ISSN 2707-1898 (print)

Український журнал інформаційних технологій
Ukrainian Journal of Information Technology
http://science.lpnu.ua/uk/ujit
https://doi.org/10.23939/ujit2021.02.058
Article received 10.09.2021 p.
Article accepted 23.11.2021 p.
UDK 004.056:061.6

Correspondence author
Ya. R. Sovyn
yaroslav.r.sovyn@lpnu.ua

Я. Р. Совин, В. В. Хома

Національний університет "Львівська політехніка", м. Львів, Україна

ЕВРИСТИЧНИЙ МЕТОД ДЛЯ BITSLICED ПОДАННЯ ВИПАДКОВО ЗГЕНЕРОВАНИХ 8×8 КРИПТОГРАФІЧНИХ S-BOX

Розглянуто питання щодо підвищення безпеки та ефективності програмної реалізації симетричних блокових шифрів. Використано bitslice-підхід до безпечної імплементації криптоалгоритмів, який має такі потенційні переваги, як високу швидкодію і невимогливість до обчислювальних ресурсів. Проте, відомі bitsliced-методи мають обмеження, оскільки працюють з детермінованими S-Box або розраховують S-Box менших розмірів. Запропоновано новий евристичний метод bitsliced-подання криптографічних 8×8 S-Box, що містять випадково згенеровані значення. Метод засновано на декомпозиції таблиці істинності, яка описує S-Box, на дві частини. Одна частина таблиці формує логічні маски, а інша – розбивається на бітові вектори, для знаходження логічного опису яких застосовано вичерпний пошук. Після знаходження опису всіх векторів ці дві частини таблиці об'єднуються в одну за допомогою логічних операцій. Використання запропонованого методу, орієнтованого на програмну реалізацію в логічному базисі {AND, OR, XOR, NOT}, забезпечує мінімізацію довільних 8×8 S-Box. Цей метод допускає імплементацію з використанням стандартних логічних інструкцій на будь-яких 8/16/32/64-бітних процесорах. Також можливе використання логічних SIMD-інструкцій із розширень SSE, AVX, AVX-512 для x86-64 процесорів, що забезпечує високу швидкодію завдяки використанню довгих регістрів. Розроблено відповідне програмне забезпечення, яке реалізує метод пошуку bitsliced-подання заданого S-Box, а також автоматично формує для нього C++ код на базі SSE, AVX і AVX-512 інструкцій. Досліджено ефективність методу на S-Box відомих блокових шифрів, зокрема Національного стандарту шифрування "Kalyna". Встановлено, що розроблений алгоритм потребує майже вдвічі менше вентилів для bitsliced-опису довільного S-Box, ніж кращий відомий алгоритм (370 вентилів проти 680 відповідно). Для шифрів, у яких використовуються дві або чотири таблиці S-Box, внаслідок спільної мінімізації можна отримати до 330 або 300 вентилів на таблицю відповідно.

***Ключові слова**:* bitslicing; S-Box; логічна мінімізація; SIMD; x86-64 CPU; програмна імплементація; блокові шифри.

Вступ

Швидкодія блокових симетричних шифрів (БСШ) є їх важливою характеристикою, яка у багатьох випадках визначає швидкодію аплікації. З огляду на це, сучасний БСШ повинен забезпечувати достатньо високу продуктивність програмної реалізації шифрування для широкого класу мікропроцесорних архітектур із різними обчислювальними можливостями й доступними ресурсами. Не менш важливою для програмної реалізації БСШ є підвищена стійкість до атак через сторонні канали: для low-end CPU (8/16/32-бітні мікроконтролери) –це насамперед атаки аналізу енергоспоживання, для high-end CPU (x86, ARM Cortex-A) – це передусім часові та кеш атаки.

Є декілька підходів до програмної реалізації БСШ, що відрізняються швидкодією, безпекою та вимогами до ресурсів: класичний, табличний, на базі SIMD-інструкцій та bitsliced. З них потенційно найвищу швидкодію має bitsliced-підхід. Окрім цього, він забезпечує constant-time імплементацію блокових шифрів з імунітетом до часових та кеш атак [2], [9], є невимогливим до ресурсів, максимально використовує можливості high-end мікропроцесорів щодо збільшення швидкодії внаслідок розпаралелювання як виконання коду (суперскалярність), так і оброблення даних (SIMD-технологія), а також допускає адаптацію для low-end CPU і апаратну реалізацію на FPGA і ASIC.

Базова ідея Bitslicing – це описати шифр у термінах логічних операцій, наприклад: AND, XOR, OR, NOT. У процесорах кожну таку логічну операцію можна представити відповідною інструкцією (зокрема і векторною), яка може одночасно обробляти багато біт. Висока швидкодія досягається завдяки тому, що CPU обробляє багато елементів шифру (байтів, блоків) паралельно, використовуючи швидкі логічні інструкції, та завдяки простішому виконанню деяких операцій, наприклад, бітових перестановок. Чим більша розрядність bitsliced-регістрів, тим ймовірніший виграш у швидкодії, тому особливо ефективний цей підхід для векторних інструкцій, які оперують багатобітними регістрами.

Основною проблемою bitsliced імплементації БСШ є подання S-Box розмірами ≥ 6×6, особливо у випадку, якщо вони згенеровані випадково і не описуються математичними перетвореннями. Оскільки чим довша довжина регістрів, тим програмний bitslicing дає більший виграш. Ми акцентуємо увагу на представленні довільних 8×8 S-Box у випадку використання векторних 128/256/512-бітних SIMD-розширень системи команд x86-64 процесорів.

Об'єкт дослідження – випадково згенеровані криптографічні 8×8 S-Box.

Предмет дослідження – спосіб опису S-Box мінімальною кількістю логічних вентилів.

Мета роботи – розробити метод для bitsliced подання довільних 8×8 S-Box мінімальною кількістю логічних інструкцій, за умови використання стандартних логічних інструкцій, що входять у ISA та SIMD-розширення x86-64 CPU.

Для досягнення поставленої мети визначено такі *основні завдання дослідження*:

- вибрати формат логічного подання 8×8 S-Box;
- розробити метод пошуку та відбору логічних представлень векторів із мінімальною кількістю AND/OR/XOR/NOT вентилів;
- розробити утиліту для автоматичного формування програмного коду на базі SIMD-інструкцій для знайденого bitsliced опису S-Box;
- оцінити ефективність алгоритму на 8×8 S-Box різних БСШ.

Наукова новизна отриманих результатів дослідження – розроблено метод автоматичного пошуку bitsliced-подання для довільних 8×8 S-Box зі значно зменшеним числом стандартних логічних операцій, що дає змогу збільшити швидкодію програмних реалізацій багатьох блокових шифрів.

Практична значущість результатів дослідження – запропонований підхід до bitsliced-опису довільних S-Box усуває обмеження відомих методів такого подання, що стримували використання bitcliced-підходу під час удосконалення програмних реалізацій блокових шифрів для багатьох процесорних архітектур.

Матеріали та методи дослідження. Bitsliced імплементації використаних у роботі S-Box та утиліти для їхнього формування доступні за посиланням [14].

Аналіз останніх досліджень та публікацій. Найважчий етап у bitsliced імплементації БСШ, що значною мірою визначає швидкодію загалом, є логічне подання таблиць нелінійної заміни S-Box. У випадку апаратної реалізації як логічний базис виступають логічні вентилі (Gate Equivalent, GE) {AND, OR, XOR, NOT}, у програмній bitsliced імплементації вентилі заміняються відповідними інструкціями, які присутні в більшості процесорних архітектур. Тому надалі в роботі поняття вентиль та інструкція будемо вживати як синоніми. У деяких процесорах немає інструкції NOT, яка емулюється інструкцією XOR, з огляду на те, що $\bar{x} = x \oplus 1$. Оскільки процесорні логічні інструкції здебільшого обробляють два операнди, то й логічні вентилі повинні бути двовходові (рис. 1), щоби можна було однозначно перейти від логічного подання до програмного.

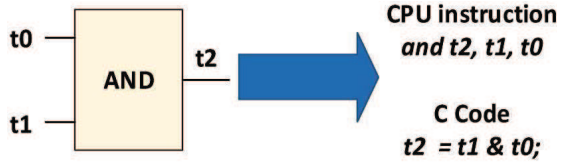

Рис. 1. Перехід від логічного до програмного bitsliced подання

Для мінімізації логічного опису S-Box застосовують різні критерії, зокрема в контексті цієї роботи важливі такі критерії:

a) Bitslice gate complexity (BGC) – мінімальна кількість вентилів {AND, OR, XOR, NOT}. Використовується у програмних bitsliced імплементаціях для різних процесорних архітектур;

b) Gate complexity (GC) – мінімальна кількість вентилів із додатковим використанням NAND, NOR та XNOR, що потрібно для ефективної апаратної реалізації або програмної на процесорах із підтримкою відповідних інструкцій.

Проаналізуємо підходи до подання S-Box у вигляді комбінаційної логічної схеми з мінімальною кількістю двовходових вентилів AND, OR, XOR, NOT (критерій BGC).

Є три варіанти побудови 8×8 S-Box, що визначають і підходи до їхньої оптимізації:

1. S-Box із закладеною алгебраїчною структурою, як, наприклад, у шифрах AES чи SM4. Такого типу S-Box допускають компактну bitsliced імплементацію внаслідок використання властивостей закладених перетворень. Водночас наявність аналітичного опису полегшує криптоаналіз.
2. Використання S-Box менших розмірів (переважно 4×8), з яких генеруються 8×8 S-Box. Ці S-Box теж можуть мати компактну bitsliced імплементацію через відому структуру. Прикладом є S-Box шифрів CLEFIA, Crypton, ICEBERG та багато інших.
3. Неалгебраїчні S-Box згенеровані випадково, що мають задані криптографічні властивості. Відсутність будь-яких аналітичних залежностей, які б описували дані S-Box, ускладнює їхнє bitsliced подання, але одночасно й робить більш стійкими до диференційного, лінійного та алгебраїчного криптоаналізу. Прикладом такого підходу є вісім S-Box шифру "Kalyna", S-Box шифрів "Kuznyechik", KHAZAD і Anubis.

Багато робіт [5], [7], [10], [13] присвячено логічній мінімізації S-Box алгоритму AES. Основний підхід полягає в розбитті таблиці на лінійну й нелінійну частини і їх мінімізації з використанням евристичних методів. З погляду програмної імплементації найкращий результат у 128 вентилів має робота [5].

У роботі [3] здійснено реверс-інжиніринг прихованої структури SBox шифру "Kuznyechik", хоча, за твердженням розробників шифру, цей S-Box був згенерований випадково, що дало змогу описати його аналітичними виразами. Базуючись на декомпозиції з праці [3], у роботі [1] синтезовано bitsliced подання S-Box шифру "Kuznyechik", що потребує 226 вентилів.

У роботах [1], [3], [7], [5], [10], [13] експлуатовано алгебраїчну структуру, закладену в S-Box таблицю, тому не можуть бути використані для мінімізації випадково згенерованих S-Box.

Для мінімізації невеликих S-Box (переважно 4×4), які характерні для легковагових шифрів чи використовуються для генерації 8×8 S-Box, застосовують програми SAT-Solvers [15]. Проблемою є те, що такий підхід працює тільки для невеликих S-Box, розміром до 5×5, хоча вже тут виникають складнощі й не завжди вдається знайти рішення. Для 8×8 S-Box цей підхід неможливо реалізувати з погляду обчислювальної складності. Можна розбивати 8×8 таблицю на менші частини та мінімізувати за допомогою SAT-Solvers кожну частину окремо, проте результат буде не оптимальним, бо не враховуватиме залежності між ними.

Як приклад мінімізації неалгебраїчних S-Box можна зазначити роботу [12], у якій описано прямолінійну табличну реалізацію (без використання алгебраїчної структури AES S-Box), синтезовану в середовищі Synopsys Design Compiler, що потребує 1312 вентилів.

У патенті [4] описано алгоритм логічної мінімізації довільних S-Box за критерієм BGC, який формує bitsliced подання 8×8 S-Box у середньому із 650-680 вентилів, зокрема для опису S-Box шифру "Kuznyechik" потрібні 681 вентиль. Алгоритм для заданої таблиці істин-

ності мінімізує кількість булевих операцій, що входять у сформований для неї поліном Жегалкіна.

S-Box 8×8 можна розглядати як логічні функції задані таблицями істинності. Класичні методи мінімізації логічних функцій, поданих таблицею істинності, такі як метод карт Карно чи метод простих імплікант Куайна-Мак-Класкі, не підходять у цьому випадку, оскільки здійснюють дворівневу мінімізацію, використовуючи тільки операції AND, OR, NOT (без XOR) та безпосередньо не враховують вимогу двовходовості вентилів.

Світовим стандартом де-факто для мінімізації функцій із великою кількістю змінних є програма Espresso, що використовує евристичний алгоритм [6]. Espresso видає результати, що на практиці близькі до глобального мінімуму, але потребує значно менше пам'яті та часу, порівнюючи з іншими методами. Алгоритм Espresso використовують у багатьох засобах і CAD-пакетах синтезу логічних схем (FPGA, ASIC) на етапі мінімізації.

Проте застосування Espresso до неалгебраїчних S-Box теж не є найкращим рішенням, бо зберігається проблема з використанням тільки операцій AND, OR, NOT (операція XOR може бути впроваджена вже над результатом мінімізації, але не в процесі) та не враховується обмеження на кількість входів вентилів. Це, зокрема, підтверджують результати мінімізації з допомогою програми Logic Friday (надає графічний інтерфейс до алгоритму Espresso) S-Box шифру "Kuznyechik", який потребує 1604 вентилів та AES-подібного S-Box, що описується 1582 вентилями [4].

Отже, жодний із розглянутих методів мінімізації або не дає задовільних результатів або не може бути безпосередньо застосований до неалгебраїчних 8×8 S-Box. Тому ми розробили та імплементували мовою Python власний евристичний метод мінімізації таких S-Box, що використовує тільки операції {AND, OR, XOR, NOT} і тому може бути реалізований як програмно на будь-яких процесорах, що мають необхідні інструкції для виконання зазначених операцій (зокрема SIMD), так і апаратно відповідними логічними вентилями.

Утиліта евристичної мінімізації здійснює автоматичну генерацію C++ функцій на базі SSE (128 біт), AVX (256 біт) та AVX-512 (512 біт) x86-64 SIMD-інструкцій, що реалізують bitsliced обчислення заданого S-Box. Щоби представити SIMD-інструкції в C++ коді використовуються Intrinsic-функції [4], які є високорівневою обгорткою над асемблерними командами.

Результати дослідження та їх обговорення

Подання S-Box. У bitsliced наведені таблиці заміни можна розглядати як логічні функції 8×8 задані таблицями істинності. Наприклад, фрагмент *Sbox*0 із шифру "Kalyna" має вигляд, показаний у табл. 1, де x_0-x_7 – вхідні змінні, а y_0-y_7 – вихідні.

Основна ідея запропонованої в роботі евристичної мінімізації – це розбити таблицю на дві частини, де більша частина таблиці розглядається як сукупність бітових векторів певної розрядності та для кожного вектора знайти мінімальне подання з врахуванням попередньо знайдених векторів та знову з'єднати таблицю в одне ціле. Ми обрали довжину вектора в 32 біти, що забезпечує реалістичні вимоги до пам'яті та обчислювальної складності в процесі мінімізації, зокрема дає змогу вести вичерпний перебір під час пошуку векторів. Перевагою такого підходу є можливість надалі використовувати різні логічні базиси та нарощувати довжину бітових векторів.

Отже, для кожної вихідної змінної y_0-y_7 довжиною 256-біт її значення послідовно розбиваються на вектори довжиною 32 біт: $y_i = y_{i255...224} \,|\, y_{i223...192} \,|\, ... \,|\, y_{i31...0} = y_i[j]$, i=0,...,7, j=0,...,7, які кодуються відповідним числом, що є десятковим представленням цього двійкового вектору. Набір усіх 64 векторів $y_i[j]$, що описують таблицю істинності, позначимо $sbox=\{y_i[j]\}$, i,j=0,...,7, а певне значення з *sbox* будемо позначати $sbox_i$, i=0,...,63.

Оскільки вектори $y_i[j]$ 32-бітні, то для мінімізації використовуються 5 вхідних змінних x_0-x_4: $y_i[j] = f(x_0, x_1, x_2, x_3, x_4)$. Змінні x_5-x_7 формують 8 масок-мінтермів m_j, що накладаються на вектори відповідних змінних y_0-y_7. Для *Sbox*0 таблиця істинності набуде вигляду, показаного в табл. 2.

Формула, що описує пропоноване подання:

$$yi = \bigcup_{j=0}^{7} \left(m_j \,\&\, yi[j] \right), \quad i = 0,...,7 \,. \tag{1}$$

Вхідними даними для алгоритму є вектори $x0$=0x*aaaaaaaa*, $x1$=0x*cccccccc*, $x2$=0x*f0f0f0f0*, $x3$=0x*ff00ff00*, $x4$=0x*ffff*0000. Також є тривіальні нульовий 0x0 та одиничний вектори inv=0x*ffffffff*.

Табл. 1. Фрагмент таблиці істинності для Sbox0 шифру "Kalyna"

x	x_7-x_0								y_0-y_7								y=*Sbox*[x]
0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0xa8
1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1	0x43
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
255	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0x80

Табл. 2. Подання Sbox0 в евристичному алгоритмі мінімізації

Маски m_j			*Вектори* $y_i[j] = f(x_0, x_1, x_2, x_3, x_4)$							
x_7	x_6	x_5	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	0e9a6e01	7b3b63f6	9bc32171	8d067ba4	b9fdc2d5	5ce51e6 c	348a161 e	fdbf9fb6
0	0	1	edd5ea76	15fab80 a	e9ea0a35	57576773	5ac413cb	2d8487bb	40e7e3ed	ce385f0 e
0	1	0	31161727	c70e2a73	36c7a4ea	103801cb	38893f41	5e33f329	4dffbcde	eac2940 c
0	1	1	2d604b6 c	8a206e96	0edf325 d	bed2be8 d	6588be36	bb34814 d	27f3ed98	aaa26ec8
1	0	0	cc73c925	1be583be	a58d9e0 c	fcf8ba75	181df377	1d329396	0969b0a5	d575f701
1	0	1	7e7f45c8	46248929	e6bbdca9	5d8b8843	2b7e3061	d31e9d08	e057e600	0ca9310 e
1	1	0	c923954 e	e5730dc6	78336197	1cf159d0	98be498 f	17194adb	669789a3	460947b3
1	1	1	ad4db831	5c3e7436	16c84a05	50026d7 a	38989aa1	577ab085	6cd07142	2349de60

Оцінимо кількість двовходових логічних вентилів GE за такого підходу для 8-бітного S-Box. Обчислення масок m_0–m_7 потребує 15 GE. Обчислення добутків m_j & $y_i[j]$ та їхніх сум потребує (8+7)×8=120 GE. Знаходження 64 векторів $y_i[j]$ із середнім числом операцій на один вектор x потребує 64x GE. Сумарна кількість вентилів буде дорівнювати:

$$Total\ GE = 15 + 120 + 64x = 135 + 64x\ . \qquad (2)$$

Отже, задачу мінімізації для запропонованого подання S-Box можна сформулювати так: задано набір із 5-х векторів (32-бітних чисел) x_0-x_4. Потрібно, використовуючи мінімум логічних операцій (проміжних змінних), обчислити всі 64 вектори $y_i[j]$, що описують конкретний S-Box.

Метод евристичної мінімізації. У нашому bitsliced представленні використовуються тільки стандартні логічні інструкції AND, OR, XOR, NOT і тому цей варіант можна реалізувати на будь-яких 8/16/32/64-бітних процесорах, разом із найпростішими 8-бітними MCU, а також на x86-64 процесорах із підтримкою векторних команд, де також є потрібні SIMD-інструкції AND, OR, XOR. Інструкція NOT емулюється операцією XOR, де один із векторних регістрів містить усі 1 (одиничний вектор *inv*). Вектори x_0-x_4 формують базу $base = \{x_0, x_1, x_2, x_3, x_4\}$, з якої починається пошук. Надалі до бази додаються значення проміжних змінних t та знайдених векторів $sbox_i$.

Послідовність проміжних змінних t та значень $sbox_i$ формують так звану трасу tr. Наприклад, якщо для знаходження значення $sbox_5$ потрібно виконати три вказані операції, то його траса буде така:

Операції:	Траса:
$t_0=x_0 \hat{} x_1$	$tr=[t_0 t_1 sbox_5]$
$t_0=t_0 \& x_4$	
$sbox_5=t_1 \mid x_3$	

Після знаходження кожного вектора $sbox_i$ відповідна йому траса заноситься в загальну трасу tr, з якої потім формується система логічних рівнянь, а проміжні значення t_i та $sbox_i$ додаються до бази:

$$base = base \cup tr = \{x_0, x_1, x_2, x_3, x_4, t_0, t_1, sbox_5\}.$$

Цю нову базу тепер будемо використовувати під час пошуку трас для решти $sbox_i$. Оскільки значення $sbox_5$ вже знайдено, воно вилучається з пошуку на наступних кроках. Кількість вентилів для побудови траси будемо позначати ge, наприклад, для цього випадку ge=3.

В алгоритмі на кожній ітерації використовується вичерпний пошук (англ. *Exhaustive Search*) усіх трас на глибину від 1 до 4 вентилів (рис. 2) з допомогою відповідних функцій, позначених q_0-q_3. Як буде показано далі, середня кількість вентилів на один вектор $sbox_i$ у нашому алгоритмі становить приблизно 3.6, тому заданої глибини пошуку, як правило, достатньо (окрім початку роботи алгоритму, про що буде йтися пізніше).

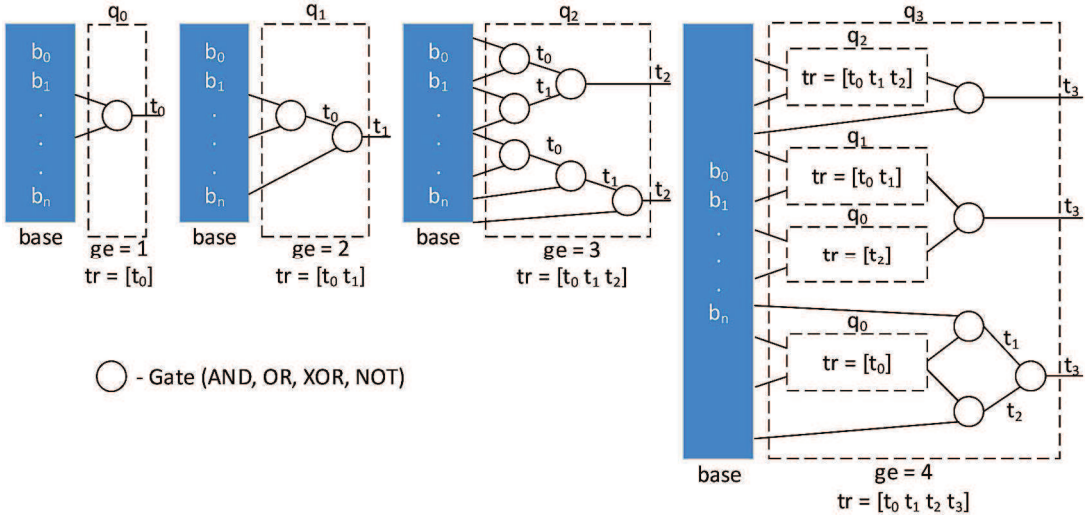

Рис. 2. Функції вичерпного пошуку на глибину до 4-х вентилів q_0-q_3

Алгоритм пошуку і вибору трас для значень $sbox_i$ складається з декількох кроків.

Крок 1. На першому кроці з допомогою функції *FIND_NEXT* здійснюється пошук і відбір усіх трас tr_{istep1}, що містять значення зі множини $sbox$ з мінімальною кількістю вентилів (глибиною пошуку) – рис. 3.

Результати Кроку 1 описано трьома параметрами:

- ge_{step1} – мінімальна глибина (довжина траси), на якій знайдено $sbox_i$;
- n_tr_{step1} – кількість знайдених трас;
- n_sbox_{step1} – кількість унікальних значень $sbox_i$ у всіх знайдених трасах.

Якщо n_tr_{step1}=1, то відповідна траса tr_{0step1} додається до загальної траси й бази, після чого пошук повторюється з Кроку 1.

Якщо n_tr_{step1}>1, то потрібно вирішити, яку саме трасу вибрати. Для цього використовуються наступні кроки, які здійснюють відбір таким способом, щоби ймовірно зменшити кількість вентилів у трасах під час пошуку наступних значень $sbox_i$. Переважно значення n_tr_{step1} становить декілька тисяч.

Крок 2. На другому кроці для кожної знайденої на Кроці 1 траси tr_{istep1} з допомогою швидкої функції *ESTIMATE* оцінюємо параметри ge_{step2}, n_tr_{step1}, n_sbox_{step2}, які можна отримати, якщо додати її до бази – рис. 4. Оскільки самі траси не будуються, а для кожної нової бази $base_{istep2}$ більшість елементів у ній є однакові (*base*), це дає змогу пришвидшити обчислення й ефективно обробити велику кількість трас-кандидатів, отриманих на Кроці 1.

Відбираються траси tr_{istep1} з мінімальним ge_{step2}, тобто ті, які дають змогу знайти наступне значення $sbox_i$ з мінімальною кількістю вентилів. Якщо є траси з однаковим мінімальним значенням ge_{step2}, то з них відбира-

ються траси з максимальним значенням n_sbox_{step2} – тобто траси, які забезпечують знаходження більшого числа $sbox_i$ з допомогою ge_{step2} вентилів (на рис. 4 відібрані траси, позначені зеленим кольором).

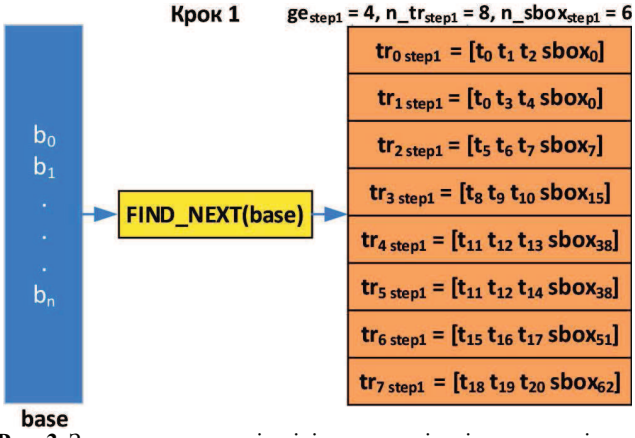

Рис. 3. Знаходження трас із мінімальною кількістю вентилів на Кроці 1

Якщо на Кроці 2 відібрана тільки одна траса tr_{istep1}, то вона додається до загальної траси й бази, після чого пошук повторюється з Кроку 1.

Якщо кількість відібраних трас більше одиниці, то знову потрібно вирішити, яку саме трасу вибрати. Для цього використовуються два наступні кроки, які здійснюють ітеративний відбір.

Крок 3. На третьому кроці для кожної з відібраних на Кроці 2 трас здійснюється пошук усіх трас з $sbox_i$ на глибині ge_{step2} з допомогою відповідної функції q_0-q_3. Після чого для базису з включеними з Кроку 2 і Кроку 3 відповідними трасами відбираються траси з мінімальним ge_{step3} та максимальним значеннями n_sbox_{step3} – рис. 5.

Якщо на Кроці 3 відібрана тільки одна траса tr_{istep3}, то вона і відповідна їй відібрана траса з Кроку 2 (з якої вона була згенерована) додаються до загальної траси й бази, після чого пошук повторюється з Кроку 1.

Якщо кількість відібраних трас більше одиниці, то переходимо до Кроку 4.

Крок 4. Аналогічний Кроку 3, але вхідними трасами виступають траси tr_{istep3} відібрані на Кроці 3 (позначені зеленим кольором на рис. 5). Якщо після Кроку 4 все ще залишається декілька кандидатів, то можна або завжди вибирати, наприклад першу трасу – це забезпечить відтворюваність результатів під час кожного запуску алгоритму пошуку, або вибирати серед трас випадково – це дасть змогу отримувати у разі кожного запуску алгоритму дещо інше bitsliced подання, й обрати серед них подання з мінімальним числом вентилів.

Рис. 4. Оцінка і відбір трас на Кроці 2

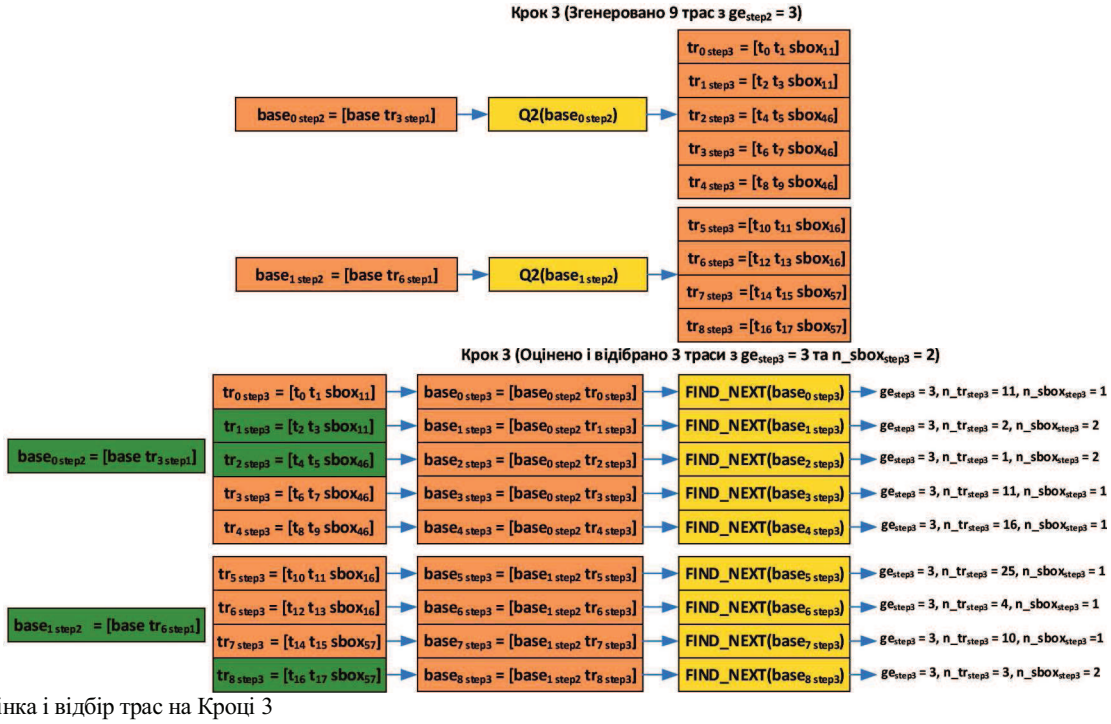

Рис. 5. Оцінка і відбір трас на Кроці 3

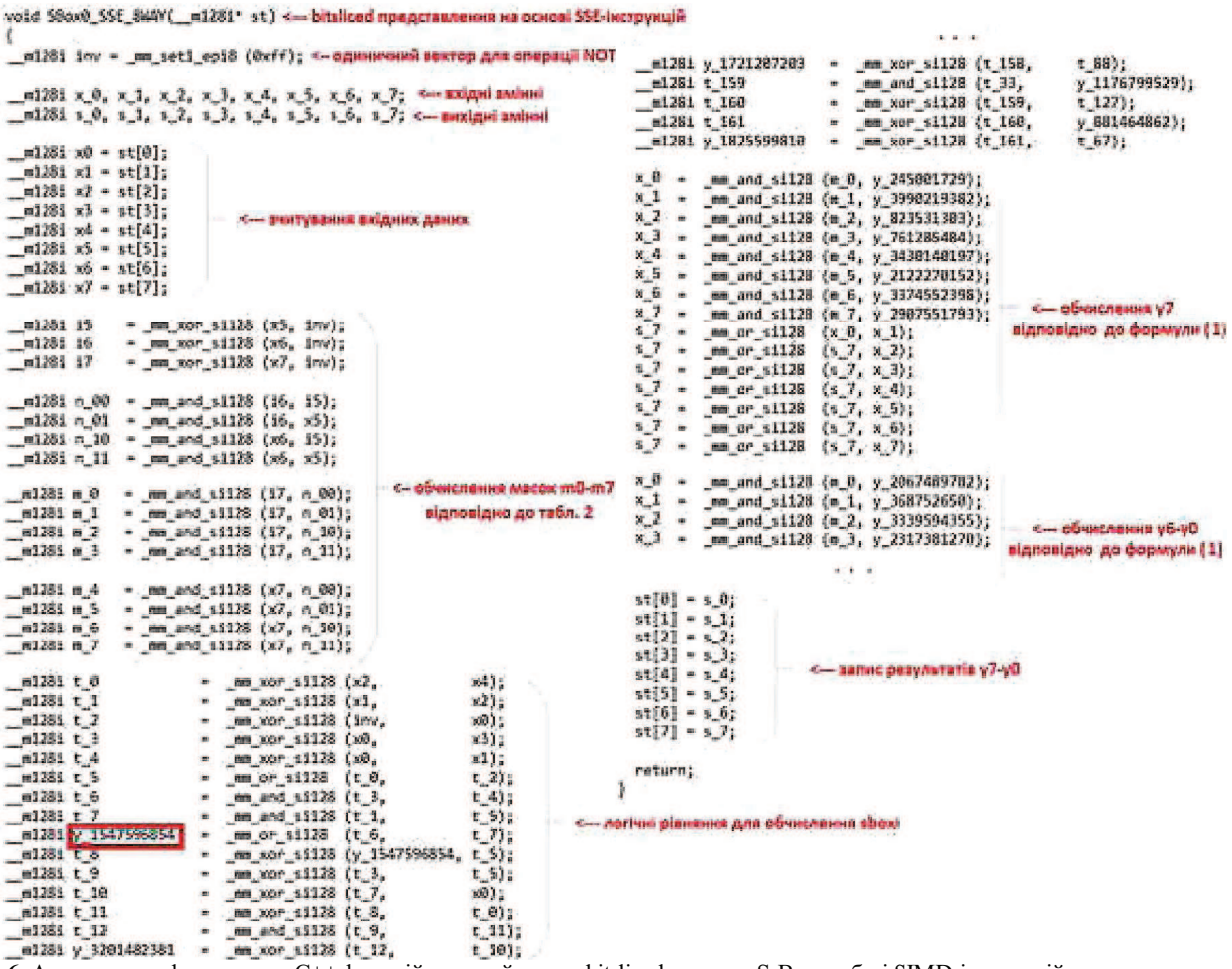

Рис. 6. Автоматичне формування C++ функцій для знайденого bitsliced подання S-Box на базі SIMD-інструкцій

Після знаходження всіх значень *sbox i*з загальної траси формується система логічних рівнянь відповідно до формули (1) для обчислення вихідних значень y_0-y_7. Ці рівняння записуються у файли у вигляді C++ функцій із використанням 128-, 256- і 512-бітних SIMD-інструкцій. Структуру C++ функції для 128-бітних SSE-інструкцій зображено на рис. 6.

Як йшлося, представлений алгоритм здійснює пошук $sbox_i$ на глибину до 4 вентилів. Але на старті алгоритму чи на початку роботи, коли база *base* є досить малою, знайти найближче значення з допомогою не більш ніж 4-х операцій не вдається. Наприклад, як видно на рис. 6, перше значення *y*_1547596854 знайдено тільки на 9-му кроці. Вести вичерпний пошук навіть на глибину більш ніж 6 є практично неможливо з погляду необхідної обчислювальної потужності та пам'яті. Тому ми цю проблему вирішуємо в такий спосіб. Якщо на Кроці 1 з допомогою повного перебору на глибину 4-х вентилів жодного значення $sbox_i$ не знайдено, то до бази додаються всі можливі значення t_i, знайдені на глибині 1 (*ge*=1) з допомогою функції q_0: $base_{q0} = base_{q0} \cup q_0(base)$. Після чого повторюється пошук на глибину 4 вже для бази $base_{q0}$. Для знайдених трас формуються логічні рівняння і відбираються траси з мінімальним сумарним *ge*.

На рис. 7 показано приклад роботи алгоритму на старті, коли $base = \{x_0, x_1, x_2, x_3, x_4\}$. Функція $q_0(base)$ формує 35 векторів, які додаються до наявної бази *base*, отже $base_{q0}$ буде містити 40 значень. На глибині *ge*=4 було знайдено одне значення $sbox_i$, яке потребує 5 проміжних значень t_0-t_4, породжених функцією $q_0(base)$. Знайдена траса з 9 вентилів (*ge*=9) додається до бази *base* й повторюється алгоритм пошуку з Кроку 1. Зазвичай після того, як база збільшується до 40 вентилів (7-8 знайдених $sbox_i$), потреби в розширенні бази, використовуючи q_0, немає.

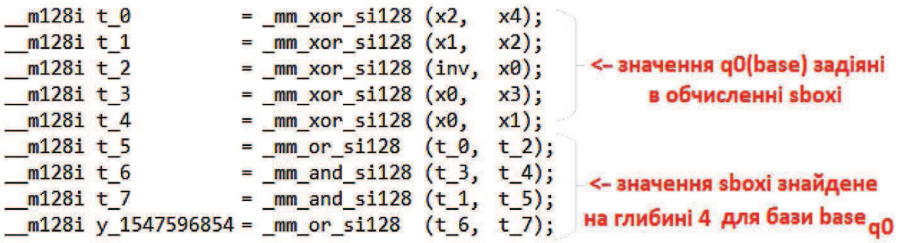

Рис. 7. Знаходження $sbox_i$ через розширення бази значеннями q0

У деяких випадках на старті алгоритму навіть розширення бази через q_0 може бути недостатньо, щоби знайти перше значення $sbox_i$. У такому випадку для $base_{q0}$ здійснюється частковий пошук на глибину 5 з допомогою функції q_4, як показано на рис. 8.

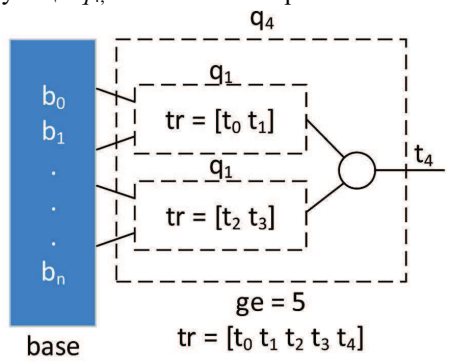

Рис. 8. Частковий пошук на глибину 5 функцією q_4

Результати роботи алгоритму евристичної мінімізації. Результати застосування розробленого алгоритму до різних S-Box наведено в табл. 3. Для досліджень ми обрали 8 таблиць нелінійної заміни з українського стандарту блокового шифру "Kalyna". "Kalyna" [11] належить до SPN-шифрів із довжиною ключа й розміром блоку 128/256/512 біт. У ньому використовуються 4 неалгебраїчні 8×8 S-Box для прямого (*Sbox*0-3) і 4 – для інверсного (*InvSbox*0-3) перетворення. Можливо проводити спільну мінімізацію декількох таблиць, наприклад двох таблиць: *Sbox*0 і *Sbox*1 (*Sbox*01),*Sbox*2 і *Sbox*3 (*Sbox*23), *InvSbox*0 і *InvSbox*1 (*InvSbox*01), *InvSbox*2 і *InvSbox*3 (*InvSbox*23), або чотирьох: *Sbox*0-*Sbox*3 (*Sbox*03) та *InvSbox*0-*InvSbox*3 (*InvSbox*03).

Також досліджували випадково згенеровані прямий (*Kuznyechik*) та інверсний (*InvKuznyechik*) S-Box SPN-шифру "Kuznyechik". Для порівняння розроблений алгоритм застосували до S-Box шифрів AES (*AES*/*InvAES*) і SM4 (SM4), хоча з практичного погляду застосування запропонованого евристичного алгоритму до них немає сенсу, оскільки їхня алгебраїчна структура допускає значно ефективнішу оптимізацію.

Табл. 3. Результати мінімізації 8×8 S-Box за критерієм BGC

S-Box	Total GE	GE/S-Box	GE/вектор
Sbox0	361	361	3.6
Sbox1	367	367	3.6
Sbox2	366	366	3.6
Sbox3	368	368	3.6
InvSbox0	364	364	3.6
InvSbox1	367	367	3.6
InvSbox2	362	362	3.6
InvSbox3	367	367	3.6
Kuznyechik	368	368	3.6
InvKuznyechik	361	361	3.6
AES	371	371	3.7
InvAES	364	364	3.6
SM4	364	364	3.6
Sbox01	663	332	3.2
Sbox23	667	334	3.2
InvSbox01	663	332	3.2
InvSbox23	664	332	3.2
Sbox03	1210	303	2.8
InvSbox03	1204	301	2.8

Обговорення результатів дослідження. Загалом для опису за критерієм BGC довільний 8×8 S-Box потребує приблизно 370 вентилів, а в разі спільної мінімізації декількох таблиць можна досягнути 300 вентилів/S-Box. Отримані результати значно кращі за ті, що можна здобути з використанням CAD-програм (1300 GE), утиліт на базі алгоритму Espresso (1500-1600 GE) чи алгоритму, описаному в патенті [4] (650-680 GE).

Висновок

1. Оглянуто методи bitsliced подання криптографічних S-Box. Запропоновано евристичний алгоритм формування bitsliced подання 8×8 S-Box за критерієм BGC, що допускає як програмну реалізацію для будь-яких процесорів, так і апаратну. Розроблено підходи для забезпечення стабільної роботи алгоритму на різних етапах функціонування.

2. Розроблено утиліту для автоматичного пошуку bitsliced подання та формування на його основі C++ коду на базі 128/256/512-бітних SIMD-інструкцій з SSE/AVX/AVX-512 розширень x86-64 CPU.

3. Застосовано розроблений алгоритм до S-Box деяких відомих БСШ, зокрема і вітчизняного шифру "Kalyna", та показано його істотну перевагу, порівняно з наявними алгоритмами мінімізації криптографічних S-Box.

4. Запропонований у роботі алгоритм може бути адаптований для пошуку bitsliced представлень 8×8 S-Box за критерієм GC чи з використанням спеціалізованих логічних інструкцій процесорів, таких як *andn* чи *vpternlogd*.

References

[1] Avraamova, O., Fomin, D., Serov, V., Smirnov, A., & Shokov, V. (2021). A compact bit-sliced representation of Kuznyechik S-box. *Mathematical Aspects of Cryptography, 12*(2), 21–38. https://doi.org/10.4213/mvk354

[2] Biham, E. (1997). A fast new DES implementation in software. In: Biham E. (Eds.) Fast Software Encryption. FSE 1997. *Lecture Notes in Computer Science*, *1267*, 260–272. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0052352

[3] Biryukov, A., Perrin, L., & Udovenko, A. (2016) Reverse-Engineering the S-Box of Streebog, Kuznyechik and STRIBOBr1. In: Fischlin M., Coron JS. (Eds.) Advances in Cryptology – EUROCRYPT 2016. *Lecture Notes in Computer Science, Vol.* 9665, 372–402. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-49890-3_15

[4] Borisenko, N., Vasinev, D., & Khoang, D. (2016). Method of forming s-blocks with minimum number of logic elements (RU Patent No. 2572423). *Federal service for intellectual property*. Retrieved from https://patents.google.com/patent/-RU2572423C2/enhttps://patents.google.com/patent/RU2014112547A/en

[5] Boyar, J., & Peralta, R. (2012). A Small Depth-16 Circuit for the AES S-Box. In: Gritzalis D., Furnell S., Theoharidou M. (Eds.) Information Security and Privacy Research. SEC 2012. IFIP *Advances in Information and Communication Technology*, Vol. 376, 287–298. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-30436-1_24

[6] Brayton, R., Hachtel, G., McMullen, C., & Sangiovanni-Vincentelli, A. (1984). Logic Minimization Algorithms for VLSI Synthesis. *Kluwer Academic Publishers*, Hingham, USA. https://doi.org/10.1007/978-1-4613-2821-6

[7] Canright, D. (2005). A Very Compact S-Box for AES. In: Rao J. R., Sunar B. (Eds.) Cryptographic Hardware and Embedded Systems – CHES 2005. CHES 2005. *Lecture Notes in Computer Science, Vol*. 3659, 441–455. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11545262_32

[8] Intel. (2021). *Intel Intrinsics Guide*. Retrieved from https://software.intel.com/sites/landingpage/IntrinsicsGuide/
[9] Käsper, E., & Schwabe, P. (2009). Faster and Timing-Attack Resistant AES-GCM. In: Clavier C., Gaj K. (Eds.) Cryptographic Hardware and Embedded Systems – CHES 2009. CHES 2009. *Lecture Notes in Computer Science*, *5747*, 1–17. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-04138-9_1
[10] Maximov, A., & Ekdahl, P. (2019). New Circuit Minimization Techniques for Smaller and Faster AES SBoxes. IACR *Transactions on Cryptographic Hardware and Embedded Systems*, 2(4), 91–125. https://doi.org/10.13154/tches.v2019.i4.91-125
[11] Oliynykov, R., et al. (2015). A New Encryption Standard of Ukraine: The Kalyna Block Cipher. IACR *Cryptology ePrint Archive, 2*(650). Retrieved from https://eprint.iacr.org/2015/650.pdf
[12] Raghuraman, S. (2019). *Efficiency of Logic Minimization Techniques for Cryptographic Hardware Implementation*. Masters Thesis, Virginia Polytechnic Institute and State University.
[13] Reyhani-Masoleh, A., Taha, M., & Ashmawy, D. (2018). Smashing the Implementation Records of AES S-box. IACR *Transactions on Cryptographic Hardware and Embedded Systems*, 2(2), 298–336. https://doi.org/10.13154/tches.v2018.i2.298-336
[14] Sovyn, Y., & Khoma, V. (2021). Bitsliced S-Box. Retrieved from https://drive.google.com/drive/folders/1yotZ4Hu5d3u0-A4SoQnSS__BrcNZDOKYh?usp=sharing
[15] Stoffelen, K. (2016). Optimizing S-Box Implementations for Several Criteria Using SAT Solvers. In: Peyrin T. (Eds.) Fast Software Encryption. FSE 2016. *Lecture Notes in Computer Science, Vol.* 9783, 140–160. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-52993-5_8

Ya. R. Sovyn, V. V. Khoma

Lviv Polytechnic National University, Lviv, Ukraine

HEURISTIC METHOD FOR BITSLICED REPRESENTATION OF RANDOMLY GENERATED 8×8 CRYPTOGRAPHIC S-BOX

The article is devoted to the issues of increasing the security and efficiency of software implementation for the symmetric block ciphers. For the implementation of cryptoalgorithms on low-end CPUs (8/16/32-bit microcontrollers), it is important to provide increased resistance to power consumption analysis attacks. With regard to the implementation of ciphers on high-end CPUs (x86, ARM Cortex-A), it is important to eliminate the vulnerability primarily to timing and cache attacks. The authors used a bitslice approach to securely implement block ciphers, which has potential advantages such as high speed and low computing resources. However, the known bitsliced methods have a significant limitation, since they work with deterministic S-Boxes or arbitrary S-Boxes of smaller sizes. The paper proposes a new heuristic method for bitsliced representation of cryptographic 8×8 S-Boxes containing randomly generated values. These values defy description using algebraic expressions. The method is based on the decomposition of the truth table, which describes the S-Box, into two parts. One part of the table forms logical masks, and the other is split into bit vectors. To find a logical description of these vectors an exhaustive search is used. After finding the description of all vectors, these two parts of the table are combined into one using logical operations. The use of this method oriented on software implementation in the logical basis {AND, OR, XOR, NOT} ensures the minimization of arbitrary 8×8 S-Boxes. The proposed method can be implemented using standard logical instructions on any 8/16/32/64-bit processors. It is also possible to use logical SIMD instructions from the SSE, AVX, AVX-512 extensions for x86-64 processors, which provides high performance due to the use of long registers. The corresponding software has been developed that implements the method of searching for bitsliced representations of a given S-Box, and also automatically generates C++ code for it based on SSE, AVX and AVX-512 instructions. The effectiveness of the method on the S-Box of known block ciphers, in particular the Ukrainian encryption standard "Kalyna", has been investigated. It was found that the developed algorithm requires almost half as many gates for the bitsliced description of an arbitrary S-Box than the best of known algorithm (370 gates versus 680, respectively). For ciphers that use two or four S-Box tables, joint minimization can yield up to 330 or 300 gates per table, respectively.

***Keywords**:* bitslicing; S-Box; logical minimization; SIMD; x86-64 CPU; software implementation; block ciphers.

Інформація про авторів:

Совин Ярослав Романович, канд. техн. наук, доцент, кафедра захисту інформації. **Email:** yaroslav.r.sovyn@lpnu.ua; https://orcid.org/0000-0002-5023-8442

Хома Володимир Васильович, д-р техн. наук, професор, кафедра захисту інформації. **Email:** volodymyr.v.khoma@lpnu.ua; https://orcid.org/0000-0001-9391-6525

Цитування за ДСТУ: Совин Я. Р., Хома В. В. Евристичний метод для bitsliced подання випадково згенерованих 8×8 криптографічних S-Box. *Український журнал інформаційних технологій.* 2021, т. 3, № 2. С. 58–65.

Citation APA: Sovyn, Ya. R., & Khoma, V. V. (2021). Heuristic method for bitsliced representation of randomly generated 8×8 cryptographic S-Box. *Ukrainian Journal of Information Technology, 3*(2), 58–65. https://doi.org/10.23939/ujit2021.02.058