Vol. 4, No. 1, 2022

DOI: https://doi.org/10.23939/csn2022.01.131

УДК 004.056:061.68

МЕТОД ТА УТИЛІТА ДЛЯ МІНІМІЗАЦІЇ BITSLICED-ПРЕДСТАВЛЕННЯ 4×4 S-BOXES

Я. Р. Совин, В. В. Хома, І. Р. Опірський

Національний університет «Львівська політехніка»,
кафедра захисту інформації
E-mail: yaroslav.r.sovyn@lpnu.ua, volodymyr.v.khoma@lpnu.ua, ivan.r.opirskyi@lpnu.ua

Статтю присвячено методам і засобам для генерації bitsliced-описів бієктивних 4×4 S-Boxes зі зменшеною кількістю вентилів/інструкцій. Згенеровані запропонованим методом bitsliced-описи дають змогу покращити безпеку та продуктивність як програмних імплементацій криптоалгоритмів, що використовують 4×4 S-Boxes, на різноманітних процесорних архітектурах, так і апаратних засобів на базі FPGA і ASIC.

У роботі розроблено евристичний метод мінімізації, що використовує стандартні логічні інструкції AND, OR, XOR, NOT, які доступні в більшості 8/16/32/64-бітних процесорах. Завдяки поєднанню в методі різних евристичних технік (попередніх обчислень, вичерпному пошуку на певну глибину, DFS-алгоритму, уточнюючому пошуку) вдалося зменшити кількість вентилів у bitsliced-описах S-Boxes порівнюючи з іншими відомими методами. Розроблено відповідне програмне забезпечення у вигляді утиліти мовою Python і протестовано її роботу на 225 S-Boxes різноманітних криптоалгоритмів. Встановлено, що розроблений метод у 57 % випадках генерує bitsliced-опис із меншим числом вентилів порівнюючи з найкращими відомими на сьогодні методами, реалізованими в утилітах LIGHTER /Peigen.

Ключові слова: bitslicing, 4×4 S-Box, CPU, логічна мінімізація, програмна імплементація, швидкодія.

Вступ

З огляду на постійно зростаючі обсяги та швидкості обробки даних дуже важливою вимогою до криптографічних алгоритмів (КА) є забезпечення достатньо високої продуктивності для широкого класу мікропроцесорних архітектур. Не менш важливим аспектом для програмної реалізації криптографічних алгоритмів є підвищена стійкість до атак через сторонні канали (side-channel attacks): 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.

Щоби забезпечити високу продуктивність криптоалгоритмів, використовуються різні підходи до їх програмної імплементації: це створення передобчислених таблиць (Lookup Tables, LUT) для певних операцій, інтеграція в процесор апаратних криптоакселераторів (напр. AES-NI у x86-процесорах), застосування SIMD-технології для розпаралелення процесу шифрування (напр. векторні інструкції SSE, AVX2, AVX-512 у x86-64 CPU) тощо. Проте всі ці підходи мають низку недоліків та обмежень, і не завжди можуть бути реалізовані в конкретному процесорі, особливо у low-end процесорах, що орієнтовані на IoT та вбудовані системи, і для яких характерними є обмежені ресурси та обчислювальні можливості.

Bitslicing [1] є одним із перспективних підходів, що забезпечує високопродуктивну constant-time імплементацію КА з імунітетом до часових та кеш-атак [2], максимально використовує можли-

вості сучасних high-end мікропроцесорів щодо збільшення швидкодії внаслідок розпаралелювання як виконання коду, так і обробки даних, а також допускає адаптацію для low-end CPU і апаратну реалізацію на FPGA і ASIC. Для багатьох КА саме bitsliced підхід забезпечує найбільшу швидкодію в програмній реалізації (якщо не використовуються апаратні криптоакселератори) для різного типу процесорних архітектур [1–7].

Базова ідея Bitslicing це конвертувати КА в послідовність бітових логічних операцій AND, XOR, OR, NOT. У процесорах кожну таку логічну операцію можна представити відповідною інструкцією, в апаратних засобах – відповідним вентилем. Висока швидкодія програмного Bitslicing досягається завдяки тому, що CPU обробляє багато елементів шифру (байтів, блоків) паралельно, використовуючи швидкі логічні інструкції та простішому виконанню деяких операцій (наприклад, бітових перестановок, зсувів тощо). Відсутність звернень до передобчислених таблиць у пам'яті та кеші і використання простих логічних інструкцій робить bitsliced-реалізації невразливими до часових і кеш-атак, та одночасно ускладнює атаки через сторонні канали.

Очевидно, щоб отримати максимальну швидкодію потрібно мінімізувати кількість логічних операцій, що входять до bitsliced-опису криптоалгоритму. Більшість криптографічних операцій породжують однозначний опис при переході до bitsliced-опису або не дають багато простору до мінімізації за винятком нелінійних перетворень. У КА операції нелінійної заміни задаються у вигляді $n \times m$ LUT-таблиць, так званих S-Boxes, що переважно мають розмір 4×4 ($n = 4$) або 8×8 ($n = 8$) біт. Таблиці 4×4 біт характерні як для легковагових криптоалгоритмів, спеціально розроблених для ефективної імплементації на обмежених у ресурсах процесорах (напр., блокові шифри PRINCE, LED, Piccolo, хеш-функції PHOTON, Spongent), так і криптоалгоритмів загального призначення (напр. блокові симетричні шифри Serpent, Twofish, хеш-функції BLAKE, Whirlpool).

Основною проблемою при bitsliced імплементації КА є представлення S-Boxes мінімально можливою кількістю логічних вентилів/інструкцій. Ця задача є NP-повною й допускає точний розв'язок лише для дуже простих випадків ($n \leq 3$ і деяких $n = 4$), тому більшість сучасних методів та утиліт для генерації bitsliced-опису S-Boxes використовують евристичні підходи, які не гарантують, що отриманий розв'язок є оптимальним з огляду на кількість вентилів, проте забезпечують значно кращий результат порівняно з універсальними методами мінімізації логічних функцій (напр., метод карт Карно чи метод простих імплікант Куайна-Мак-Класкі).

Отже, проблема знаходження оптимального bitsliced представлення навіть для невеликих S-Boxes (4×4) далека від вирішення, що потребує пошуку нових евристичних алгоритмів, один із яких представлений у нашій роботі.

Огляд літературних джерел

Найважчий етап в bitsliced імплементації, що значною мірою визначає швидкодію загалом, є логічне подання таблиць нелінійної заміни S-Boxes. У випадку апаратної реалізації в якості логічного базису 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 використовувати як синоніми. Слід зазначити, що в деяких процесорах відсутня інструкція NOT, яка емулюється інструкцією XOR, з огляду на те, що $\bar{x} = 1 \oplus x$. Оскільки процесорні логічні інструкції здебільшого обробляють два операнди, то й логічні вентилі повинні бути двовходові (рис. 1), щоби можна було однозначно перейти від логічного представлення до програмного.

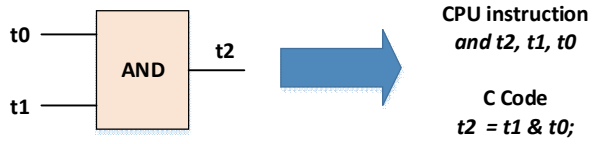

Рис. 1. Перехід від логічного до програмного bitsliced-представлення

Bitsliced-підхід до представлення криптоалгоритму був уперше запропонований E. Biham у роботі [1] для пришвидшення програмної імплементації шифру DES. У цій ж роботі E. Biham описав алгоритм bitsliced-представлення DES S-Boxes (6×4) логічними вентилями XOR, AND, OR, NOT. В алгоритмі із шести вхідних змінних вибираються дві вхідні змінні, що формують усі можливі комбінації за допомогою заданих логічних вентилів, з яких потім будуються чотири вихідні змінні. У середньому при bitsliced-опису за цим методом один DES S-Box потребує 100 вентилів.

У роботі [8] M. Kwan запропонував значно ефективніший підхід до знаходження bitsliced-представлення на прикладі DES S-Boxes. У ньому кожен вихідний біт S-Box розглядається як функція шести вхідних біт, що представлена картою Карно й розташовується у 64-бітній змінній. Усі вхідні та проміжні змінні теж можна розглядати як 6-бітні карти Карно, що описуються 64-бітними числами. Тоді задача сформулюється так: потрібно скомбінувати існуючі вхідні та проміжні карти таким способом, щоб отримати шукану вихідну змінну. Одна вхідна змінна виступає в якості селектора, що об'єднує функції п'яти змінних. Для пошуку представлення функцій п'яти змінних із мінімальною кількістю вентилів застосовується вичерпний перебір (brute force) та вентилі знайдені на попередніх кроках. Залежно від того, в якому порядку буде здійснюватися пошук доступні 6! варіантів для вхідних змінних та 4! варіантів для вихідних змінних. Це всього дає 17280 варіантів пошуку серед яких вибирається варіант із мінімальним числом вентилів. У результаті середнє число вентилів для bitsliced-опису одного DES S-Box зменшилося зі 100 до 56.

Алгоритм M. Kwan з деякими удосконаленнями реалізований у вигляді утиліти *sboxgates*, яка генерує bitsliced-опис для довільних S-Boxes до 8×8 включно [9]. Ця утиліта дає змогу задавати довільний набір двовходових вентилів, використовувати LUT-подібні тернарні логічні інструкції, що стали доступні в GPU та x86-CPU з підтримкою AVX-512, вказувати кількість ітерацій алгоритму пошуку, розпаралелювати пошук між процесорними ядрами тощо.

Для мінімізації S-Boxes можна застосувати програми SAT-Solvers, призначені для ефективного вирішення задачі здійсненності булевих формул (SATisfiability problem, SAT). Об'єктом задачі SAT є булева формула, що складається тільки з констант (0/1), змінних і операцій AND, OR, NOT. Задача полягає в такому: чи можна призначити всім змінним значення False і True так, щоби формула стала істинною. Спеціалізовані програми SAT-Solvers, побудовані на ефективних алгоритмах розв'язку, приймають на вході набір рівнянь і видають результат у вигляді *SAT*, якщо рішення знайдено і *UNSAT*, якщо рішення не знайдено. Щоби знайти логічну схему з заданою кількістю вентилів можна сформувати рівняння, де б змінні задавали всі можливі зв'язки між вентилями та операції й попробувати вирішити їх із допомогою SAT-Solvers. Перевагою цього підходу є те, що якщо знайдено рішення з n вентилів (*SAT*) і для $n - 1$ вентиля отримано *UNSAT*, то ми гарантовано знайшли мінімально можливий bitsliced-опис.

У роботі [10] SAT-Solvers були використані для знаходження bitsliced-представлення 4-бітних S-Boxes, а деякі отримані результати представлені в табл. 1, де критерій Bitslice Gate Complexity (BGC) позначає оптимальне рішення з мінімальною кількістю вентилів/операцій.

Таблиця 1

SAT-мінімізація S-Boxes за критерієм BGC [10]

S-Box	Size $n \times m$	Bitslice gate complexity
Prost	4×4	8 (4 AND, 4 XOR)
Piccolo/Piccolo^{-1}	4×4	10 (1 AND, 3 OR, 4 XOR, 2 NOT)
Lac	4×4	11 (2 AND, 2 OR, 6 XOR, 1 NOT)
Rectangle	4×4	$\in \{10,11,12\}$ (4 OR, 7 XOR, 1 NOT)
Rectangle^{-1}	4×4	$\in \{11,12\}$ (1 AND 3 OR, 7 XOR, 1 NOT)
Minalpher	4×4	≥ 11

Дані в табл. 1 слід інтерпретувати так. Наприклад, для S-Box шифру Piccolo вдалося з допомогою SAT-Solvers знайти bitsliced-представлення з 10 вентилів та довести, що представлення з BGC = 9 не існує (*UNSAT*) й отже BGC(Piccolo) = 10. Для S-Box шифру Rectangle вдалося знайти представлення з BGC = 12 та довести, що представлення з BGC = 9 не існує (*UNSAT*), проте не вдалося довести, що не існує рішення з BGC = 10 або 11. Для шифру Minalpher взагалі не вдалося знайти bitsliced-опису, а лише вдалося довести, що рішення з BGC = 10 не існує.

Отже, проблемою SAT-Solvers є те, що вони не завжди знаходять рішення для «важких" S-Boxes, таких як, наприклад, Minalpher, що можуть потребувати понад 12-13 вентилів. Для відносно простих S-Boxes з 11-13 вентилями SAT-Solvers не завжди можуть довести, що знайдене представлення є мінімальне, як це видно на прикладі Rectangle. Також недоліком цього методу є погана масштабованість: SAT-підхід працює лише для невеликих S-Boxes, розміром до 5×5, проте для 8×8 S-Boxes цей підхід неможливо реалізувати з погляду обчислювальної складності.

У роботі [11] теж пропонується використати SAT-Solvers для мінімізації S-Boxes. Відмінність підходу в тому, що спочатку з допомогою SAT-Solvers знаходять логічне представлення S-Box за критерієм Multiplicative Complexity (MC) – тобто представлення, яке містить мінімально можливу кількість нелінійних вентилів (AND). У такий спосіб логічне представлення S-Box розбивається на дві частини: нелінійну (вентилі AND) та лінійну (вентилі XOR, NOT), після чого лінійна частина мінімізується окремо, теж із використанням SAT-Solvers.

Для цього підходу характерні всі недоліки попередньо розглянутого підходу: знаходження Multiplicative Complexity теж є NP-повною задачею, яку за допомогою SAT-Solvers можна вирішити для відносно простих S-Boxes, погана масштабованість. Крім того, хоча рішення на кожному з двох кроків є оптимальним це не гарантує, що загальний розв'язок теж є оптимальним.

У роботі [12] описано open source утиліту LIGHTER, яка є найефективнішою на сьогодні утилітою для пошуку bitsliced-опису 4×4-бітних S-Boxes. LIGHTER має можливість гнучко задавати набір дво- і тривходових вентилів та їх вагові коефіцієнти, що враховуються при мінімізації. Це дає змогу більш реалістично здійснювати оптимізацію у випадку апаратної реалізації, коли різні логічні вентилі відрізняються площею кристалу, енергоспоживанням, затримкою тощо, через врахування цих параметрів у вагових коефіцієнтах. Для програмної імплементації, коли логічні інструкції є рівноцінні, достатньо встановити однакові вагові коефіцієнти для всіх вентилів.

Сам алгоритми пошуку LIGHTER комбінує два підходи: пошук за допомогою breath-first-search (BFS) алгоритму та meet-in-the-middle (MITM) стратегію. Тобто будуються два графи: один починається з базових векторів і здійснює пошук уперед, а інший стартує із шуканих векторів і здійснює пошук назад. Обидва графи рухаються один на зустріч одному, використовуючи задані логічні операції, поки вони не зустрінуться. Далі відбирається шлях, який поєднує ці два графи з мінімальною вартістю, яка враховує вагові коефіцієнти для кожного вентиля. Утиліта демонструє високу часову ефективність порівняно з SAT-методами, а її результати, які хоча не можуть вважатися оптимальними, досить близькі до результатів отриманих SAT-утилітами.

У роботі [13] описано open source утиліту Peigen (Platform for Evaluation, Implementation, and Generation of S-boxes), що дає змогу знаходити bitsliced-опис S-Boxes у різних логічних базисах, застосовуючи задані критерії мінімізації для апаратних і програмних імплементацій. Утиліта Peigen може оцінювати криптографічні властивості S-Boxes, генерувати S-Boxes за заданими критеріями, вести пошук оптимізованого за певними критеріями представлення S-Boxes, зокрема, й за критеріями BGC, MC та ін. Алгоритми пошуку bitsliced-опису утиліти Peigen спираються на алгоритми з утиліти LIGHTER [12], але покращено їх часову ефективність, зокрема, використано передобчислення та низку додаткових технік. Проте навіть із внесеними удосконаленнями утиліта ефективно працює лише з 4-бітними S-Boxes.

Постановка завдання

Метою нашої статті є представити метод та утиліту для генерації bitsliced-опису 4×4 S-Boxes, що забезпечують кращі результати порівняно з наявними, а це дасть змогу збільшити швидкодію та безпеку програмно-апаратних реалізацій широкого кола криптографічних алгоритмів, які використовують S-Boxes заданого типу.

Особливості представлення S-Boxes для bitsliced імплементації

У специфікаціях КА S-Boxes переважно задаються у вигляді LUT-таблиць. Наприклад, 4×4 S-Box шифру PRESENT має вигляд показаний у табл. 2.

Таблиця 2

LUT-таблиця S-Box шифру PRESENT

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S(x)	12	5	6	11	9	0	10	13	3	14	15	8	4	7	1	2

У bitsliced представленні LUT-таблиці розглядаються як логічні функції задані таблицями істинності. Наприклад, S-Box шифру PRESENT матиме вигляд показаний у табл. 3.

Таблиця 3

Bitsliced-орієнтоване представлення S-Box шифру PRESENT

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Hex
x_0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	*0xff00*
x_1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	*0xf0f0*
x_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	*0xcccc*
x_3	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	*0xaaaa*
S(x)	12	5	6	11	9	0	10	13	3	14	15	8	4	7	1	2	
y_0	1	0	0	1	1	0	1	1	0	1	1	1	0	0	0	0	*0x0ed9*
y_1	1	1	1	0	0	0	0	1	0	1	1	0	1	1	0	0	*0x3687*
y_2	0	0	1	1	0	0	1	0	1	1	1	0	0	1	0	1	*0xa74c*
y_3	0	1	0	1	1	0	0	1	1	0	1	0	0	1	1	0	*0x659a*

Отож, компактне представлення S-Box у вигляді таблиці істинності буде мати вигляд: $S(x) = y$, де $x = \{x_0, x_1, x_2, x_3\} = \{0xff00, 0xf0f0, 0xcccc, 0xaaaa\}$ – вхідні bitsliced змінні, $y = \{y_0, y_1, y_2, y_3\} = \{0x0ed9, 0x3687, 0xa74c, 0x659a\}$ – вихідні bitsliced змінні, що визначають конкретну таблицю заміни, а 16-бітні числа, що задають x та y будемо називати векторами.

Задачу пошуку bitsliced представлення S-Box за критерієм BGC можна сформулювати так: задано чотири базові вектори $base = \{x_0, x_1, x_2, x_3\}$, потрібно знайти вектори $y = \{y_0, y_1, y_2, y_3\}$ використовуючи мінімальну кількість 2-входових логічних вентилів NOT, AND, OR, XOR.

Попередні обчислення

На етапі передобчислень одноразово знаходяться та зберігаються певні дані, які потім багатократно використовуються в нашому алгоритмі пошуку bitsliced-опису. Ці дані є двох типів:

1. Для кожного 16-бітного вектора v знаходиться BGC(v) – мінімальна кількість вентилів де потрібна для його представлення, так звана «складність" вектора.

Оскільки вектори представляються 16-бітними числами, усього є 65536 векторів, з них чотири це базові вектори $base = \{x_0\text{-}x_3\}$ і два це логічні константи $const = \{0x0000, 0xffff\}$ для позначення 0 і 1 для яких BGC рівне 0, тому залишається 65530 векторів, складність яких потрібно оцінити.

У табл. 4 представлено знайдений розподіл векторів за їх значенням BGC.

Як видно з табл. 4 максимальна складність рівна 8, тобто будь-який 16-бітний вектор можна представити за допомогою не більш ніж 8 вентилів {XOR, AND, OR, NOT}. Це дає верхню оцінку bitsliced-складності довільного S-Box, що описується чотирма векторами $y_0\text{-}y_3$, рівну 32-м вентилям.

Таблиця 4

Розподіл 16-бітних векторів за BGC

BGC	0	1	2	3	4	5	6	7	8
Кількість векторів	6	22	126	691	3181	12639	27165	19670	2036

Для непрямої попередньої оцінки «складності" S-Box можна використати такий показник як сумарне значення складності векторів y_0-y_3, що його описують: чим ближче до 32-х сумарне значення, тим більше вентилів слід очікувати у bitsliced-описі й навпаки. Наприклад, мінімальне сумарне значення 8 (2/2/2/2) зі всіх S-Boxes, які розглядаються в статті, має S-Box CS_cipher_F (bitsliced-опис теж потребує 8 вентилів), а максимальне сумарне значення 30 (7/7/8/8) має S-Box Magma_4 (bitsliced-опис потребує 15 вентилів).

2. Побудова LUT-таблиці для представлення всіх графів на глибину ge = 6 вентилів.

Побудова таблиці здійснюється покроково. На першому кроці будується таблиця $\boldsymbol{q_0}$, яка містить усі можливі значення, що можна отримати з базових векторів x_0-x_3 за допомогою одного вентиля (інструкції) і які не входять в *base* і *const*. Для цього використовується алгоритм *VECT_SQUARE*, який утворює з вхідних векторів усі можливі комбінації за допомогою операцій XOR, AND, OR, NOT.

Таких векторів загалом є 22. Згенеровані значення заносяться в таблицю, а значення x_0-x_3 у таблиці не зберігаються для економії пам'яті, хоча присутні для кожного рядка неявно. Отже, таблиця $\boldsymbol{q_0}$ = *VECT_SQUARE*($\{x_0, x_1, x_2, x_3\}$) має розмірність 22×1 (рис. 2).

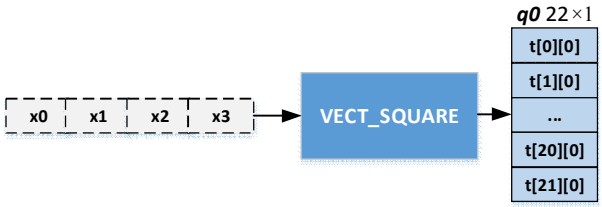

Рис. 2. Формування таблиці $\boldsymbol{q_0}$

Кожен *i*-й рядок таблиці $\boldsymbol{q_0}$ можна розглядати як новий базис $\{x_0$-$x_3, q_0[i]\}$, який використовується для генерації всіх можливих векторів на наступному кроці, а самі рядки LUT таблиць будемо називати графами.

Так, зокрема, на другому кроці генерується таблиця $\boldsymbol{q_1}$ = *GEN_TABLE*($\boldsymbol{q_0}$), для чого з кожного рядка таблиці $\boldsymbol{q_0}$ знову алгоритмом *VECT_SQUARE* породжуються всі можливі значення, що можна отримати з базових векторів x_0-x_3 та $q_0[i]$ за допомогою одного вентиля, формуються рядки з двох векторів та додаються в загальну таблицю. Після цього здійснюється фільтрація логічно еквівалентних схем: якщо рядки таблиці $\boldsymbol{q_1}$ містять однакові значення в довільному порядку, то з них залишається лише один рядок (рис. 3).

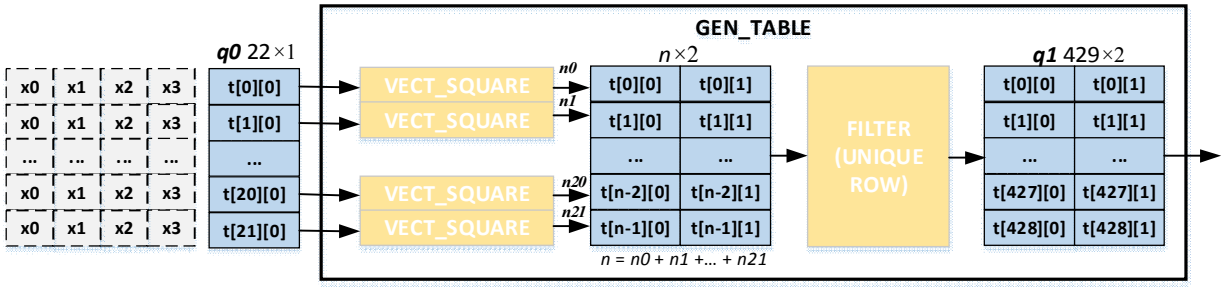

Рис. 3. Формування таблиці $\boldsymbol{q_1}$

Таблиця q_1 має розмірність 429×2, а коефіцієнт розгалуження *branching* при переході від таблиці q_0 до q_1 рівний 429/22 = 19.5. Тепер таблиця q_1 може бути використана для побудови аналогічним чином таблиці $q_2 = GEN\_TABLE(q_1)$ і т. ін.

Продовжуючи побудову таблиць описаним чином ми дійдемо до LUT-таблиці q_5, що містить усі можливі графи *з ge* = 6 вентилів, а отримані покрокові результати представлені в табл. 5.

Таблиця 5

Властивості LUT-таблиць q_0-q_5

Таблиця	q_0	q_1	q_2	q_3	q_4	q_5
Розмірність	22×1	429×2	8593×3	186434×4	4462108×5	118491958×6
Branching	-	19.5	20.0	21.7	23.9	26.6

Подальша побудова передобчислених таблиць недоцільна, оскільки вимагає занадто багато пам'яті: таблиця q_5 для зберігання потребує 1.3 Гбайт, що при прогнозованому коефіцієнті розгалуження близько 28 дає оцінку розміру таблиці q_6 у 40 Гбайт і понад 1 Тбайт для q_7.

Алгоритм пошуку bitsliced-представлення

На верхньому рівні алгоритму пошуку здійснюється перебір усіх значень y_0-y_3, генерація для кожного з них із передобчисленої LUT-таблиці ***q5*** матриці графів-кандидатів $gr_i = STEP\_0(y_i)$ і передача їх у алгоритм пошуку в глибину $FIND\_BS(gr_i)$. Алгоритм пошуку в глибину *FIND_BS* знаходить решту значень *y* намагаючись використати мінімум вентилів і повертає побудовані доповнені матриці графів gr_0-gr_3. З отриманих результатів вибираються графи з мінімальним значенням BGC (рис. 4).

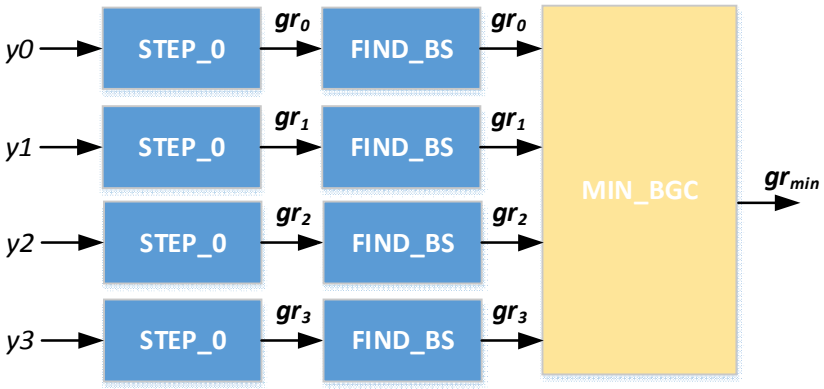

Рис. 4. Узагальнена структура алгоритму пошуку bitsliced-опису S-Box

Отже, алгоритм пошуку здійснює чотири ітерації, починаючи роботу з різних значень *y*. Позначимо це початкове значення y_{start}. На етапі $gr_i = STEP\_0(y_{start})$, використовуючи LUT-таблицю ***q5***, генерується матриця графів gr_i, що містить усі можливі графи з вектором y_{start} на певній глибині d_{start} вентилів. Залежно від того до якої BGC-групи належить вектор y_{start} евристично підібрані значення d_{start}, представлені в табл. 6, щоби забезпечити прийнятні час обчислення та обсяг потрібної пам'яті.

Таблиця 6

Глибина генерації графів, що містять y_{start} на кроці *STEP_0*

BGC-група y_{start}	1	2	3	4	5	6	7	8
d_{start}	6	6	7	8	8	8	8	9

Отже, якщо, наприклад, $bgc(y_0) = 1$, то матриця графів $\boldsymbol{gr_0}$ після *STEP_0* буде містити всі графи довжиною 6 вентилів ($d_{start} = 6$) у яких зустрічається вектор y_0.

Далі всі графи кандидати в $\boldsymbol{gr_i}$ сортуються на три групи: ***gr_1y***, ***gr_2y***, ***gr_3y*** з однаковою кількістю векторів y у кожному графі групи – 1, 2 і 3 відповідно. Позначимо цю кількість *y_find*. Далі пошук ведеться для кожної непорожньої групи окремо відповідно до рис. 5.

Алгоритм *FIND_NEXT* здійснює почерговий пошук y_i, поки не будуть знайдені всі чотири значення y_0-y_3. На вхід подається матриця графів ***gr*** у вигляді таблиці $n \times m$, кожен рядок якої містить *y_find* значень зі множини $\{y_0\text{-}y_3\}$. Кожен рядок таблиці зберігає у явному вигляді m векторів та вектори x_0-x_3 неявно.

Спочатку для групи ***gr*** здійснюється оцінка мінімальної віддалі d_{min}, на якій розташовується найближче значення y_x серед усіх графів - *ESTIMATE_DEPTH*. Для цього розроблена швидка функція *FAST_FIND* вичерпного пошуку вперед на задану глибину 1/2/3/4 кроки. Пошук і відсікання варіантів ведеться за допомогою алгоритму пошуку в глибину з ітеративним заглибленням – IDDFS (Iterative Deepening Depth-First Search).

Якщо в наборі ***gr*** на всіх глибинах пошуку (1/2/3/4) жодного значення y_x не знайдено ($d_{min} \geq 5$), тоді робиться крок уперед і з таблиці ***gr*** за допомогою *GEN_TABLE* генерується нова таблиця розміром уже $n_{new} \times (m + 1)$, після чого повторюється пошук і т. д – рис. 6.

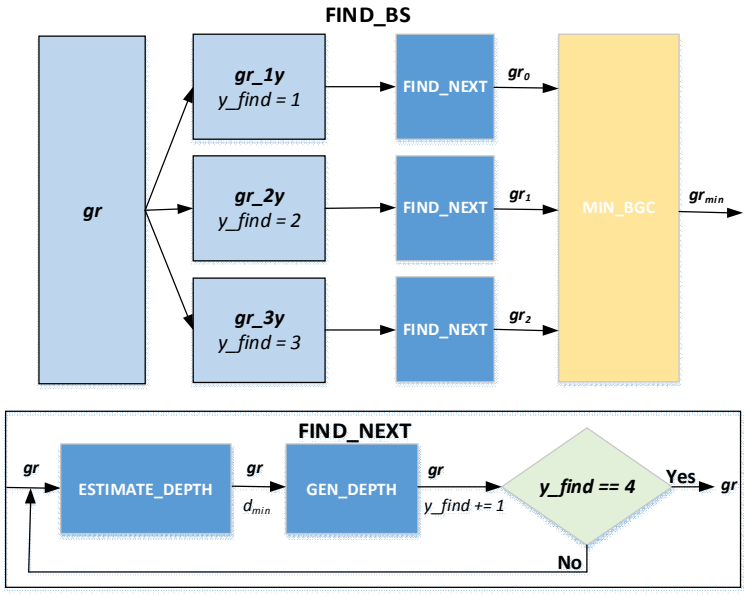

Рис. 5. Узагальнена схема пошуку bitsliced представлення алгоритмом FIND_BS

Після того як знайдена оцінка $d_{min} \leq 4$ за допомогою алгоритму *GEN_DEPTH* здійснюється перехід від набору графів з $y\_find = n\_y$ до набору графів з $y\_find = n\_y + 1$.

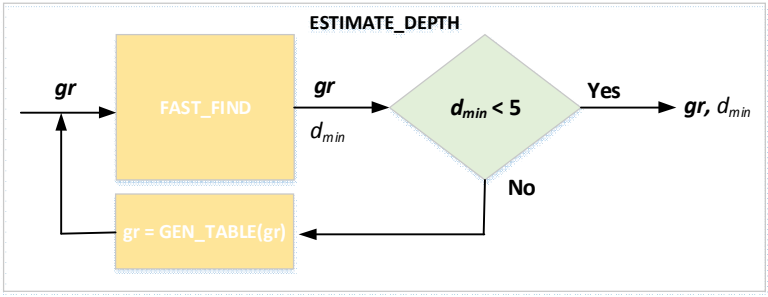

Рис. 6. Оцінка глибини пошуку для знаходження наступного вектора y_x

Для цього з групи ***gr*** відбираються графи з знайденим значенням d_{min} і для цієї групи $\boldsymbol{gr_{min}}$ робиться крок уперед ***gr*** = *GEN_TABLE*($\boldsymbol{gr_{min}}$). Для згенерованого набору ***gr*** знову відбираються графи з знайденим значенням $d = d_{min} - 1$, для них робиться крок уперед і так далі, поки d не стане рівним 0. Після цього в групу відбираються лише ті графи, що містять $n\_y + 1$ значень y. Далі ці кроки повторюються, поки не будуть знайдені всі значення y.

Алгоритм *FIND_BS* на кожному кроці здійснює оцінку мінімальної віддалі d_{min}, на якій розташовується найближче значення y_x та генерує відповідні графи. Як показано на рис. 7 цей маршрут починається з графів, що містять y_a, згенерованих за допомогою *STEP_0*, від яких найближче значення y_b розташоване на відстані d_{ab} вентилів, далі переходимо до y_c розташованого на мінімальній відстані d_{bc} від y_b та на відстані d_{cd} знаходимо останній вектор y_d.

Проте не завжди рух мінімальними кроками по траєкторії від вектора y_a до y_d дає оптимальний результат загалом (хоча це є так у більшості випадків). Може бути ситуація, коли вибір мінімального значення d на перших кроках, призводить до більших значень d на наступних кроках, і в результаті – до неоптимального логічного представлення. Наприклад, будемо вважати, що на першому кроці ми отримали $d_{ab} = 1$, на другом $d_{bc} = 4$ і на третьому – $d_{cd} = 3$, тобто маршрут складе сумарно 8 вентилів (рис. 7), проте можливо, що якщо б на першому кроці ми пішли іншим маршрутом, і були б відібрані графи з $d_{ab} = 2$, то на другому кроці вдалося б знайти значення y_c з $d_{bc} = 3$ і на третьому y_d з $d_{cd} = 2$, й ми отримали б коротший сумарний маршрут із 7 вентилів. Отже, другий маршрут привів до bitsliced представлення з меншим значенням BGC.

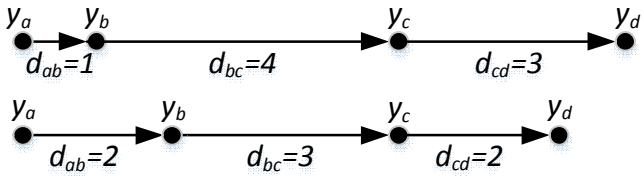

Рис. 7. Знаходження bitsliced-опису для різних маршрутів

Щоби врахувати різні можливі маршрути в алгоритмі пошуку проводяться уточнюючі пошуки за схемою, яка представлена на рис. 8. Якщо ми маємо набір графів, що містять 3 з 4 можливих значення y, то пошук четвертого значення завжди здійснюється на мінімально можливій глибині d_{min} (*SEARCH_3Y*). Для графів із двома значеннями y (y_find = 2) пошук третього значення відбувається за двома маршрутами: d_{min} і d_{min} + 1, після чого для знайдених графів з y_find = 3 застосовується пошук *SEARCH_3Y*. Для графів з одним значенням y (y_find = 1) пошук другого значення відбувається за трьома маршрутами: d_{min}, d_{min} + 1 і d_{min} + 2, після чого для знайдених графів з y_find = 2 застосовується пошук *SEARCH_2Y*.

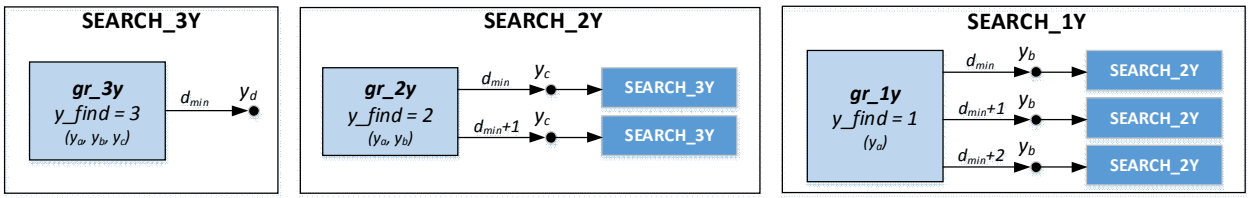

Рис. 8. Схема уточнюючого пошуку в алгоритмі FIND_BS

Результати

Запропонований у роботі метод був імплементований мовою Python, а для забезпечення швидкодії основні функції обробки даних реалізовані на базі бібліотеки numpy.

Для оцінки нашого алгоритму було взято 225 4×4 S-Boxes різноманітних криптографічних алгоритмів. Нами було використані open source проєкти sboxgates, LIGHTER і PEIGEN, щоб отримати з

їхньою допомогою оцінку BGC для обраних S-Boxes, і мати можливість порівняти з нашими результатами. Bitsliced-описи S-Boxes отримані нашим методом доступні за посиланням [14].

Результати представлені в табл. 7. Дані стовпців у табл. 7 слід інтерпретувати так:

LUT – представлення S-Box у табличному вигляді, де рядок '0123456789abcdef' слід розуміти як $S(x) = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$.

BS – представлення S-Box у bitsliced-форматі. Рядок '0ed9_3687_a74c_659a' слід розуміти так: $y_0 = 0x0ed9$, $y_1 = 0x3687$, $y_2 = 0xa74c$, $y_3 = 0x659a$.

CY – BGC векторів y_0-y_3. Рядок '6285' слід інтерпретувати так: $BGC(y_0) = 6$, $BGC(y_1) = 2$, $BGC(y_2) = 8$, $BGC(y_3) = 5$.

OURS – містить значення BGC отримане за допомогою описаного у статті методу.

L/P – містить значення BGC отримане за допомогою утиліт LIGHTER/PEIGEN [12, 13]. Ці утиліти використовують однаковий алгоритм пошуку, проте внаслідок оптимізацій іноді можуть давати різний результат для одного й того ж S-Box, у цих випадках вибиралося мінімальне значення.

SG – містить значення BGC отримане за допомогою утиліти sboxgates, число ітерацій пошуку було задано 10000 [9].

Червоним кольором позначені S-Boxes, що мають більше значення BGC порівнюючи з отриманим нашим алгоритмом, жовтим – що мають однакове з нашим алгоритмом значення BGC.

Таблиця 7

Порівняння BGC для різних S-Boxes

S-Box	LUT	BS	CY	OURS	L/P	SG
1	2	3	4	5	6	7
Piccolo	e4b238091a7f6c5d	cd94_1e1d_fc03_aaa5	6533	10	11	12
Piccolo^{-1}	68341eca5792df0b	b714_aaa5_3369_b4e2	7346	10	11	12
Lac	e9f0d4ab128376c5	9996_3ac5_f035_44d7	3566	11	11	11
Prost	048f15e927acbd63	b2b8_d748_6a6a_3ccc	5622	8	8	8
Rectangle	65ca1e79b03d8f42	2dd2_a569_6867_39ac	4466	12	12	16
Rectangle^{-1}	94fae106c7382b5d	e625_369c_c396_a91d	7437	12	12	16
Minalpher	b34128cf5de069a7	a38b_d493_97c4_66e1	7876	15	16	17
SKINNY	c6901a2b385d4e7f	cd94_e1e2_fc03_aaa5	6533	11	11	13
TWINE	c0fa2b9583d71e64	1ee4_6a3c_ec85_256d	6577	15	15	19
PRINCE	bf32ac916780e5d4	62c7_131f_f322_5473	7557	16	18	20
Lucifer_S0	cf7aedb026319458	5c66_075e_6237_907b	5657	15	17	18
Lucifer_S1	72e93b04cd1a6f85	a639_3837_b385_6b2c	7577	16	18	20
PRESENT	c56b90ad3ef84712	659a_a74c_3687_0ed9	4777	14	14	18
PRESENT^{-1}	5ef8c12db463079a	69a5_ad46_2697_c19e	4786	14	14	18
JH_S0	904bdc3f1a26758e	31d9_9ec8_b8b4_c2b9	7658	16	16	17
JH_S1	3c6d5719f204bae8	11f9_7325_493e_f18a	7766	16	18	19
Iceberg_S0	d7329ac1f45e60b8	4597_592e_1f43_c971	7678	15	16	19
Iceberg_S1	4afc0d9be6173582	3ce4_9b86_2b2d_41ee	6764	15	15	20
Luffa	de015a76b39cf824	1759_53e2_98d3_3d23	7786	13	14	18
Noekeon	7a2c48f0591e3db6	7741_d847_a959_6a6a	6852	12	12	12
Hb1_S0	865f1ca9eb2470d3	d29c_974a_592e_43e9	6666	15	16	22
Hb1_S1	07e15b823ad6fc49	953a_1ba6_7c16_b664	6766	14	15	20
Hb1_S2	2ef5c19ab468073d	e16c_6587_a61e_89d6	6666	16	16	20
Hb1_S3	0734c1afde6b2895	c9a6_1ec6_879a_6bd0	6666	14	16	20
Hb1_S0^{-1}	d4afb21c07695e83	9a59_a63c_368b_689d	6587	15	16	17
Hb1_S1^{-1}	0378e4b16f95da2c	1ec6_6356_9b34_b658	6666	15	15	16
Hb1_S2^{-1}	c50e93adb6784f12	65b2_a768_368b_29d9	6686	16	16	21

Продовження табл. 7

1	2	3	4	5	6	7
Hb1_S3^{-1}	05c23fa1de6b4897	c9b2_8e78_9726_6b64	6676	14	16	22
Hb2_S0	7ce9215fb6d048a3	85e9_c395_16c7_658e	7686	15	16	22
Hb2_S1	4a168f7c30ed59b2	7964_c56a_1ce9_6cb2	7576	15	16	22
Hb2_S2	2fc156ade8340b97	e49a_a563_89b6_63c6	7665	15	15	18
Hb2_S3	f4589721a30e6cdb	c2b5_9b61_7827_e919	7766	15	16	19
Hb2_S0^{-1}	b54fc690d3e81a27	934b_e629_853e_2d59	7777	16	16	22
Hb2_S1^{-1}	92f80c364d1e7ba5	b645_78c6_9ba4_6a2d	7667	16	17	20
Hb2_S2^{-1}	c30ab45f9e6d2781	a9d2_369a_2ee1_4b99	6666	15	15	19
Hb2_S3^{-1}	a76912c5348fdeb0	599a_6927_3ac6_7c49	5768	15	16	19
DES_S0_0	e4d12fb83a6c5907	b16c_8771_9c27_2ae5	5766	15	16	21
DES_S0_1	0f74e2d1a6cb9538	78c6_4b36_265e_9d52	6676	13	14	16
DES_S0_2	41e8d62bfc973a50	5d92_39e4_4b35_279c	6586	14	15	21
DES_S0_3	fc8249175b3ea06d	87e1_5e89_c993_9a27	6868	15	15	21
DES_S1_0	f18e6b34972dc05a	4b63_8679_5a99_992d	6656	14	14	20
DES_S1_1	3d47f28ec01a69b5	e41b_58b9_919e_69d2	5855	15	15	19
DES_S1_2	0e7ba4d158c6932f	b1cc_e81e_8d66_965a	5653	12	12	15
DES_S1_3	d8a13f42b67c05e9	a539_47b4_6e61_c927	6666	15	17	18
DES_S2_0	a09e63f51dc7b428	1be4_5879_2ed8_964d	4777	15	16	19
DES_S2_1	d709346a285ecbf1	e41b_69d2_5c63_7a89	5567	14	15	19
DES_S2_2	d6498f30b12c5ae7	9369_e562_d827_6939	6755	13	15	17
DES_S2_3	1ad069874fe3b52c	3aa5_5e92_a794_9666	6553	13	14	17
DES_S3_0	7de3069a1285bc4f	994b_92ad_e827_b4c6	6776	15	16	20
DES_S3_1	d8b56f03472c1ae9	92ad_66b4_4b39_e827	7677	14	17	19
DES_S3_2	a690cb7df13e5284	17e4_2d63_99d2_49b5	6757	15	16	22
DES_S3_3	3f06a1d8945bc72e	2d63_e81b_b64a_99d2	7765	15	16	18
DES_S4_0	2c417ab6853fd0e9	9e58_4cf1_5a96_d962	6745	15	16	20
DES_S4_1	eb2c47d150fa3986	35e2_9c27_8579_6c4b	7677	15	16	19
DES_S4_2	421bad78f9c5630e	2b6c_b15a_9d61_87b8	7575	16	17	20
DES_S4_3	b8c71e2d6f09a453	ca99_9369_63ac_1aa7	6657	15	16	19
DES_S5_0	c1af92680d34e75b	e61a_b46c_7a49_929d	6666	15	16	19
DES_S5_1	af427c9561de0b38	66d2_691b_0db6_ac63	6766	15	17	22
DES_S5_2	9ef528c3704a1db6	718d_c996_a54e_6867	7566	16	16	19
DES_S5_3	432c95fabe17608d	8d72_1bc6_9a69_c3d8	4665	14	14	20
DES_S6_0	4b2ef08d3c975a61	9d92_691e_5a99_26da	5556	14	15	19
DES_S6_1	d0b7491ae35c2f86	266d_b38c_ad19_69a5	7584	14	14	17
DES_S6_2	14bdc37eaf680592	626d_87e4_26da_4b9c	6666	15	16	20
DES_S6_3	6bd814a7950fe23c	4b96_78c3_9aa5_994e	5555	14	14	18
DES_S7_0	d2846fb1a93e50c7	96e1_8d72_d839_4b65	6477	14	15	18
DES_S7_1	1fd8a374c56b0e92	4a67_ac72_27c6_691e	8765	15	16	20
DES_S7_2	7b419ce206adf358	781b_36c3_5a65_9c72	7555	14	15	20
DES_S7_3	21e74a8dfc90356b	b58a_d12d_639c_87e4	5646	14	15	17
Serpent_S0	38f1a65bed42709c	52cd_19b5_9764_c396	8863	14	14	19
Serpent_S1	fc27905a1be86d34	6359_568d_b44b_2e93	7848	14	14	20
Serpent_S2	86793cafd1e40b52	639c_a4d6_4da6_25e9	4766	13	13	19
Serpent_S3	0fb8c963d124a75e	63a6_b4c6_e952_913e	6667	15	15	21
Serpent_S4	1f83c0b6254a9e7d	d24b_69ca_e692_b856	6675	15	15	20
Serpent_S5	f52b4a9c03e8d671	d24b_662d_7493_1ce9	6687	15	15	18
Serpent_S6	72c5846be91fd3a0	3e89_69c3_196d_5b94	8486	14	14	21
Serpent_S7	1df0e82b74ca9356	7187_a9d4_c716_1cb6	7676	16	16	19

Продовження табл. 7

1	2	3	4	5	6	7
Serpent_S0^{-1}	d3b0a65c1e47f982	3947_9a36_1ee1_7295	8648	14	14	19
Serpent_S1^{-1}	582ef6c3b4791da0	3d91_45bc_2679_695a	7774	14	14	20
Serpent_S2^{-1}	c9f4be12036d58a7	9a56_c6b4_9c2d_6837	4676	13	13	22
Serpent_S3^{-1}	09a7be6d35c248f1	c39a_497c_56e8_64b6	5766	15	15	20
Serpent_S4^{-1}	5083a97e2cb64fd1	e469_2dd8_7ac1_66b4	7576	15	15	20
Serpent_S5^{-1}	8f2941deb6537ca0	1d6a_5b86_36d2_61cb	5666	15	15	20
Serpent_S6^{-1}	fa1d536049e72c8b	8a3d_9c63_2d59_e60b	7478	14	14	17
Serpent_S7^{-1}	306d9ef85cb7a142	2d59_9c65_4b6c_16f8	7765	16	16	22
GOST_1	4a92d80e6b1c7f53	f614_b38a_7991_2ab6	6686	15	16	23
GOST_2	eb4c6dfa23810759	ea62_23d3_607d_84eb	5577	16	16	21
GOST_3	581da342efc7609b	ca2d_9bb0_1f49_c71a	8776	16	18	19
GOST_4	7da1089fe46cb253	d0cb_b585_4f83_19e6	7554	15	16	20
GOST_5	6c715fd84a9e03b2	647c_ea25_0977_4ee2	7677	17	19	21
GOST_6	4ba0721d36859cfe	59d2_c336_ea91_f486	6577	15	17	19
GOST_7	db413f590ae7682c	08fb_5e32_9c65_a6a3	5775	14	16	17
GOST_8	1fd057a4923e6b8c	2537_3e62_98b6_e946	6676	16	16	21
LBlock_0	e9f0d4ab128376c5	9996_3ac5_f035_44d7	3566	11	11	11
LBlock_1	4be9fd0a7c562813	c53a_9996_0f35_22be	4366	11	11	12
LBlock_2	1e7cfd06b593248a	0f35_9996_22be_c53a	6364	11	11	13
LBlock_3	768b0f3e9acd5241	9969_22eb_5ca3_0fac	4655	11	11	13
LBlock_4	e5f072cd1849ba63	9996_f035_44d7_3ac5	3665	11	11	13
LBlock_5	2dbcfe097a631845	9996_0f35_c53a_22be	3646	11	11	11
LBlock_6	b94e0fad6c573812	5ca3_9969_0fac_22eb	5456	11	11	11
LBlock_7	daf0e49b218375c6	3ac5_9996_f035_44d7	5366	11	11	11
LBlock_8	87e5fd06bc9a2413	c53a_9996_22be_0f35	4366	11	11	12
LBlock_9	b5f0729d481cea36	44d7_f035_9996_3ac5	6635	11	11	12
SC2000_4	25ac7f1bd609483e	49f2_c2b5_933a_a9ac	6755	15	16	19
MIBS	4f38dac0b57e2619	c716_3d26_2e53_897a	7786	17	17	20
KLEIN	74a91fb0c3268ed5	c279_2e65_e923_716c	7777	17	17	21
Panda	0132fc9ba6875ed4	58d6_2b9c_fa30_65f0	7655	14	15	19
MANTIS	cad3ebf789150246	0eec_a0fa_c8d5_0377	6465	13	14	19
GIFT	1a4c6f392db7508e	1ee1_8d72_9a3c_c6aa	4454	11	11	15
UDCIKMP11	086d5f7c4e2391ba	7878_ce64_03fc_d2aa	2424	8	8	8
Luffa_v1	7dbac4835f60912e	3387_c68d_8733_925e	5666	13	13	14
Enocoro_S4	139a5e72d0cf486b	8957_c8ea_5d70_ad2c	7567	16	17	22
Qarma_sigma0	0e2a9f8b6437dc15	dcb0_0dae_bb22_30fa	6646	14	14	20
Qarma_sigma1	ade6f735980cb124	31f2_507d_88be_1b17	6656	15	15	19
Qarma_sigma2	b68fc09e3745d21a	5b49_a38b_1e9a_90dd	7757	16	17	21
Midori_Sb0	cad3ebf789150246	0eec_a0fa_c8d5_0377	6465	13	14	18
Midori_Sb1	1053e2f7da9bc846	0dcd_8af8_d1d4_3f50	6556	15	17	20
Anubis_S0	d7329ac1f45e60b8	4597_592e_1f43_c971	7678	15	16	19
Anubis_S1	4afc0d9be6173582	3ce4_9b86_2b2d_41ee	6764	15	15	18
Khazad_P	3fe054bcda967821	9553_5a47_19b6_27c6	6666	15	17	20
Khazad_Q	9e56a23cf04d7b18	7945_317a_1d8e_a993	6766	16	17	22
Fox_S1	2519eac8647fdb03	bc0e_ad31_1f52_38f8	7875	16	17	21
Fox_S2	b41f03eda875c296	4cad_a569_9cca_53c9	7467	13	13	16
Fox_S3	dab14389572cf06e	13ad_d626_db11_98c7	7577	16	18	19
Whirlpool_E	1b9cd6f3e874a250	44d7_35e2_4d78_135e	6777	16	18	20
Whirlpool_R	7cbde49f638a2510	62cd_1b95_21bb_0cde	6866	15	16	18

Продовження табл. 7

1	2	3	4	5	6	7
SMASH_256_S1	6dc7f13a8b5024e9	867a_52d9_641f_c396	6883	14	14	19
SMASH_256_S2	1b60ed5ac29738f4	5c63_5a96_c974_65b2	6476	13	13	19
SMASH_256_S3	429c81e7f50b6a3d	cba4_79c2_93c9_a95c	7665	15	15	21
CS_cipher_G	a602be18d453fc79	dd50_583b_7722_b1b1	5834	13	14	16
GOST2_1	6af43850de712bc9	ad54_3617_474d_e326	6667	16	17	22
GOST2_2	e0817a56d2493fcb	b958_b2b1_65d1_e925	7666	16	17	20
Magma_1	c462a5b9e8d703f1	ece0_695c_4d27_47d1	4676	16	17	23
Magma_2	68239a5c1e47bd0f	b958_9a2d_aec1_b2b2	7774	16	17	20
Magma_3	b3582fade174c960	26a7_4573_5da4_31e9	7767	16	16	24
Magma_4	c821d4f670a53e9b	d958_b5c4_29f1_e453	7788	15	15	22
Magma_5	7f5a816d093eb42c	16a7_5c4b_a8c7_9a9a	8773	15	16	19
Magma_6	5df692cab78143e0	2b17_63ac_524f_45d6	6576	15	17	20
Magma_7	8e25691cf4b0da37	d568_e516_939a_35a3	7656	16	17	21
Magma_8	17ed05834fa69cb2	52ab_ce86_2b2e_764c	7666	16	17	20
CLEFIA_SS0	e6ca872fb14059d3	f3a0_81eb_54a7_619d	5767	15	17	20
CLEFIA_SS1	640d2ba39cef8751	e9a8_2cf1_6e0b_1f68	6675	15	16	19
CLEFIA_SS2	b85ea64cf72310d9	db05_0f39_43ec_c19b	6557	16	16	20
CLEFIA_SS3	a26d345e0789bfc1	ba58_3297_62ec_7c89	6667	15	16	21
Golden_S0	035869c7dae41fb2	71a6_e692_2dd4_6768	6766	15	15	18
Golden_S1	03586cb79eadf214	59c6_36d2_9ab4_1f68	6665	15	15	20
Golden_S2	03586af4ed9217cb	b646_a972_63d4_c768	5766	15	16	21
Golden_S3	03586cb7a49ef12d	b4c6_59d2_9ab4_9d68	6666	14	16	20
Twofish_Q0_T0	817d6f320b59eca4	0e6e_52f4_b43c_7a29	6648	15	16	21
Twofish_Q0_T1	ecb81235f4a6709d	d1d4_1d65_9b83_c50f	5676	16	17	19
Twofish_Q0_T2	ba5e6d90c8f32471	cc65_5c1b_653c_076b	5767	15	16	19
Twofish_Q0_T3	d7f4126e9b3085ca	2717_86e6_60cf_d385	6658	14	15	21
Twofish_Q1_T0	28bdf76e31940ac5	873c_21f5_c8f8_649e	5656	16	17	19
Twofish_Q1_T1	1e2b4c376da5f908	3ac9_15ce_1bb2_b62a	6766	15	16	18
Twofish_Q1_T2	4c75169a0ed82b3f	e45c_f2a4_862f_aec2	7576	15	17	22
Twofish_Q1_T3	b951c3de647f208a	0c6f_9da1_0fd4_c8d3	6757	16	17	19
Serpent_type_S0	03567abcd4e9812f	a956_c47a_879c_9de0	3767	13	13	15
Serpent_type_S1	035869a7bce21fd4	71a6_2dd2_e694_6768	6476	12	12	19
Serpent_type_S2	035869b2d4e1af7c	6966_74d2_e714_b568	4666	12	13	17
Serpent_type_S3	03586af4ed9217cb	b646_a972_63d4_c768	5766	15	16	21
Serpent_type_S4	03586cb79eadf214	59c6_36d2_9ab4_1f68	6665	15	15	21
Serpent_type_S5	03586cb7a49ef12d	b4c6_59d2_9ab4_9d68	6666	14	16	19
Serpent_type_S6	03586cb7ad9ef124	36c6_59d2_9ab4_1f68	4665	12	12	15
Serpent_type_S7	03586cb7dae41f29	b1c6_66d2_2db4_a768	6656	13	13	18
Serpent_type_S8	03586cf1a49edb27	b4c6_e952_9a74_3d68	6656	14	15	18
Serpent_type_S9	03586cf2e9b7da41	9e46_2dd2_5974_3768	7455	13	13	15
Serpent_type_S10	03586df19c2ba74e	29e6_bc52_e274_9b68	6656	13	14	16
Serpent_type_S11	03586df274eba19c	6966_1dd2_8774_dc68	4666	12	13	17
Serpent_type_S12	03586df2c9a4be17	d266_b4d2_a974_3768	5455	12	13	19
Serpent_type_S13	03586fa179e4bcd2	53a6_9572_6d34_7668	6766	14	15	18
Serpent_type_S14	0358749ef62badc1	a956_1f92_63b4_79c8	3767	13	13	19
Serpent_type_S15	035879beadf4c261	8676_65d2_5e94_17e8	6675	14	14	20
Serpent_type_S16	03589ce7adf46b12	6696_b5c2_1ee4_2778	4665	12	14	15
Serpent_type_S17	0358ad94f621cb7e	6966_e712_d3a4_b178	4665	12	13	15
Serpent_type_S18	0358bc6fe9274ad1	ca96_2dd2_59e4_63b8	6477	13	13	21

Закінчення табл. 7

1	2	3	4	5	6	7
Serpent_type_S19	035a7cb6d429e18f	a956_94da_93b4_d968	3767	13	13	15
BLAKE_1	ea489fd61c02b753	f170_b8a3_62e5_127b	6786	16	17	22
BLAKE_2	b8c052fdae367194	74d1_1f61_9ad4_43c7	5876	15	16	20
BLAKE_3	7931dcbe265a40f8	445f_4bc5_56b1_c8f2	6886	16	17	23
BLAKE_4	905724afe1bc683d	c68d_55d8_99ac_adc1	6558	15	16	21
BLAKE_5	2c6a0b834d75fe19	dea0_34ad_3f06_b26a	6766	16	16	20
BLAKE_6	c51fed4a0763928b	9a2e_ae98_067b_d0b9	5577	15	16	20
BLAKE_7	db7ec13950f4862a	05e7_e44e_2d1d_949b	7456	15	15	17
BLAKE_8	6fe9b308c2d714a5	9c3a_4a37_ad07_459e	6857	14	16	19
BLAKE_9	a2847615fb9e3cd0	57d0_1b33_69b8_6f05	6566	15	17	19
GOST_IETF_1	96328b17a4efc0d5	c8e5_0dae_de82_5d31	7667	16	17	24
GOST_IETF_2	37e98af0526cb4d1	d14b_1667_6d46_587c	7877	15	15	21
GOST_IETF_3	e462b3d8cf5a0719	e670_2a3d_2747_8bd1	6776	16	17	20
GOST_IETF_4	e7acd13902b4f856	54f2_9647_d81b_349d	6768	15	16	23
GOST_IETF_5	b5198df0e423c7a6	286f_ed41_b362_5179	7767	16	16	22
GOST_IETF_6	3adc120b75948fe6	2795_e1a3_eb0c_748e	8766	13	14	21
GOST_IETF_7	1d297a608c45f3be	781b_f074_9e52_d32a	7557	15	17	23
GOST_IETF_8	baf50ce8623917d4	7c0d_2747_e16c_48e7	7766	15	16	20
Kuznyechik_nu0	253b69ea04f18dc7	ac2e_84dd_e652_74e8	6676	17	18	19
Kuznyechik_nu1	76c90f8145bed23a	56a9_ec23_1b27_9c6c	4553	11	12	12
Kuznyechik_sigma	cd048bae3952f167	b722_d9e0_d48b_12f3	5676	15	16	23
Optimal_S0	012d47f68bc93ea5	9a6a_72e4_a4f8_6f48	3567	13	14	17
Optimal_S1	012d47f68be359ac	3a6a_4ee4_94f8_e748	4465	12	13	15
Optimal_S2	012d47f68be3ac59	ca6a_1ee4_64f8_b748	4673	12	13	16
Optimal_S3	012d47f68c53aeb9	cc6a_78e4_26f8_f348	5664	14	15	18
Optimal_S4	012d47f68c9bae53	cc6a_b8e4_62f8_3f48	5674	14	15	21
Optimal_S5	012d47f68cb9ae35	cc6a_74e4_a2f8_3f48	5654	15	16	18
Optimal_S6	012d47f68cb9ae53	cc6a_b4e4_62f8_3f48	5574	14	15	19
Optimal_S7	012d47f68ceba935	e86a_5ce4_86f8_3f48	5774	14	15	20
Optimal_S8	012d47f68e95ab3c	6c6a_72e4_8af8_b748	6553	12	12	15
Optimal_S9	012d47f68eb359ac	3c6a_4ee4_92f8_e748	6465	14	15	16
Optimal_S10	012d47f68eb5a93c	6c6a_56e4_8af8_b748	6653	15	15	17
Optimal_S11	012d47f68eba59c3	b46a_8ee4_52f8_6f48	6647	15	16	20
Optimal_S12	012d47f68eba93c5	b46a_2ee4_c2f8_5f48	6766	15	16	20
Optimal_S13	012d47f68ec95ba3	b86a_e2e4_16f8_6f48	5657	15	15	20
Optimal_S14	012d47f68ecb395a	786a_9ae4_46f8_af48	6766	15	15	22
Optimal_S15	012d47f68ecb93a5	b86a_6ae4_86f8_5f48	5776	15	16	22
Num1_DL_04_0	0bc5619a3ef8d427	956a_c792_b61c_1ec6	3556	12	14	14
Num1_DL_04_1	0cda5be7f6213894	59b4_17e8_83d6_616e	6556	12	13	15
Num1_DL_13_0	0c9761f23b4ed8a5	936c_4bd8_9c5a_7a46	3656	12	12	15
Num1_DL_13_1	0c97f2613b4ea5d8	639c_1b78_6c5a_da16	4656	12	12	16
Num1_DL_13_2	0b85fc36e47921da	6c5a_95d2_47b8_c936	5643	12	12	15
Num1_DL_13_3	0d4b7e926a3581fc	6c5a_47b8_c936_d26a	5436	11	12	15
Num1_DL_22_0	0d82eb75f63c419a	65e2_8778_1bd2_c936	6363	11	11	13
Num1_DL_22_1	0be1a7d46c9f5832	5c6a_c936_1be4_2e56	6346	11	11	14
Num1_DL_22_2	0b69c53ed7842af1	c36a_72c6_4bb4_659a	5644	11	11	16
Num1_DL_22_3	0e95f8a73b6c41d2	639c_87d2_5c9a_4a76	4466	11	11	13
mCrypton_S0	4f38dac0b57e2619	c716_3d26_2e53_897a	7786	17	17	20
mCrypton_S1	1c7a6d53fb20849e	43e5_879c_a176_d32a	8677	16	18	21
mCrypton_S2	7ec209da3f5864b1	c761_538b_3647_4ae6	7877	16	17	20
mCrypton_S3	b0a7d642ce3915f8	7c19_46ad_6378_cb15	8867	17	18	21
				Σ=3190	Σ=3349	Σ=4165

Загалом, як свідчать результати в табл. 7, розроблений нами метод продемонстрував найкращі результати порівнюючи з усіма іншими. Якщо порівнювати наш метод із найближчим конкурентом, представленим утилітами LIGHTER та PEIGEN, то для 129 S-Boxes з 225 (57,3 %) він забезпечує bitsliced-опис із меншою кількістю вентилів. Сумарна кількість вентилів для представлення всіх 225 S-Boxes у нашому методі 3190, що на 4,7 % менше порівнюючи з 3349 вентилями для алгоритмів LIGHTER/PEIGEN. Утиліта sboxgates значно поступається запропонованому методу – лише для 8 S-Boxes (3,5 %) вона змогла згенерувати bitsliced-опис з однаковим із нашим алгоритмом значенням BGC. Варто також зазначити, що розроблений метод для ‘нескладних’ S-Boxes генерує мінімально можливий опис, на що вказують однакові результати з отриманими за допомогою SAT-Solvers.

Висновки

У роботі представлено метод та утиліта для генерації bitsliced-опису довільних 4×4 бієктивних S-Boxes, орієнтовані на програмні реалізації на будь-яких 8/16/32/64-бітних процесорах, що підтримують інструкції AND, OR, XOR, NOT. На сьогодні запропонований у статті метод є найефективнішим із відомих нам методів за критерієм BGC, що підтверджують наведенні в роботі результати досліджень. Метод поєднує в собі евристичні техніки на різних етапах пошуку bitsliced-представлення, зокрема: передобчислення, вичерпний пошук на глибину до 4-х вентилів, IDDFS-алгоритм для пошуку і відсікання варіантів, уточнюючий пошук. За потреби розроблений метод може бути адаптований для підтримки додаткових логічних вентилів.

Список літератури

1. Biham E. «A fast new DES implementation in software», in International Workshop on Fast Software Encryption, 1997. Pp. 260–272. DOI: https://doi.org/10.1007/BFb0052352.

2. Kasper E. and Schwabe P. «Faster and timing-attack resistant AES-GCM», in Proc. 11th International Workshop Cryptographic Hardware and Embedded Systems, 2009. Pp. 1–17. DOI: https://doi.org/10.1007/978-3-642-04138-9_1.

3. Adomnicai A. and Peyrin T. «Fixslicing AES-like ciphers: New bitsliced AES speed records on ARM-Cortex M and RISC-V», IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(1). Pp. 402–425. DOI: https://doi.org/10.46586/tches.v2021.i1.402-425.

4. Schwabe P. and Stoffelen K. «All the AES you need on Cortex-M3 and M4», in International Conference on Selected Areas in Cryptography, 2016. Pp. 180–194. DOI: https://doi.org/10.1007/978-3-319-69453-5_10.

5. Zhang J., Ma M., and Wang P. «Fast implementation for SM4 cipher algorithm based on bit-slice technology», in International Conference on Smart Computing and Communication, 2018. Pp. 104–113. DOI: https://doi.org/10.1007/978-3-030-05755-8_11.

6. Nishikawa N., Amano H., and Iwai K., «Implementation of bitsliced AES encryption on CUDA-enabled GPU», in International Conference on Network and System Security, 2017. Pp. 273–287. DOI: https://doi.org/10.1007/978-3-319-64701-2_20.

7. Matsuda S.and Moriai S. «Lightweight cryptography for the cloud: exploit the power of bitslice implementation», in International Workshop on Cryptographic Hardware and Embedded Systems, 2012. Pp. 408–425. DOI: https://doi.org/10.1007/978-3-642-33027-8_24.

8. Kwan M. «Reducing the Gate Count of Bitslice DES”, IACR Cryptology ePrint Archive, 2000 (51). URL: http://fgrieu.free.fr/Mattew %20Kwan %20- %20Reducing %20the %20Gate %20Count %20of %20Bitslice %20DES.pdf [accessed: 24 October 2022].

9. Dansarie M. «sboxgates: A program for finding low gate count implementations of S-boxes”, Journal of Open Source Software, 6(62), 2021. Pp. 1–3. DOI: https://doi.org/10.21105/joss.02946.

10. Stoffelen K. «Optimizing S-Box Implementations for Several Criteria Using SAT Solvers», in Proc. 23rd International Conference on Fast Software Encryption, 2016. Pp. 140–160. DOI: https://doi.org/10.1007/978-3-662-52993-5_8.

11. Courtois N., Mourouzis T. and Hulme D. «Exact logic minimization and multiplicative complexity of concrete algebraic and cryptographic circuits", International Journal On Advances in Intelligent Systemsю Vol. 6. No. 3 and 4. Pp. 165–176, 2013.

12. Jean J., Peyrin T.,. Sim S, Tourteaux J. «Optimizing Implementations of Lightweight Building Blocks", IACR Transactions on Symmetric Cryptology, 2017(4), 130–168. DOI: https://doi.org/10.13154/tosc.v2017.i4.130-168.

13. Bao Z., Guo J., Ling S. and Sasaki Y. «Peigen – a platform for evaluation, implementation, and generation of S-boxes», IACR Transactions on Symmetric Cryptology. Pp. 330–394, 2019. DOI: https://doi.org/10.13154/tosc.v2019.i1.330-394.

14. Ya. Sovyn, «Bitsliced sbox», 2022. [Online]. URL: https://drive.google.com/drive/folders/1Ae5lXvzhBcVAEq3VaB8lZvFL-gy_B4ZH?usp=sharing [accessed: 24 October 2022].

METHOD AND UTILITY FOR MINIMIZING BITSLICED REPRESENTATIONS OF 4×4 S-BOXES

Ya. Sovyn, V. Khoma, I. Opirskyy

Lviv Polytechnic National University,
Department Information Security

The article is devoted to methods and tools for generating bitsliced descriptions of bijective 4×4 S-Boxes with a reduced number of gates/instructions. Bitsliced descriptions generated by the proposed method make it possible to improve the security and performance of both software implementations of cryptoalgorithms using 4×4 S-Boxes on various processor architectures, as well as FPGA and ASIC-based hardware.

The paper develops a heuristic method of minimization that uses standard logical instructions AND, OR, XOR, NOT, which are available in most 8/16/32/64-bit processors. Due to the combination of different heuristic techniques (preliminary calculations, exhaustive search to a certain depth, DFS algorithm, refining search) in the method, it was possible to reduce the number of gates in bitsliced descriptions of S-Boxes compared to other known methods. The corresponding software in the form of a utility in the Python language was developed and its operation was tested on 225 S-Boxes of various cryptoalgorithms. It is found that the developed method generates a bitsliced description with a smaller number of gates in 57 % of cases compared to the best known methods implemented in the LIGHTER/Peigen utilities.

Keywords: bitslicing, 4×4 S-Box, CPU, logical minimization, software implementation, speed.