Коломицев Михайло Володимирович, кандидат технічних наук, доцент Фізико-технічного інституту КПІ ім. Ігоря Сікорського.
E-mail: box144.85@gmail.com.
Orcid ID: 0000-0001-8460-3041.
Коломыцев Михаил Владимирович, кандидат технических наук, доцент Физико-технического института КПИ им. Игоря Сикорского.
Kolomytsev Myhailo, candidate of technical sciences, associate professor of Institute of Physics and Technologies of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

Носок Світлана Олександрівна, кандидат технічних наук, доцент Фізико-технічного інституту КПІ ім. Ігоря Сікорського.
E-mail: nos.sv.ol@gmail.com.
Orcid ID: 0000-0002-0016-9346.
Носок Светлана Александровна, кандидат технических наук, доцент Физико-технического института КПИ им. Игоря Сикорского.
Nosok Svitlana, candidate of technical sciences, associate professor of Institute of Physics and Technologies of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

DOI: 10.18372/2410-7840.23.16407
УДК 004.056:061.68

ЕВРИСТИЧНИЙ МЕТОД ЗНАХОДЖЕННЯ BITSLICED-ОПИСУ ДОВІЛЬНИХ КРИПТОГРАФІЧНИХ S-Box

Ярослав Совин, Іван Опірський, Дмитро Євенко

Bitsliced-підхід до імплементації блокових шифрів поєднує такі переваги як потенційно високу швидкодію, безпеку і невимогливість до обчислювальних ресурсів. Головною проблемою при переході до bitsliced-опису шифру є представлення S-Box мінімальною кількістю логічних операцій. Відомі методи мінімізації логічного опису S-Box мають низку обмежень, наприклад, працюють лише з S-Box невеликих розмірів, є повільними або неефективними, що загалом стримує використання bitsliced-підходу. У роботі запропоновано новий евристичний метод bitsliced-опису довільних криптографічних S-Box та здійснено порівняння його ефективності з існуючими методами на прикладі S-Box шифру DES. Запропонований метод орієнтований на програмну реалізацію в логічному базисі AND, OR, XOR, NOT, що допускає імплементацію з використанням стандартних логічних інструкцій на будь-яких 8/16/32/64-бітних процесорах. Метод використовує низку евристичних технік, таких як, швидкі алгоритми вичерпного пошуку на невелику глибину, гнучку процедуру планування процесу пошуку, пошук в глибину тощо, що в комплексі забезпечують високу ефективність і швидкодію. Це дає змогу адаптувати його для мінімізації 8×8 S-Box, що на сьогодні є дуже актуальним для багатьох блокових шифрів, зокрема вітчизняного шифру «Калина». Запропонований підхід до bitsliced-опису довільних S-Box усуває обмеження відомих методів такого подання, що стримували використання bitcliced-підходу при удосконаленні програмних реалізацій блокових шифрів для широкого кола процесорних архітектур.

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

ВСТУП

Важливою характеристикою блокових симетричних шифрів (БСШ) є швидкодія, яка у багатьох випадках визначає швидкодію аплікації чи сервісу.

З огляду на різноманіття застосувань БСШ мусить забезпечувати достатньо високу продуктивність для широкого класу мікропроцесорних архітектур із різними обчислювальними можливостями й доступними ресурсами.

Не менш важливою для програмної реалізації БСШ є підвищена стійкість до side-channel атак: 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 найвищою швидкодією володіє bitsliced-підхід, а крім того він забезпечує constant-time імплементацію блокових шифрів з імунітетом до часових та кеш атак [9], є невимогливим до ресурсів, максимально використовує можливості high-end мікропроцесорів щодо збільшення швидкодії внаслідок розпаралелювання як виконання коду (суперскалярність), так і

обробки даних (SIMD-технологія), а також допускає адаптацію для low-end CPU і апаратну реалізацію на FPGA і ASIC.

У bitsliced-підході всі стандартні операції шифру на рівні байтів чи слів, такі як заміна, зсуви, додавання чи множення за модулем тощо, описуються у термінах логічних операцій – наприклад, AND, XOR, OR, NOT. У процесорах кожну таку логічну операцію можна представити відповідною інструкцією (зокрема, і SIMD), 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 простішому виконанню деяких операцій, наприклад, бітових перестановок.

Чим більша розрядність bitsliced-регістрів, тим більший виграш у швидкодії, тому особливо ефективний цей підхід для векторних інструкцій, які оперують багатобітними регістрами. Також чим менше логічних операцій потрібно для bitsliced-опису шифру, тим більша буде швидкодія, тому постає задача мінімізації такого логічного опису. Найбільші складнощі під час переходу до bitsliced-опису шифру виникають з представленням за допомогою мінімальної кількості логічних операцій таблиць нелінійної заміни (S-Box) з розмірами ≥ 6 біт, особливо у випадку, якщо вони згенеровані випадковим чином і не описуються математичними перетвореннями. Саме такому випадку і присвячена ця стаття. На прикладі S-Box шифру DES (6×4), для яких відомі декілька підходів до bitsliced-представлення, нами пропонується власний евристичний метод мінімізації, який можна адаптувати до знаходження bitsliced-представлення найпоширеніших у БСШ S-Box розмірами 8×8 і при цьому забезпечити високу ефективність логічного опису. Таким чином, S-Box шифру DES вибрані в якості полігону для відпрацювання евристичних технік алгоритму та оцінки йогоефективності у порівнянні з відомими методами.

АНАЛІЗ ОСТАННІХ ДОСЛІДЖЕНЬ І ПУБЛІКАЦІЙ

Найважчий етап в bitsliced імплементації БСШ, що значною мірою визначає швидкодію загалом, є логічне подання таблиць нелінійної заміни S-Box.

У випадку апаратної bitsliced імплементації в якості логічного базису виступають логічні вентилі (Gate Equivalent, GE), у програмній імплементації вентилі заміняються відповідними інструкціями, які присутні в більшості процесорних архітектур.

Тому надалі в роботі поняття вентиль та інструкція будемо використовувати як синоніми. Треба зазначити, що в 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 представлення до програмного.

Проаналізуємо підходи до представлення S-Box у вигляді комбінаційної логічної схеми з мінімальною кількістю двовходових вентилів.

S-Box можуть генеруватися двома способами: на підставі певних математичних перетворень чи випадковим чином. Залежно від того, яким чином було утворено S-Box відрізняються і підходи до їх bitsliced-опису.

S-Box з закладеною алгебраїчною структурою, як, наприклад, у шифрах AES чи SM4 допускають компактну bitsliced імплементацію внаслідок використання властивостей закладених перетворень.

При цьому S-Box розбивається на лінійну і нелінійну частини, які мінімізуються евристичними методами [5, 11, 13].

Але оскільки наша мета розробити методи мінімізації довільних S-Box цей підхід нам не підходить.

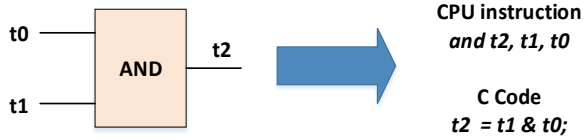

Рис. 1 Перехід від логічного до програмного bitsliced представлення

Неалгебраїчні S-Box, тобто S-Box згенеровані випадковим чином, не описуються будь-якими аналітичними залежностями, що ускладнює їхнє bitsliced представлення, але одночасно робить їх більш стійкими до диференційного, лінійного та алгебраїчного криптоаналізу. Прикладами таких таблиць є S-Box шифрів «Kalyna», «Kuznyechik», KHAZAD і Anubis.

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

Світовим стандартом де-факто для мінімізації табличних функцій є програма Espresso, що використовує евристичний алгоритм [4]. Алгоритм Espresso використовується в 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 лише операцій AND, OR, NOT (операція XOR може бути впроваджена вже над результатом мінімізації, але не в процесі) та не враховується обмеження на число входів вентилів. Внаслідок цього отриманий опис є далеко не оптимальний.

Одна з перших утиліт пошуку bitsliced-представлення була розроблена для S-Box шифру Serpent [7], одного з фіналістів конкурсу AES. Ця публічно доступна утиліта написана мовою C і здійснює пошук логічного представлення 4×4 S-Box використовуючи типовий для CPU набір інструкцій {XOR, AND, OR, NOT}. Сам алгоритм базується на пошуку depth-first-search (DFS) у поєднанні з евристичними техніками і забезпечує хоч і не оптимальні, проте достатньо хороші результати.

У роботі [12] запропоновано евристичний алгоритм для пошуку bitsliced-представлення 4×4 S-Box. Особливістю алгоритму є орієнтація на програмну реалізацію на процесорах з x86-архітектурою, у яких є відносно невелика кількість регістрів загального призначення (8 на момент публікації). Це накладає обмеження на кількість тимчасових змінних, які можуть одночасно використовуватися, і тому алгоритм здійснює пошук таких послідовностей логічних інструкцій, які б у кожен момент часу потребували не більше 5 регістрів. Проте це обмеження приводить до появи надлишкових інструкцій. На сьогодні x86-64 CPU містять 16 регістрів загального призначення та 16 (SSE, AVX) чи 32 (AVX-512) векторних регістрів, тому для них проблема нестачі регістрів вже не так актуальна. Те саме стосується і більшості сучасних RISC-процесорів з 16 або 32 регістрами.

Як показано в роботі [15], для мінімізації невеликих S-Box можна застосовувати програми SAT-solvers. Перевагою такого підходу є те, що він забезпечує гарантовано оптимальний розв'язок (якщо буде знайдений). Проблемою є те, що такий підхід працює лише для невеликих S-Box, розміром до 5×5, хоча вже і тут виникають складнощі й не завжди вдається знайти рішення. Для більших S-Box цей підхід неможливо реалізувати з погляду обчислювальної складності. Можна розбивати велику таблицю на менші частини та мінімізувати за допомогою SAT-solvers кожну частину окремо, проте результат буде далеко не оптимальним, бо не враховуватиме залежності між ними.

У роботі [8] описано утиліту LIGHTER, що здатна генерувати ефективні bitsliced-імплементації 4×4-біт S-Box для заданих набору вентилів та їх вагових коефіцієнтів. 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. Сам алгоритми пошуку LIGHTER комбінує два підходи: пошук з 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 вперед, а інший стартує з шуканих векторів і здійснює пошук на-

зад. 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 [2] описано утиліту Peigen, що дає змогу знаходити bitsliced-опис S-Box у різних логічних базисах, застосовуючи задані критерії мінімізації для апаратних і програмних імплементацій.

Алгоритми пошуку bitsliced-опису утиліти Peigen спираються на алгоритми з утиліти LIGHTER [8], але покращено їх ефективність, зокрема використано передобчислення та ряд додаткових технік. Проте навіть з внесеними удосконаленнями утиліта не здатна знаходити оптимальне bitsliced-представлення для S-Box з розрядністю 5 і більше біт, і ефективно працює лише з 4-бітними S-Box.

Таким чином, розглянута група алгоритмів та утиліт здебільшого придатна для роботи з S-Box з розрядністю не більше 4 біт ($\leq 4 \times 4$), і вже навіть для DES S-Box (6×4) є проблема їх масштабування, не кажучи про 8-бітні S-Box.

E. Biham у роботі [3] описав алгоритм представлення DES S-Box (6×4) мінімальною кількістю стандартних логічних вентилів XOR, AND, OR, NOT (у середньому 100 вентилів на S-Box). У алгоритмі з шести вхідних змінних *x0-x5* дві вхідні змінні формують всі можливі комбінації з заданих логічних вентилів, а решта чотири виступають селекторами, які вибирають одну з можливих комбінацій.

Селекція здійснюється ієрархічно, при цьому вихідний S-Box розглядається як логічна функція шести змінних. Спочатку здійснюється перехід до двох логічних функцій п'яти змінних, а шоста змінна вибирає одну з функцій. Далі для функцій п'яти змінних одна змінна вибирається як селектор серед двох функцій чотирьох змінних і так далі, доки не дійдемо до функцій двох змінних.

Дещо кращі результати можна отримати, якщо використовувати різний порядок вхідних біт у представленні DES S-Box [10]. У роботі [3] для побудови всіх функцій двох змінних брались четвертий і п'ятий вхідні біти, а решта вхідних біт використовувалися як селектори завжди в однаковому порядку.

Проте є 6! (720) можливих перестановок порядку вхідних біт, тому перебравши для кожного S-Box всі ці перестановки можна знайти bitsliced-опис з дещо меншим числом вентилів. Завдяки цій техніці вдалося зменшити середню кількість стандартних вентилів із 100 до 88 на S-Box.

У роботі [10] запропонований новий підхід до знаходження bitsliced-імплементації DES S-Box. У ньому кожен вихідний біт S-Box розглядається як функція шести вхідних біт, що представлена картою Карно і розташовується у 64-бітній змінній. Всі вхідні та проміжні змінні теж можна розглядати як 6-бітні карти Карно, що описуються 64-бітними числами.

Тоді задача формулюється так: потрібно скомбінувати існуючі вхідні та проміжні карти таким чином, щоб отримати шукану вихідну змінну. Залежно від того в якому порядку буде здійснюватися пошук доступні 6! варіантів пошуку для вхідних змінних та 4! варіантів пошуку для вихідних змінних.

Це сумарно дає 17280 варіантів серед яких вибирається варіант з мінімальним числом вентилів. На кожному кроці пошук здійснюється рекурсивно, шляхом перебору комбінацій з одного, двох чи трьох вентилів.

Використовуючи різні набори логічних вентилів були отримані результати представлені у таблиці 1.

У середньому, з використанням стандартних вентилів, один S-Box потребує 56 GE, а всі вісім S-Box сумарно – 448 GE, для нестандартних вентилів потрібно 51 та 408 GE відповідно. Знайдені bitsliced-описи були використані у популярній програмі для зламу паролів John the Ripper.

Алгоритм M. Kwan з деякими удосконаленнями реалізований у вигляді утиліти sboxgates [6]. Ця утиліта дає змогу задавати набір вентилів з 16 можливих, додавати власні S-Box, використовува-

ти LUT-подібні 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.

R. Rusakov та A. Peslyak вдалося суттєво покращити результати M. Kwan використовуючи нестандартний набір вентилів {XOR, AND, OR, NOT, AND-NOT}. Вибраний набір інструкцій підтримується x86-64 CPU (та різними RISC-CPU), а кожна з операцій може бути представлена як звичайною інструкцією, що оперує регістрами загального призначення, так і векторною інструкцією, де операндами виступають SIMD-регістри з розширень SSE/AVX2/AVX-512. Знайдені bitsliced-описи були додані в програму John the Ripper замість описів M. Kwan.

На жаль, детальний опис використаного алгоритму у вигляді статті відсутній, як відсутній і сам код алгоритму у публічному доступі. Тому ми орієнтуємось на ту інформацію, яка доступна онлайн [14]. Головною ідеєю є сформувати 64-бітне значення з двох 32-бітних, як в алгоритмах E. Biham та M. Kwan, проте самі 32-бітні значення отримати використовуючи передобчислену таблицю розміром 2^32, що для кожного 32-бітного вектору містить мінімальну кількість вентилів, якою цей вектор описується.

Для знаходження мінімальної кількості операцій для кожного 32-бітного вектору використовується BFS алгоритм.

Таблиця 1

Bitsliced-опис DES S-Box алгоритмом M. Kwan [10]

Стандартний набір вентилів (XOR, AND, OR, NOT)									**Нестандартний набір вентилів (XOR, AND, OR, NOT, AND-NOT, XOR-NOT)**							
S-Boxes	*S1*	*S2*	*S3*	*S4*	*S5*	*S6*	*S7*	*S8*	*S1*	*S2*	*S3*	*S4*	*S5*	*S6*	*S7*	*S8*
GE	63	56	57	42	62	57	57	54	56	50	53	39	56	53	51	50

Для формування чотирьох 64-бітних вихідних значень потрібно знайти вісім 32-бітних значень, а це відбувається з допомогою DFS алгоритму з врахуванням їх складності з передобчисленої таблиці.

Отримані R. Rusakov результати представлені у таблиці 2. У середньому один S-Box потребує 44 GE, а всі вісім S-Box сумарно – 353 GE.

Таблиця 2

Bitsliced-опис DES S-Box алгоритмом R. Rusakov [14]

Нестандартний набір вентилів (XOR, AND, OR, NOT, AND-NOT)								
S-Boxes	*S1*	*S2*	*S3*	*S4*	*S5*	*S6*	*S7*	*S8*
GE	49	44	46	33	48	46	46	41

Хоча підхід R. Rusakov і зменшує на 17% середнє число вентилів у порівнянні з алгоритмом M. Kwan, проте він потребував більше трьох місяців обчислень, що само по собі досить багато. У випадку 8×8 S-Box тривалість пошуку ще більше зросте, що робить цей алгоритм непридатним для практичного використання. Алгоритм M. Kwan значно швидший і пошук bitsliced-опису для 6×4 S-Box утилітою *sboxgates* займає хвилини, але потребує багато ітерацій, щоб отримати найменший опис.

ПОСТАНОВКА ЗАВДАННЯ

Отже, розглянуті методи мінімізації або працюють лише з невеликими S-Box, або не можуть бути безпосередньо застосовані до випадково згенерованих S-Box, або не дають задовільних результатів, або не адаптуються до S-Box з розмірами 6 біт і більше. З огляду на це нами у роботі пропонується власний евристичний метод мінімізації довільних S-Box, основними вимоги до якого були:

1) ефективність мінімізації краща ніж в алгоритмі M. Kwan;

2) можливість адаптації алгоритму до ефективної роботи з 8×8 S-Box;

3) використання лише операцій {AND, OR,

XOR, NOT}, що забезпечить програмну імплементацію на будь-яких архітектурах процесорів;

4) прийнятний час роботи алгоритму.

Представлення S-Box

У нашому bitsliced представленні використовуються тільки стандартні логічні вентилі/інструкції {XOR, AND, OR, NOT} і тому цей варіант можна реалізувати на будь-яких 8/16/32/64-бітних процесорах, включаючи найпростіші 8-бітні MCU, а також на x86-64 процесорах з підтримкою векторних команд, де також є потрібні SIMD-інструкції AND, OR, XOR.

Інструкція NOT емулюється операцією XOR, де один з векторних регістрів містить всі 1 (одиничний вектор *inv*).

Усього є вісім DES S-Box з розрядністю 6×4, які будемо позначати *S1-S8*. У bitsliced представленні таблиці заміни можна розглядати як логічні функції 6×4 задані таблицями істинності.

Наприклад, фрагмент *S1* має вигляд показаний у табл. 3, де *x0-x5* вхідні біти, а *y0-y3* – вихідні.

Таблиця 3

Таблиця істинності для S1

x	*x0*	*x1*	*x2*	*x3*	*x4*	*x5*	*y0*	*y1*	*y2*	*y3*	***y=Sbox[x]***
0	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*0*	*0xe*
1	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0x0*
2	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*1*	*0*	*0*	*0x4*
3	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*0xf*
…	…	…	…	…	…	…	…	…	…	…	…
63	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*1*	*0xd*

Вхідні *x0-x5* і вихідні змінні *y0-y3* представляються 64-бітними числами (векторами). Вихідні вектори можна розглядати як логічні функції 6-ти аргументів: *yi = f(x0, x1, x2, x3, x4, x5), i = 0…3*. Вести пошук серед 64-бітних чисел (2^64 значень) є нереалістично з погляду вимог до пам'яті та обчислювальної потужності. Тому 64-бітні змінні *yi* розбиваються на 32-бітні частини, де кожна частина це вже логічна функція 5-х вхідних бітів для знаходження якої можна застосувати вичерпний пошук (exhaustive search), а 6-й біт використовується як селектор для об'єднання частинок.

Перебір всіх 6! (720) можливих S-Box, які можна отримати шляхом перестановки порядку вхідних біт, як це пропонується у методі M. Kwan, потребує надто багато часу і обчислювальних ресурсів, а у випадку переходу до 8×8 S-Box число перестановок зросте до 8! (40320). Тому нами цей підхід не використовується і пошук завжди ведеться для фіксованого порядку вхідних біт, а зменшення числа вентилів досягається за рахунок кращого пошуку на кожному кроці.

У нашому алгоритмі як селектор обрано біт *x0 = 0xffffffff00000000*. 64-бітні значення *yi* розбиваються на молодшу (*yi_l*) та старшу (*yi_h*) 32-бітні половини, які відповідно є функціями 5-х змінних *yi_l/yi_h = f(x1, x2, x3, x4, x5)*. Пошук ведеться серед 32-бітних чисел, а далі дві частинки "склеюються" в 64-бітне значення з допомогою *x0* та операцій з логічного базису одним з можливих способів, як показано на рис. 2. Такий підхід потребує накладних витрат у вигляді 9 вентилів: 8 вентилів на операції "склеювання" і формування 4-х 64-бітних векторів та 1 вентиль для інвертування *x0*.

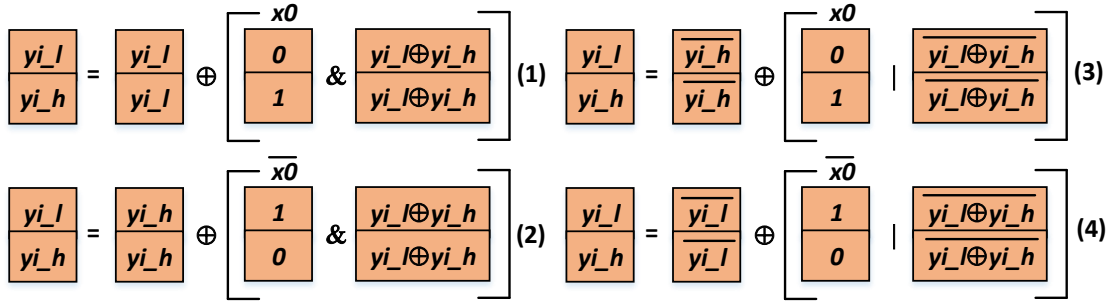

Рис. 2 Способи об'єднання 32-бітних векторів у 64-бітні для стандартного набора вентилів

Отже, один S-Box буде описуватися вісьмома 32-бітним значеннями *y_32 = {y0_l, y0_h, y1_l, y1_h, y2_l, y2_h, y3_l, y3_h}*. У свою чергу кожне 64-бітне значення *y0-y3* перед початком пошуку представлене 6-ма можливими 32-бітними векторами, залежно від того, яке рівняння (рис. 2) буде використано для "склеювання". Позначимо цю множину з 24 значень *sbox_32*: $sbox\_32 = \{yi\_l, yi\_h, \overline{yi\_l}, \overline{yi\_h}, yi\_l \oplus yi\_h, \overline{yi\_l \oplus yi\_h}\}$, $i = 0\ldots3$. Відповідно алгоритм пошуку буде складатися з 8-ми кроків, на кожному з яких знаходиться одне зі значень з *sbox_32*. У процесі пошуку цих значень використовуються вхідні біти та попередньо знайдені вектори з *sbox_32* і проміжні змінні. Наприклад, якщо на першому кроці на мінімальній глибині пошуку було знайдене значення *y0_l*, то на наступному кроці *sbox_32* міститиме 19 значень – по 6 значень для *y1-y3* та значення $y0\_l \oplus y0\_h$. Якщо ж на першому кроці було б знайдене значення $y0\_l \oplus y0\_h$, то на наступному кроці *sbox_32* міститиме 20 значень – по 6 значень для *y1-y3* та значення *yi_l* і *yi_h*.

Таким чином, задачу мінімізації для запропонованого представлення S-Box можна сформулювати так: задано набір із 5-х векторів (32-бітних чисел) *x1-x5: x5 = 0xaaaaaaaa*, *x4 = 0xcccccccc*, *x3 = 0xf0f0f0f0*, *x2 = 0xff00ff00*, *x1 = 0xffff0000*. Також є тривіальні нульовий 0x0 та одиничний вектори *inv = 0xffffffff*. Потрібно використовуючи мінімум логічних операцій (проміжних змінних) обчислити всі 8 векторів *y_32*, що описують конкретний S-Box.

Вектори *x1-x5* формують базу *base = {x1, x2, x3, x4, x5}* з якої починається пошук. У подальшому до бази додаються значення проміжних змінних *t* та знайдених векторів з *sbox_32*. Послідовність проміжних змінних *t* та значень *sbox_32* формують так звану трасу *tr*.

Наприклад, якщо для знаходження значення *y1_h* потрібно виконати три вказані операції, то траса буде така:

Операції:	**Траса:**
t0 = x2 ^ x4	*tr = [t0 t1 y1_h]*
t1 = t0 & x3	
y1_h = t1 | x1	

Після знаходження кожного вектора з *sbox_32* відповідна йому траса заноситься в загальну трасу *tr*, з якої потім формується система логічних рівнянь, а значення траси додаються до бази: *base = {x0, x1, x2, x3, x4, t0, t1, y1_h}*.

Ця нова база тепер буде використовуватися під час пошуку трас для решти значень з *sbox_32*. Кількість вентилів для побудови траси будемо позначати *ge*, наприклад, для даного випадку *ge* = 3.

Евристичні техніки мінімізації

Під час пошуку використовуються такі евристичні техніки:

1. *Позачерговий пошук*. На кожному кроці пошук 32-бітних векторів здійснюється не в межах одного 64-бітного значення *yi = {yi_l, yi_h}*, а для всіх можливих 32-бітних частинок векторів *y0-y3*, представлених у *sbox_32*. Склеювання 32-бітних векторів здійснюється по мірі готовності, коли будуть знайдені дві частини одного 64-бітного вектору. Це дає змогу на кожному кроці вибирати представлення 32-бітного вектору з мінімальною кількістю вентилів. У алгоритмі M. Kwan пошук ведеться попарно для заданого 64-бітного значення *yi*.

2. *Вичерпний пошук на глибину 4*. На кожному кроці здійснюється пошук векторів з набору *sbox_32* шляхом перебору всіх можливих значень, які можна отримати з допомогою 4-х стандартних логічних операцій. Пошук ведеться алгоритмом BFS. Це дає змогу більш точно знаходити 32-бітні вектори на кожному кроці. Для вичерпного пошуку розроблені відповідні швидкі функції позначені *q0-q3* (рис. 3), де функція *q0* здійснює вичерпний пошук на глибину 1 вентиль (*ge* = 1), *q1* – 2 вентилі (*ge* = 2) і т. д.

Якщо на глибині *ge* = 4 не знайдено жодного вектора, то застосовується частковий пошук на глибину 5 як показано на рис. 4. На перших кроках роботи алгоритму може вникнути ситуація, коли навіть пошуку на глибину 5 недостатньо. Тому нами ця проблема вирішується за рахунок

розширення бази таким чином: до бази *base* додаються всі можливі значення t_i, знайдені на глибині 1 ($ge = 1$) з допомогою функції $q0$: $base_{q0} = base \cup q0(base)$. Після чого повторюється пошук вже для бази $base_{q0}$. Для знайдених трас формуються логічні рівняння і відбираються траси з мінімальним сумарним *ge*.

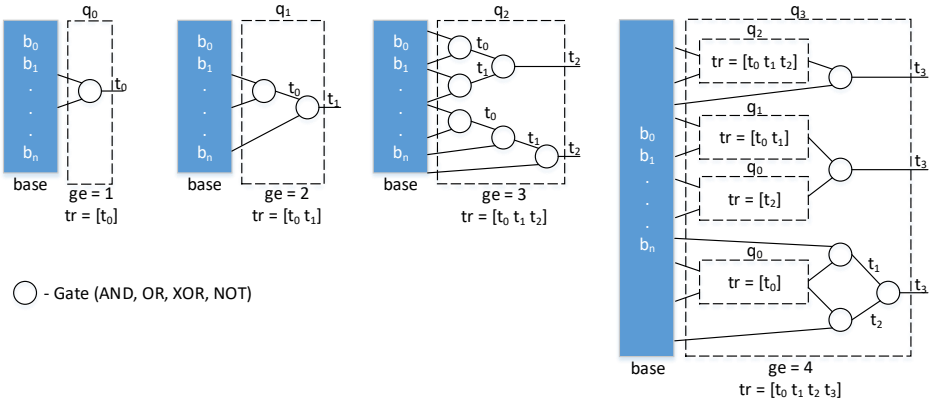

Рис. 3 Функції вичерпного пошуку на глибину до 4-х вентилів q0-q3

3. *Пошук на крок вперед*. На кожному кроці з *sbox_32* відбираються знайдені вектори не тільки з мінімальною кількістю вентилів *min_ge*, але і всі знайдені в процесі пошуку вектори з довжиною *min_ge* + 1. Ці вектори використовуються на наступному кроці пошуку. Сенс у тому, що кількість трас з *min_ge* може бути відносно невелика, тому пошук на наступному кроці може потребувати довших трас і сумарна кількість вентилів виявиться більшою (*min_ge* + 5), ніж при пошуку серед більшої кількості варіантів (*min_ge* + 4) – рис. 5.

4. *Передобчислені таблиці на першому кроці*. На першому кроці коли база *base* є малою (лише 5 значень *x1-x5*), знайти найближче значення з *sbox_32* за допомогою вичерпного пошуку на глибину 4 чи часткового пошуку на глибину 5-7 здебільшого не вдається.

Тому використовуються LUT-таблиці, що містять представлення всіх унікальних векторів на глибину 6 ($ge = 6$). Ці вектори комбінуються між собою логічними операціями для знаходження значень з *sbox_32*.

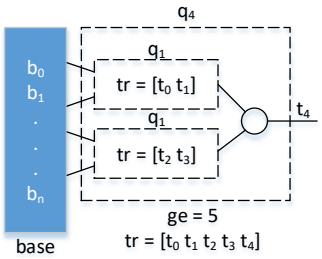

Рис. 4 Функція часткового пошуку на глибину до 5-ти вентилів

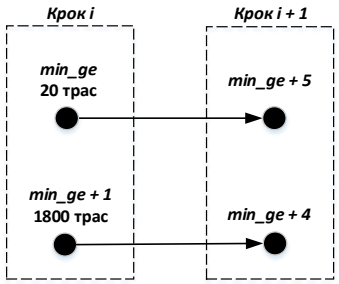

Рис. 5 Пошук на крок вперед

Розроблений метод мінімізації S-Box був імплементований мовою Python. Утиліта евристичної мінімізації здійснює автоматичну генерацію bitsliced-рівнянь, тестування їх коректності та запис результатів у файл. Отримані в роботі bitsliced-описи DES S-Box доступні за посиланням [1].

Результати застосування розробленого алгоритму до різних DES S-Box наведено в табл. 4. У середньому з використанням стандартних вентилів один S-Box потребує 48,5 GE, а всі вісім S-Box сумарно – 388 GE.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ ТА ЇХ ОБГОВОРЕННЯ

Загалом наш метод мінімізації демонструє на 13% кращі результати, ніж метод M. Kwan за умови використання стандартного набору вентилів.

Навіть у порівнянні з результатами методу M. Kwan, отриманими для нестандартного набора вентилів, наш метод потребує на 5% менше операцій, хоча використовує на два типи вентилів

менше.

Відзначимо високу швидкодію розробленого методу – пошук bitsliced-представлення для одного DES S-Box потребує 1-2 доби.

Таблиця 4

Порівняння методів bitsliced-опису DES S-Box

Стандартний набір вентилів (XOR, AND, OR, NOT)								
	S1	*S2*	*S3*	*S4*	*S5*	*S6*	*S7*	*S8*
Наш, GE	52	48	47	41	51	52	49	48
M. Kwan [10], GE	63	56	57	42	62	57	57	54
Нестандартний набір вентилів (XOR, AND, OR, NOT, AND-NOT, XOR-NOT)								
M. Kwan [10], GE	56	50	53	39	56	53	51	50

Метод M. Kwan потребує численних ітерацій для знаходження bitsliced-опису з найменшою кількістю вентилів, що може займати декілька тижнів, а метод R. Rusakov вимагає 3-4 місяці.

ВИСНОВКИ

Таким чином, у роботі було запропоновано ефективний метод знаходження bitsliced-представлення S-Box та продемонстровано його ефективність на прикладі DES S-Box.

Особливістю запропонованого методу полягає в можливості його використання для поширених у БСШ 8×8 S-Box, що дасть змогу збільшити швидкодію і безпеку програмних реалізацій багатьох блокових шифрів для широкого спектра процесорних архітектур.

При цьому, задача ефективного і швидкого пошуку bitsliced-опису була вирішена шляхом застосування евристичних технік, зокрема, швидких алгоритмів вичерпного пошуку на невелику глибину, більш гнучкого планування процесу пошуку, обмеженому використанню LUT-таблиць відносно невеликого розміру лише наприкінці роботи алгоритму.

ЛІТЕРАТУРА

[1] Совин Я. Р., Опірський І. Р., Євенко Д. А. Bitsliced DES S-Boxes, 2021. [Електронний ресурс]. Режим доступу: https://drive.google.com/drive/folders/1GRFTstGBXuwQnknyAfMR4leHWzHrgDc0?usp=sharing.

[2] Bao Z., Guo J., Ling S., Sasaki Y. PEIGEN – a Platform for Evaluation, Implementation, and Generation of S-boxes // IACR Transactions on Symmetric Cryptology. – 2019. – №1. – pp. 330–394.

[3] Biham E. A fast new DES implementation in software // In Proc. of 4th International Workshop Fast Software Encryption (FSE '97). January 20-22, 1997. – Haifa, Israel. – pp. 260-272.

[4] Brayton R., Hachtel G., McMullen C., Sangiovanni-Vincentelli A. Logic minimization algorithms for VLSI synthesis // Kluwer Academic Publishers, Hingham, USA, 1984. – 193 p.

[5] Canright D. A very compact S-Box for AES // In Proc. of 7th International Workshop Cryptographic Hardware and Embedded Systems. August 29 - September 1, 2005. – Edinburgh, UK. – pp. 441-455.

[6] Dansarie M. sboxgates: A program for finding low gate count implementations of S-boxes // Journal of Open Source Software. – 2021. – Vol. 6, №62. – pp. 1-3.

[7] Gladman B. Serpent S Boxes as Boolean Functions. [Електронний ресурс]. Режим доступу: https://panthema.net/2008/0714-cryptography-speedtest-comparison/crypto-speedtest-0.1/src/serpent-gladman.cpp.html.

[8] Jean J., Peyrin T., Sim S., Tourteaux J. Optimizing Implementations of Lightweight Building Blocks" // IACR Transactions on Symmetric Cryptology. – 2017. – №4. – pp. 130–168.

[9] Käsper E., Schwabe P. Faster and Timing-Attack Resistant AES-GCM // In Proc. of 11th International Workshop Cryptographic Hardware and Embedded Systems. September 6-9, 2009. – Lausanne, Switzerland. – pp. 1-17.

[10] Kwan M. Reducing the Gate Count of Bitslice DES // IACR Cryptology ePrint Archive, 2000:51. – 2000.

[11] Maximov A., Ekdahl P. New Circuit Minimization Techniques for Smaller and Faster AES SBoxes // IACR Transactions on Cryptographic Hardware and Embedded Systems. – 2019. – № 4. – pp. 91-125.

[12] Osvik D. Speeding up Serpent // In Proc. of the Third AES Candidate Conference. April 13-14, 2000. – New York, NY, USA. – pp. 317-329.
[13] Reyhani-Masoleh A., Taha M., Ashmawy D. Smashing the implementation records of AES S-Box // IACR Transactions on Cryptographic Hardware and Embedded Systems. – 2018. – № 2. – pp. 298-336.
[14] Rusakov R., Peslyak A. John the Ripper 1.7.8: DES speedup, 2011. [Електронний ресурс]. Режим доступу: https://www.openwall.com /lists/john-users/2011/06/24/4.
[15] Stoffelen K. Optimizing S-Box Implementations for Several Criteria Using SAT Solvers //In Proc. of 23rd International Conference Fast Software Encryption (FSE 2016). March 20-23, 2016. – Bochum, Germany. – pp. 140-160.
[16] Kuznetsov, O, Oliynikov, R, Gorbenko, Yu, Pushkaryov, A, Dirda, O & Gorbenko, I 2014, 'Requirements justification, construction and analysis of perspective symmetric cryptographical transformations on the base of block cipher codes', *Scientific journal "Computer Systems and Networks" Lviv Polytechnic National University*, vol. 806, pp. 124-140. Available from: http:/ /science.lpnu.ua/sites/default/files/journal –pa per/2017/nov/6634/21-124-141.pdf.

ЭВРИСТИЧЕСКИЙ МЕТОД НАХОЖДЕНИЕ BITSLICED-ОПИСАНИЯ ПРОИЗВОЛЬНЫХ КРИПТОГРАФИЧЕСКИХ S-BOX

Bitsliced-подход к имплементации блочных шифров объединяет такие преимущества как потенциально высокое быстродействие, безопасность и нетребовательность к вычислительным ресурсам. Главной проблемой при переходе к bitsliced-описанию шифра является представление S-Box минимальным количеством логических операций. Известные методы минимизации логического описания S-Box имеют ряд ограничений, например, работают только с S-Box небольших размеров, являются медленными или неэффективными, что в целом сдерживает использование bitsliced-подхода. В работе предложен новый эвристический метод bitsliced-описания произвольных криптографических S-Box и проведено сравнение его эффективности с существующими методами на примере S-Box шифра DES. Предложенный метод ориентирован на программную реализацию в логическом базисе AND, OR, XOR, NOT, что допускает имплементацию с использованием стандартных логических инструкций на любых 8/16/32/64-битных процессорах. Метод использует ряд эвристических техник, таких как, быстрые алгоритмы исчерпывающего поиска на небольшую глубину, гибкую процедуру планирования процесса поиска, поиск в глубину и т.п., которые в комплексе обеспечивают высокую эффективность и быстродействие. Это позволяет адаптировать его для минимизации 8×8 S-Box, что сегодня очень актуально для многих блочных шифров, в частности отечественного шифра «Калина». Предложенный подход к bitsliced-описанию произвольных S-Box устраняет ограничения известных методов такого представления, которые сдерживали использование bitcliced-подхода при совершенствовании программных реализаций блочных шифров для широкого круга процессорных архитектур.

Ключевые слова: bitslicing, S-Box, логическая минимизация, x86-64 CPU, программная имплементация, блочные шифры.

HEURISTIC METHOD OF FINDING A BITSLICED DESCRIPTION OF ARBITRARY CRYPTOGRAPHIC S-BOX

Bitsliced approach to the implementation of block ciphers combines advantages such as potentially high speed, security and unpretentiousness to computing resources. The main problem in the transition to the bitsliced-description of the cipher is the representation of the S-Box with a minimum number of logical operations. Known methods of minimizing the logical description of the S-Box have a number of limitations, for example, work only with small S-Box, are slow or inefficient, which generally hinders the use of bitsliced-approach. The paper proposes a new heuristic method of bitsliced-description of arbitrary cryptographic S-Box and compares its efficiency with existing methods on the example of S-Box DES cipher. The proposed method is focused on software implementation in the logical basis AND, OR, XOR, NOT, which allows implementation using standard logical instructions on any 8/16/32/64-bit processors. The method uses a number of heuristic techniques, such as, fast algorithms for exhaustive search at shallow depth, flexible procedure for planning the search process, search in depth, etc., which together provide high efficiency and speed. This allows you to adapt it to minimize the 8×8 S-Box, which is very relevant today for many block ciphers, including the domestic cipher "Kalyna". The proposed approach to the bitsliced-description of arbitrary S-Box eliminates the limitations of the known methods of such representation, which restrained the use of the bitcliced-approach in improving software implementations of block ciphers for a wide range of processor architectures.

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

Совин Ярослав Романович, кандидат технічних наук, доцент, доцент кафедри захисту інформації Національного університету «Львівська політехніка».
E-mail: yaroslav.r.sovyn@lpnu.ua.
Orcid ID: 0000-0002-5023-8442.

Совын Ярослав Романович, кандидат технических наук, доцент, доцент кафедры защиты информации Национального университета «Львовская политехника».

Sovyn Yaroslav, PhD Eng, Associate Professor, Associate Professor at the Department of Information Security, National University "Lviv Polytechnic".

Опірський Іван Романович, д.т.н., професор, професор кафедри захисту інформації Національного університету «Львівська політехніка».
E-mail: ivan.r.opirskyi@lpnu.ua.
Orcid ID: 0000-0002-8461-8996.

Опирский Иван Романович, д.т.н., профессор., профессор кафедры защиты информации Национального университета «Львовская политехника».

Opirskyy Ivan, Doctor of Technical Sciences, Professor, Professor at the Department of Information Security, National University "Lviv Polytechnic".

Євенко Дмитро Андрійович, студент кафедри захисту інформації Національного університету «Львівська політехніка».
E-mail: dmytro.yevenko.kb.2018@lpnu.ua.
Orcid ID: 0000-0002-9975-750X.

Евенко Дмитрий Андреевич, студент кафедры защиты информации Национального университета «Львовская политехника».

Yevenko Dmytro, student at the Department of Information Security, National University "Lviv Polytechnic".