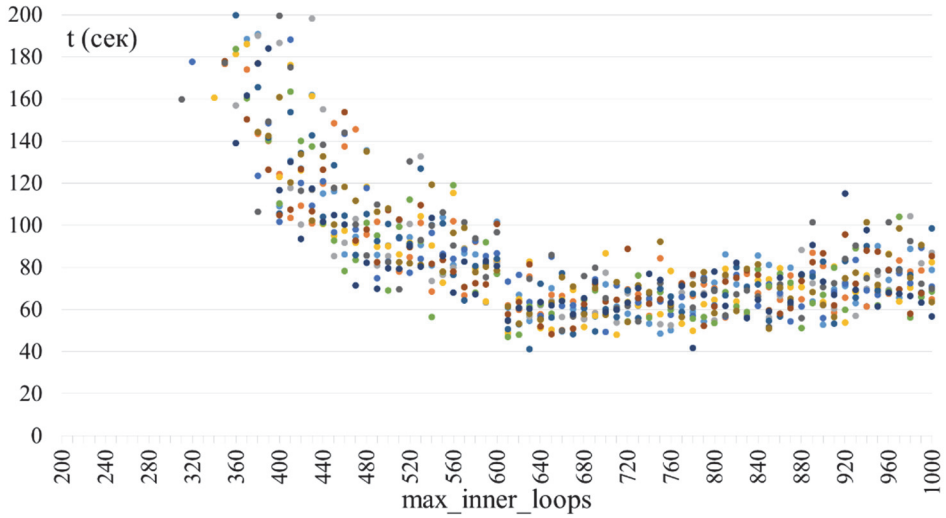

Рис. 7. Середній час пошуку цільового S-блоку у кожній групі

Доцільно розглянути середній час, що було затрачено у кожній групі на пошук цільового S-блоку. Зазначений час будемо обраховувати як відношення часу виконання алгоритму (див. рис. 6) до кількості знайдених цільових S-блоків у циклі (див. рис. 5). Обчислений таким чином середній час пошуку цільового S-блоку наведено на рис. 7. Виходячи з мінімізації значення середнього часу пошуку цільового S-блоку обираємо значення max_inner_loops = 650.

Слід відзначити, що загальна кількість ітерації при пошуку цільового S-блоку алгоритмом пошуку локального мінімуму складає: threads_count * max_outer_loops * max_inner_loops = 30 * 50 * 650 = 975 000. Від вказаного значення треба відштовхуватися під час вибору вхідних параметрах на фізичних пристроях, які підтримують іншу кількість потоків (threads_count).

5.3. Оптимальні параметри алгоритму локального пошуку цільового S-блоку

Узагальнюючи наведений матеріал встановлені наступні оптимальні (с точки зору мінімального часу) параметри для проведення пошуку S-блоку методом локального пошуку з цільовою нелінійністю $N_f = 104$ та кількістю паралельних потоків threads_count = 30:

- максимальна кількість зовнішніх циклів: max_outer_loops=25;
- максимальна кількість внутрішніх циклів: max_inner_loops=650;
- максимальна кількість поспіль зовнішніх циклів, при яких не виконано жодного покращення функції вартості: max_frozen_outer_loops=5;

Параметри наведено за умови використання цільової функції (1) з параметрами:

- $X = 36$;
- $R = 4$;

Наведені параметри справедливі для 30 потоків, для іншої кількості потоків оптимальні параметри можуть бути іншими. Це пов’язано із загальною кількістю випробувань, які проводяться для кожного циклу пошуку цільового S-блоку.

6. Середній час пошуку цільового S-блоку при оптимальних параметрах

При вказаних оптимальних параметрах було проведено 78 869 запусків алгоритму локального пошуку. Часом пошуку вважався загальний час роботи програми до знаходження цільового S-блоку. Тобто, якщо за два запуски по 30 секунд кожний не було знайдено цільового S-блоку, а впродовж третього знайдено за 10 секунд, то часом пошуку вважається 70 секунд.

Кожний запуск закінчувався або при знаходженні цільового S-блоку або виходом з циклу при досягненні граничних значень ітерації у пошуку. Загалом було знайдено 16 980

(21,5 % від загальної кількості циклів тестувань) цільових S-блоків із середнім часом пошуку одного блоку у 33,2 секунди. Гістограма розподілу кількості знайдених цільових S-блоків в залежності від часу, що був затрачений на їх пошук, наведено на рис. 8. На рисунку наведено значення, що не перевищують двох хвилин пошуку. Ще 390 цільових S-блоків (2,3 % від загальної кількості) було знайдено за час, що перевищував дві хвилини.

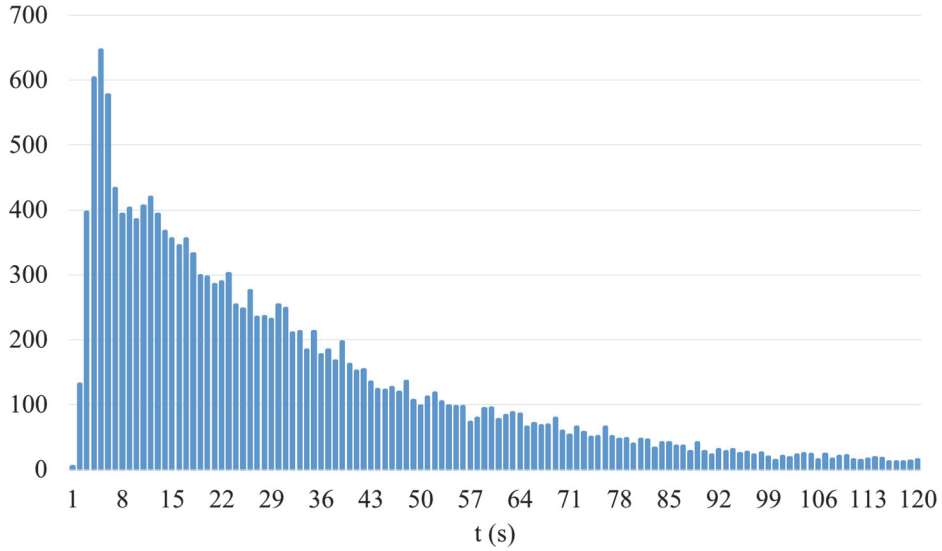

Рис. 8. Розподіл кількості знайдених цільових S-блоку в залежності від часу (t, секунди), що було затрачено на їх пошук

Висновки

Досліджено алгоритм локального пошуку з точки зору застосування у формуванні S-блоків з заданими криптографічними властивостями, серед яких було обрано нелінійність $N_f = 104$. Наведено опис алгоритму. Досліджено основні його параметри та встановлено оптимальні (з точки зору часу формування S-блоку) їх значення:

- максимальна кількість зовнішніх циклів: max_outer_loops=25;
- максимальна кількість внутрішніх циклів: max_inner_loops=650;
- максимальна кількість поспіль зовнішніх циклів, при яких не виконано жодного покращення функції вартості: max_frozen_outer_loops=5;

Вказані значення є оптимальними за умови запуску алгоритму з 30 паралельно працюючими потоками та застосування цільової функції (1) з параметрами: $X = 36$ та $R = 4$.

При вказаних параметрах середній час формування S-блоку з нелінійністю $N_f = 104$ становить 33,2 секунди та ймовірність знаходження S-блоку становить 21,5 %.

References:

1. Schneier B. Applied cryptography: protocols, algorithms, and source code in C. New York : Wiley, 1996.
2. Menezes A.J., Oorschot P.C. van, Vanstone S.A., Oorschot P.C. van, Vanstone S.A. Handbook of Applied Cryptography // CRC Press (2018). https://doi.org/10.1201/9780429466335.
3. Carlet C. Vectorial Boolean functions for cryptography // Boolean Models and Methods in Mathematics, Computer Science, and Engineering (2006).
4. Carlet C., Ding C. Nonlinearities of S-boxes // Finite Fields and Their Applications. 13, 121–135 (2007). https://doi.org/10.1016/j.ffa.2005.07.003.
5. Álvarez-Cubero J. Vector Boolean Functions: applications in symmetric cryptography, (2015). https://doi.org/10.13140/RG.2.2.12540.23685.
6. Burnett L.D. Heuristic Optimization of Boolean Functions and Substitution Boxes for Cryptography, https://eprints.qut.edu.au/16023/, (2005).
7. Clark A.J. Optimisation heuristics for cryptology, https://eprints.qut.edu.au/15777/ (1998).
8. Fuller J.E. Analysis of affine equivalent boolean functions for cryptography, https://eprints.qut.edu.au/15828/ (2003).

9. McLaughlin J. Applications of search techniques to cryptanalysis and the construction of cipher components, http://etheses.whiterose.ac.uk/3674/ (2012).

10. Battiti R., Brunato M., Mascia F. Reactive Search and Intelligent Optimization : Springer US (2009). https://doi.org/10.1007/978-0-387-09624-7.

11. Hromkovič J. Algorithmics for Hard Problems // Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. Springer-Verlag, Berlin Heidelberg (2004). https://doi.org/10.1007/978-3-662-05269-3.

12. Arya V., Garg N., Khandekar R., Meyerson A., Munagala K., Pandit V. Local search heuristic for k-median and facility location problems // Proceedings of the thirty-third annual ACM symposium on Theory of computing. pp. 21–29. Association for Computing Machinery, New York, NY, USA (2001). https://doi.org/10.1145/380752.380755.

13. Edelkamp S., Schroedl S. Heuristic Search: Theory and Applications. Morgan Kaufmann, Amsterdam; Boston (2011).

14. Millan W., Burnett L., Carter G., Clark A., Dawson E. Evolutionary Heuristics for Finding Cryptographically Strong S-Boxes // Varadharajan, V. and Mu, Y. (eds.) Information and Communication Security. pp. 263–274. Springer, Berlin, Heidelberg (1999). https://doi.org/10.1007/978-3-540-47942-0_22.

15. Freyre-Echevarría A., Alanezi A., Martínez-Díaz I., Ahmad M., Abd El-Latif A.A., Kolivand H., Razaq A. An External Parameter Independent Novel Cost Function for Evolving Bijective Substitution-Boxes // Symmetry. 12, 1896 (2020). https://doi.org/10.3390/sym12111896.

16. Clark J.A., Jacob J.L., Stepney S. The design of S-boxes by simulated annealing // New Gener Comput. 23, 219–231 (2005). https://doi.org/10.1007/BF03037656.

17. Clark J.A., Jacob J.L., Stepney S. Searching for cost functions // Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753). pp. 1517-1524 Vol.2 (2004). https://doi.org/10.1109/CEC.2004.1331076.

18. Millan W., Clark A. Smart Hill Climbing Finds Better Boolean Functions. (1997).

19. Millan W., Clark A., Dawson E. Heuristic design of cryptographically strong balanced Boolean functions // Nyberg, K. (ed.) Advances in Cryptology – EUROCRYPT'98. pp. 489–499. Springer Berlin Heidelberg, Berlin, Heidelberg (1998).

20. Millan W., Clark A., Dawson E. Boolean Function Design Using Hill Climbing Methods // Pieprzyk, J., Safavi-Naini, R., and Seberry, J. (eds.) Information Security and Privacy. pp. 1–11. Springer, Berlin, Heidelberg (1999). https://doi.org/10.1007/3-540-48970-3_1.

21. Millan W. How to improve the nonlinearity of bijective S-boxes // Boyd C. and Dawson E. (eds.) Information Security and Privacy. pp. 181–192. Springer, Berlin, Heidelberg (1998). https://doi.org/10.1007/BFb0053732.

22. Clark J.A., Jacob J.L., Stepney S. The design of s-boxes by simulated annealing // Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753). pp. 1533-1537 Vol.2 (2004). https://doi.org/10.1109/CEC.2004.1331078.

23. Kavut S., Yücel M.D. Improved Cost Function in the Design of Boolean Functions Satisfying Multiple Criteria // Johansson T. and Maitra S. (eds.) Progress in Cryptology – INDOCRYPT 2003. pp. 121–134. Springer, Berlin, Heidelberg (2003). https://doi.org/10.1007/978-3-540-24582-7_9.

24. Souravlias D., Parsopoulos K.E., Meletiou G.C. Designing bijective S-boxes using Algorithm Portfolios with limited time budgets // Applied Soft Computing. 59, 475–486 (2017). https://doi.org/10.1016/j.asoc.2017.05.052.

25. Ivanov G., Nikolov N., Nikova S. Cryptographically Strong S-Boxes Generated by Modified Immune Algorithm // Pasalic E. and Knudsen L.R. (eds.) Cryptography and Information Security in the Balkans. pp. 31–42. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-29172-7_3.

26. Eastlake 3rd, D., Schiller J., Crocker S. Randomness Requirements for Security (2005).

27. Tesar P. A New Method for Generating High Non-linearity S-Boxes (2010).

28. Laskari E.C., Meletiou G.C., Vrahatis M.N. Utilizing Evolutionary Computation Methods for the Design of S-Boxes // 2006 International Conference on Computational Intelligence and Security. pp. 1299–1302 (2006). https://doi.org/10.1109/ICCIAS.2006.295267.

29. Kapuścińsk T., Nowicki R.K., Napoli C. Application of Genetic Algorithms in the Construction of Invertible Substitution Boxes // Rutkowski L., Korytkowski M., Scherer R., Tadeusiewicz R., Zadeh L.A., and Zurada J.M. (eds.). Artificial Intelligence and Soft Computing. pp. 380–391. Springer International Publishing, Cham. (2016). https://doi.org/10.1007/978-3-319-39378-0_33.

30. Ivanov G., Nikolov N., Nikova S. Reversed genetic algorithms for generation of bijective s-boxes with good cryptographic properties // Cryptogr. Commun. 8, 247–276 (2016). https://doi.org/10.1007/s12095-015-0170-5.

31. Picek S., Cupic M., Rotim L. A New Cost Function for Evolution of S-Boxes // Evolutionary Computation. 24, 695–718 (2016). https://doi.org/10.1162/EVCO_a_00191.

32. Freyre Echevarría A., Martínez Díaz I. A new cost function to improve nonlinearity of bijective S-boxes. (2020).

Надійшла до редколегії 10.09.2021

Відомості про авторів:

Кузнецов Олександр Олександрович – д-р техн. наук, професор, Харківський національний університет імені В.Н. Каразіна, професор кафедри безпеки інформаційних систем і технологій, факультет комп'ютерних наук; Україна; e-mail: kuznetsov@karazin.ua, ORCID: https://orcid.org/0000-0003-2331-6326

Полуяненко Микола Олександрович – канд. техн. наук, Харківський національний університет імені В.Н. Каразіна, доцент кафедри безпеки інформаційних систем і технологій, факультет комп'ютерних наук; Україна; e-mail: nlfsr01@gmail.com, ORCID: https://orcid.org/0000-0001-9386-2547

Бердник Сергій Леонідович – канд. техн. наук, доцент, Харківський національний університет імені В.Н. Каразіна, в.о. завідувача кафедри фізичної і біомедичної електроніки та комплексних інформаційних технологій, факультет радіофізики, біомедичної електроніки та комп`ютерних систем; Україна; e-mail: berdnik@karazin.ua, ORCID: https://orcid.org/0000-0002-0037-6935

Кандій Сергій Олегович – технік-конструктор, АТ «Інститут інформаційних технологій», Україна; e-mail: sergeykandy@gmail.com, ORCID: https://orcid.org/0000-0003-0552-8341

Заїченко Юлія Олександрівна – магістрант, Харківський національний університет імені В.Н. Каразіна, кафедра безпеки інформаційних систем і технологій, факультет комп'ютерних наук; Україна; e-mail: yuliya.zaicenko.00@gmail.com, ORCID: https://orcid.org/0000-0001-6116-2693