УДК 519.6

ОПТИМІЗАЦІЯ S-БЛОКІВ ДЛЯ МАЛОПОТУЖНИХ ПРИСТРОЇВ

В. С. Геращенко[1]

[1] Навчально-науковий Фізико-технічний інститут

Анотація

У даній роботі розглядаються алгоритми для оптимізації реалізації S-блоків для малопотужних пристроїв та їх можливе практичне застосування для 4-бітових S-блоків, які є кандидатами у нелінійні компоненти шифру ГОСТ. Також розглядається основні критерії, за якими можна оптимізовувати реалізації S-блоків та їх основні представлення для безпечних та ефективних реалізацій.

Ключові слова: малопотужна криптографія, алгоритми оптимізації, S-блок, гейтовий розмір

Вступ

В сучасному світі все більш важливими стають повсякденні обчислення і малопотужні пристрої. З поширенням інтернету речей (IoT) і RFID-чипів у побуті та промисловості, захист інформації на таких пристроях є актуальним. Тому в останні роки малопотужна криптографія стає все більш популярною темою для досліджень.

Дослідження можливості оптимізації булевих функцій та логічних схем є відомою задачею. Одним з перших алгоритмів оптимізації загальних логічних схем є алгоритм ESSPRESSO [1], який було запропоновано у 1982 році. Після того як симетричний шифр AES став стандартом у 2002 році, з'явилось багато досліджень про оптимізацію логічної схеми шифру та його лінійних та нелінійних компонент для використання у малопотужних пристроях [2, 3, 4]. У 2015 році у дослідженні можливих атак на малопотужний шифр MIDORI [5] було запропоновано алгоритм, який генерує 4-бітові булеві функції з метою мінімізації глибини та використання їх у S-блоку шифру MIDORI. У 2016 році було запропоновано перший працюючий метод, який використовує SAT-вирішувач [6]. Згодом, у 2023 році було розглянуто модифікацію цього алгоритму [7]. У 2017 році було реалізовано алгоритм LIGHTER [8], який генерує оптимізовані логічні схеми для лінійних та нелінійних складових симетричних шифрів, який згодом став частиною набору інструментів PEIGEN [9].

У даній роботі буде розглянуто найкращі алгоритми для оптимізації та генерації S-блоків для малопотужних пристроїв та аналіз складності наведентих алгоритмів. Також будуть наведені основні метрики, за якими оптимізовують реалізації S-блоків. Буде наведено знайдені оптимізовані реалізації S-блоків, які є кандидатами для шифру ГОСТ.

1. Основні означення та алгоритми

1.1. S-блок та його властивості

Однією з важливих властивостей блокових шифрів є нелінійність. Цю властивість зазвичай забезпечує єдиний нелінійний елемент шифру, **S-блок**.

S-блок (англ. substitution box, блок підстановки) — нелінійне відображення $\{0,1\}^n \to \{0,1\}^m$, причому n і m не обов'язково однакові.

S-блок є багатовимірною булевою функцією, тому його можна представити декількома способами; у даній будуть розглядатися такі:

1) Look-up Table (надалі LUT) — таблиця підстановки розміром 2^n, де для кожного можливого входу вказано результат;
2) Формульне представлення — кортеж координатних функцій S-блоку.
3) Логічна схема (англ. logic circuit) — орієнтований ациклічний граф, у якого вершинами є вхідні змінні, логічні операції та вихідні змінні.

Для малопотужних пристроїв використовують останні два представлення, тому що це є більш ефективним варіантом для реалізації. Зазвичай представлення у вигляді схеми є кращим ніж формульне представлення, бо S-блок розглядається як єдине ціле, а не як окремі булеві функції.

Для початку введемо основне означення, яке буде використовуватись протягом даної роботи.

Гейт (англ. gate, logic gate) — базова логічна операція або булева функція, яка повертає 1 або 0. У вітчизняній літературі зазвичай позначається синонімічним терміном «вентиль».

Зазвичай під гейтами маються на увазі фізичні пристрої, які реалізують певну базову операцію. Базовими операціями ми вважаємо:

1) NOT — логічне НІ, у формулах позначається $\neg x$;

2) AND — логічне ТА, у формулах позначається $x \wedge y$;
3) OR — логічне АБО, у формулах позначається $x \vee y$;
4) XOR — виключне АБО, у формулах позначається $x \oplus y$.

Також визначаються NAND, NOR, та XNOR — заперечення ТА, АБО, та виключного АБО відповідно.

Гейти можуть бути не тільки з одним або двома входами. Наприклад, також існують NAND3, NOR3, XOR3, XNOR3 — аналоги операцій, які були вище, але з трьома змінними та операції з чотирма змінними:

- $\mathrm{MAOI1}(a, b, c, d) = \neg((a \wedge b) \vee (\neg(c \vee d)))$;
- $\mathrm{MOAI1}(a, b, c, d) = \neg((a \vee b) \wedge (\neg(c \wedge d)))$.

Ці операції імплементовані як окремі гейти для чипів UMC 180nm [10].

Далі приведено основні метрики, за якими можна порівнювати булеві функції для реалізації [6].

Мультиплікативна складність (англ. multiplicative complexity) — кількість нелінійних гейтів у реалізації, які приймають два входи. Наприклад, при обмеженні реалізації функції тільки операціями {AND, OR, XOR, NOT} треба буде враховувати лише AND та OR. Оптимізація цієї властивості впливає на захист від атак за сторонніми каналами (англ. side-channel attack) з використанням випадкових масок. Оскільки ця властивість не показує складність реалізації і має суто криптографічну властивість, в подальшому ми не будемо робити великий акцент на даній метриці.

Бітова складність гейтів (англ. bit-sliced complexity) — кількість операцій {AND, OR, XOR, NOT} у реалізації. Зазвичай ця метрика використовується для архітектур, де немає окремих гейтів для NAND, NOR, XNOR [6].

Гейтовий розмір (англ. gate size, інколи gate area, area cost) — кількість гейтів, яка треба для реалізації. На відміну від попередньої метрики, бітової складності, тут дозволені операції NAND, NOR, XNOR. У подальшому ця метрика буде основною властивістю для визначення мінімальної кількості гейтів.

Оскільки внаслідок фізичних відмінностей різні гейти можуть мати неоднаковий розмір, обирають за стандартну одиницю виміру GEs (англ. gate equivalents) — розмір NAND-гейту в технології, яка розглядається [10]. Усі інші гейти вимірюються відносно GEs, тому що їх розмір, з точки зору фізичної реалізації, більший. Для технології UMC 180nm [10], розмір гейтів приведено у Таблиці 1.

Деякі дослідження розглядають не тільки розміри гейтів, але і їх час виконання відносно одне одного, що також може суттєво впливати на імплементацію [5, 9].

Шлях затримки — час виконання базового гейта, наскільки гейт «затримує» вивід результату.

Ця метрика, так само як і розмір, визначається відносно NAND-гейта і напряму корелює з розміром гейту.

Глибина (англ. depth) — максимальна сума послідовних шляхів затримки базових операцій AND, OR, NAND, NOR, NOT, XOR та XNOR у представленні формулою, або максимальна довжина шляху у орієнтованому ациклічному графі від вхідної вершини до вихідної [5].

Головною задачею у даній роботі є мінімізувати гейтовий розмір і, по можливості, глибину S-блоку. Оскільки S-блок є багатовимірною булевою функцією, треба переформулювати надані вище означення.

Гейтовий розмір S-блоку — сума гейтових розмірів координатних булевих функцій або сума усіх гейтів у схемі. *Глибина S-блоку* — максимальна глибина координатних булевих функцій або максимальний шлях у схемі від вхідної до вихідної вершини.

1.2. Основні алгоритми оптимізації

Існує ряд спеціальних алгоритмів, які спеціалізуються на оптимізації одного чи декількох критеріїв S-блоку. Ці алгоритми приймають на вхід LUT S-блоку та видають реалізацію у вигляді формул координатних функцій або логічної схеми.

1. S-box Depth Evaluation Tool [5] — рекурсивний алгоритм для побудови булевих функцій мінімальної глибини. Кожна булева функція будується починаючи з вхідних змінних, які є функціями без глибини. Поступово застосовуються логічні операції, будуючи таким чином функції різної глибини. Зберігаються тільки збалансовані булеві функції. Таким чином, будується велика таблиця, за допомогою якої можна будувати власні реалізації булевих функцій та S-блоків. Алгоритм генерує реалізації з оптимізованою глибиною та гейтовою складністю у вигляді координатних булевих функцій.
2. Метод, який використовує SAT-вирішувач — алгоритм, вперше запропонований Ко Штоффеленом [6], який будує систему булевих рівнянь у АНФ, які кодують в собі інформацію про те, які гейти будуть використовуватись, заданий гейтовий розмір або глибину, та заданий S-блок. Після цього система рівнянь конвертується у КНФ та передається SAT-вирішувачу. Алгоритм генерує реалізації з оптимізацією чогось на вибір: мультиплікативної, бітової, гейтової складності або глибини у вигляді логічної схеми;
3. Евристичні алгоритми — алгоритми, які шукають наближений, але не точний розв'язок. Найкращим є алгоритм LIGHTER [8] та його покращення у алгоритмі PEIGEN [9], які кодують проблему у граф і використовують техніку зустрічі посередині (Meet-in-the-Middle (MITM)). Алгоритм генерує реалізації з оптимізованим гейтовим розміром або глибиною у вигляді логічної схеми.

Усі вони працюють у припущенні, що $n = m$, та працюють тільки при $n \leq 5$.

Таблиця 1. Розміри гейтів у бібліотеці UMC 180nm, приведені у GEs

Гейти	AND OR	NOT	NAND NOR	XOR XNOR	NAND3 NOR3	XOR3 XNOR3	MAOI1	MOAI1
Розмір (GEs)	1.33	0.67	1.00	3.00	1.33	4.67	2.67	2.00

2. Аналіз складності алгоритмів

Задача знаходження мінімальної можливої реалізації булевої функції є Σ_p^2-складною задачею, тому це дає загальне розуміння про складність знаходження такої оптимізації. Далі наведено аналіз приблизної складності алгоритмів для оптимізації S-блоку.

2.1. S-box Depth Evaluation Tool

Як вже було згадано раніше, S-box Depth Evaluation Tool — це рекурсивний алгоритм, який генерує булеві функції для кожної глибини d таким чином:

- $d = 0$: самі вхідні біти (змінні) є функціями з глибиною 0;
- $d = 0.5$: застосувати операцію NOT до функцій з глибиною 0;
- $d = 1$: застосувати операцію NAND/NOR до функцій з глибиною 0;
- $d = 1.5$: три різних способи щоб згенерувати таку функцію:
  1) використати операцію AND/OR до двох різних функцій з глибиною 0;
  2) використати операцію NAND/NOR до двох різних функцій з глибиною 0.5;
  3) використати операцію NOT до функцію з глибиною 1.
- $d \geq 2$: чотири різних способи щоб згенерувати таку функцію: використати операції XOR/XNOR, AND/OR, NAND/NOR та NOT до функцій з глибиною $d-2$, $d-1.5$, $d-1$, $d-0.5$ відповідно.

У дослідженні [5] розглядались булеві функції розміру $n = 4$. У фінальний результат записуються тільки збалансовані функції. Якщо булеві функції однакової глибини під час генерування повторюються, то у фінальний результат записуються тільки функції з найменшим гейтовим розміром.

Фактично, цей алгоритм перебирав усі булеві функції розміру $n = 4$ по декілька разів. Усього таких функцій буде $2^{2^n} = 2^{2^4} = 2^{16}$, а усього було згенеровано $\approx 2^{55}$ різних імплементацій. В теорії цей алгоритм можна застосувати для більшого n, але на практиці це є дуже складною задачею.

2.2. Методи, які використовують SAT-вирішувач

Алгоритми Ко Штоффеллена та його модифікації складаються з декількох етапів, але самою складною частиною є саме знаходження результату згенерованої SAT-задачі. Через те, що немає точної оцінки для розв'язку SAT-задачі і вона залежить від складності КНФ, яка подається на вхід, то оцінити роботу алгоритму важко. Але відомо, що результати у цих роботах були приведені тільки для $n = 4$, навіть з використанням потужних обчислювальних ресурсів.

2.3. Евристичний алгоритм LIGHTER

LIGHTER є самим швидким з усіх алгоритмів, які розглядаються. Також він дозволяє задавати розмір гейтів для оптимізації, як, наприклад, було приведено у таблиці 1.

Нехай $\boldsymbol{B}$ — множина операцій, для яких буде генеруватись реалізація S-блоку. Алгоритм будує одночасно два орієнтованих графа, у яких вершинами є LUT S-блоку, а ребрами є операції, які показують, чи можна отримати задану вершину за допомогою операції у $\boldsymbol{B}$. Початковою вершиною є тотожний S-блок, а кінцевою — S-блок, імплементацію якого ми хочемо знайти. З початкової вершини будується прямий граф, застосовуючи операції з $\boldsymbol{B}$. Так само будується з кінцевої вершини, але зворотній граф. Алгоритм зупиняє свою роботу при знаходженні перетину цих двох графів, і шлях від початкової до кінцевої вершини і є імплементацію цього S-блоку.

Складність цього алгоритму залежить від $b = |\boldsymbol{B}|$ та фінальної кількості гейтів в імплементації k. Цей алгоритм використовує техніку зустрічі посередині, тому його асимптотична складність $O(b^{\frac{k}{2}})$.

Це є справедливим і для PEIGEN, який покращує роботу LIGHTER, але не змінює основну структуру алгоритму. Також, для $n > 5$ цей алгоритм стає занадто складним, тому що k зростає експоненційно відносно n.

3. Результати застосування алгоритмів

У цій роботі було згенеровано реалізації восьми S-блоків, наведених у роботі [11], а саме:

$$S_1 = [7, 9, 4, D, 0, 2, C, B, A, 8, 1, 6, E, 5, F, 3],$$
$$S_2 = [1, 9, 6, 5, B, E, 2, 8, 4, A, F, 3, 0, 7, C, D],$$
$$S_3 = [A, C, 3, 8, B, 7, D, 0, 4, 5, 1, F, E, 9, 6, 2],$$
$$S_4 = [E, A, F, 1, 0, D, 7, 4, 5, 2, 8, 6, 3, B, 9, C],$$
$$S_5 = [B, 0, D, E, 6, 4, 7, 9, 5, 1, C, 2, 8, F, A, 3],$$
$$S_6 = [1, C, 3, 8, 0, 6, E, D, F, B, 4, 5, 9, 2, A, 7],$$
$$S_7 = [E, 7, B, D, 8, 2, 5, 6, 3, 0, 1, C, F, 9, 4, A],$$
$$S_8 = [4, 3, A, B, D, E, 8, 5, 9, 1, 7, C, F, 2, 6, 0].$$

Отримані реалізації будуть розглядати тільки гейти з одним або двома входами, бо інші гейти підтримуються тільки LIGHTER. Також для позначення гейтового розміру було взято приблизні значення, які використовувались у S-box Depth Evaluation Tool [5]:

- NOT= 0.5 GEs;

Таблиця 2. Гейтовий розмір згенерованих реалізацій S-блоків

S-блок	S-box Depth Evaluation Tool [5]	Ko Stoffelen [6]	LIGHTER [5]
S1	39 GEs	35.5 GEs	26 GEs
S2	39 GEs	41.5 GEs	22.5 GEs
S3	38.5 GEs	37 GEs	25 GEs
S4	38.5 GEs	38.5 GEs	27.5 GEs
S5	38.5 GEs	35 GEs	27.5 GEs
S6	39.5 GEs	35.5 GEs	24.5 GEs
S7	38 GEs	35 GEs	23 GEs
S8	38 GEs	32 GEs	27.5 GEs

- NAND/NOR= 1 GEs;
- AND/OR= 1.5 GEs;
- XOR/XNOR= 2 GEs.

Результати застосування алгоритмів наведено в таблиці 2. Можна помітити, що найкращий результат видає LIGHTER, не дивлячись на те, що це евристичний алгоритм. Результати для LIGHTER є ще кращими, якщо використовувати гейти і розміри гейтів з таблиці 1. Результати алгоритму, який використовує SAT-вирішувач в деяких випадках гірше, тому що цей метод не враховує розміри гейтів, і кількість XOR-гейтів у такій імплементації може бути завеликою. В теорії, при оптимальній реалізації через алгоритм Ko Stofellen результати будуть набагато кращі, але результати при запусках для кількості гейтів нижче 20 не були отримані.

Висновки

У даній роботі розглянуто основні представлення S-блоку у малопотужних пристроях, властивості які розглядаються при їх реалізації та алгоритми, які генерують реалізації S-блоків з оптимальними властивостями. Також було застосовано згадані алгоритми для оптимізації 8 S-блоків з розміром $n = 4$. Отримані результати є прийнятними за властивостями для малопотужних пристроїв і можуть використовуватись на практиці.

Перелік використаних джерел

1. Logic minimization algorithms for VLSI synthesis. T. 2 / R. K. Brayton, G. D. Hachtel, C. McMullen, A. Sangiovanni-Vincentelli. — Springer Science & Business Media, 1984.
2. *Canright D.* A very compact S-box for AES // International Workshop on Cryptographic Hardware and Embedded Systems. — Springer. 2005. — С. 441—455.
3. *Morioka S., Satoh A.* An optimized S-Box circuit architecture for low power AES design // International Workshop on Cryptographic Hardware and Embedded Systems. — Springer. 2002. — С. 172—186.
4. *Boyar J., Peralta R.* A new combinational logic minimization technique with applications to cryptology // Experimental Algorithms: 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings 9. — Springer. 2010. — С. 178—189.
5. Midori: A Block Cipher for Low Energy / S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, F. Regazzoni // Advances in Cryptology – ASIACRYPT 2015. — Springer Berlin Heidelberg, 2015. — С. 411—436. — ISBN 9783662488003. — DOI: 10.1007/978-3-662-48800-3_17.
6. *Stoffelen K.* Optimizing S-Box Implementations for Several Criteria Using SAT Solvers // Fast Software Encryption. — Springer Berlin Heidelberg, 2016. — С. 140—160. — ISBN 9783662529935. — DOI: 10.1007/978-3-662-52993-5_8.
7. *Zhang F., Huang Z.* Optimizing S-box Implementations Using SAT Solvers: Revisited // IACR Cryptol. ePrint Arch. — 2023. — Т. 2023. — С. 1721. — URL: https://api.semanticscholar.org/CorpusID:265355622.
8. Optimizing Implementations of Lightweight Building Blocks / J. Jean, T. Peyrin, S. M. Sim, J. Tourteaux // IACR Transactions on Symmetric Cryptology. — 2017. — Груд. — С. 130—168. — ISSN 2519-173X. — DOI: 10.46586/tosc.v2017.i4.130-168.
9. PEIGEN – a Platform for Evaluation, Implementation, and Generation of S-boxes / Z. Bao, J. Guo, S. Ling, Y. Sasaki // IACR Transactions on Symmetric Cryptology. — 2019. — Бер. — С. 330—394. — ISSN 2519-173X. — DOI: 10.46586/tosc.v2019.i1.330-394.
10. Pushing the Limits: Searching for Implementations with the Smallest Area for Lightweight S-Boxes / Z. Lu, W. Wang, K. Hu, Y. Fan, L. Wu, M. Wang // IACR Cryptol. ePrint Arch. — 2021. — Т. 2021. — С. 1644. — URL: https://api.semanticscholar.org/CorpusID:245104018.
11. *Яковлєв С. В.* Збалансовані критерії якості довгострокових ключових елементів алгоритму ГОСТ 28147-89 // Інформаційні технології та комп'ютерна інженерія. — 2009. — № 1. — С. 48—55.