Pushing the Limits: Searching for Implementations with the Smallest Area for Lightweight S-Boxes

Zhenyu Lu, Weijia Wang, Kai Hu, Yanhong Fan, Lixuan Wu, and Meiqin Wang(✉)

[1] School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, 266237, China
[2] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao, Shandong, 266237, China
{luzhenyu,hukai,fanyh,lixuanwu}@mail.sdu.edu.cn,
{wjwang,mqwang}@sdu.edu.cn

Abstract. The area is one of the most important criteria for an S-box in hardware implementation when designing lightweight cryptography primitives. The area can be well estimated by the number of gate equivalent (GE). However, to our best knowledge, there is no efficient method to search for an S-box implementation with the least GE. Previous approaches can be classified into two categories, one is a heuristic that aims at finding an implementation with a satisfying but not necessarily the smallest GE number; the other one is SAT-based focusing on only the smallest number of gates while it ignored that the areas of different gates vary. Implementation with the least gates would usually not lead to the smallest number of GE.

In this paper, we propose an improved SAT-based tool targeting optimizing the number of GE of an S-box implementation. Given an S-box, our tool can return the implementation of this S-box with the smallest number of GE. We speed up the search process of the tool by bit-sliced technique. Additionally, our tool supports 2-, 3-, and 4-input gates, while the previous tools cover only 2-input gates. To highlight the strength of our tool, we apply it to some 4-bit and 5-bit S-boxes of famous ciphers. We obtain a better implementation of RECTANGLE's S-box with the area of 18.00GE. What's more, we prove that the implementations of S-boxes of PICCOLO, SKINNY, and LBLOCK in the current literature have been optimal. When using the DC synthesizer on the circuits produced by our tool, the area are much better than the circuits converted by DC synthesizers from the lookup tables (LUT). At last, we use our tool to find implementations of 5-bit S-boxes, such as those used in KECCAK and ASCON.

Keywords: Lightweight ciphers · S-box implementations · Gate equivalent complexity · SAT-solvers

1 Introduction

Lightweight cryptographic primitives are deployed more and more in the source-constraint devices that manipulate sensitive data. The National Institute of Standards and Technology (NIST) has initiated a competition to call for a new lightweight cryptography standard for constrained environments [12]. The designer of lightweight cryptography needs to consider both the security property and implementation performance. The hardware implementation performance includes many criteria, e.g., throughput, area, energy, power, and latency, where the area is a crucial criterion for the implementation of lightweight ciphers.

Since the area cost of different gates depends on the technology library, measuring and comparing the area cost of implementations requires a standard unit. A gate equivalent usually stands for the unit of measure which allows specifying manufacturing-technology-independent complexity of digital electronic circuits. Practically, the NAND constitutes the unit area commonly referred to as a gate equivalent while the GE of other gates are measured based on the NAND gates. For example, in the library of UMC 180nm [8], the GE of some gates are listed in Table 1.

Table 1. Area cost of typical cell gates under UMC 180nm library [8]. The values are given in GE.

Techniques	AND OR	NOT	NAND NOR	XOR XNOR	NAND3 NOR3	XOR3 XNOR3	MAOI1	MOAI1
UMC 180nm	1.33	0.67	1.00	3.00	1.33	4.67	2.67	2.00

To predict the area of a hardware implementation of a given S-box, we commonly compute the number of GE of this implementation. As a result, to find an optimal implementation of an S-box with the smallest area, we need to find the optimal combination of a set of gates whose number of GE is the smallest.

Before this work, no approach is suitable to find the implementation of an S-box with the smallest area directly. Here we briefly introduce two main-stream methods to find the implementation of an S-box.

Heuristic search. In the domain of logic synthesis, several heuristic algorithms provide satisfactory solutions, such as BOOM [7] and ESPRESSO [13] which are probably implemented in many commercial synthesizers. An automated tool `LIGHTER` proposed by Jean et al. [9] uses a graph-based meet-in-the-middle search algorithm under the assumption that every instructions is invertible. Despite of the efficiency and practical applicability for different S-boxes, these algorithms rely on some heuristics and are infeasible to prove that their results are optimal implementation of S-box circuits.

SAT-based search. At FSE 2016, Stoffelen models the problem of finding an efficient implementation of a lightweight S-box as a SAT problem [15]. Then with a SAT solver, this tool can find the implementation of S-box with the smallest number of gates. However, as Table 1 shows, the area costs of different gates are different. The smallest number of gates will still lead to a large number of GE.

Our Contributions. In this paper, we give the first method to search for the optimal area implementation of small S-boxes by SAT solver. The main contributions are shown below.

A New Searching Algorithm. Based on the SAT method [15], we propose an algorithm to find the optimal implementation of a lightweight S-box focusing on the area. We reduce the search space by a pre-computed algorithm. This algorithm first searches for the optimal implementation in the terms of number of gates, then it calculates the lower and upper bounds of the number of gates and area.

Within this range, we find out the optimal implementation by querying the SAT solver. The number of variables in the SAT model has a great dependence on the types and the number of gates. As the number of variables increases, the efficiency would be lower. Consequently, we use the bit-sliced technique to reduce the number of variables and then speed up the model.

A Generalization to 2-, 3-, 4-input Gates. In [1], the authors have shown that replacing several simple gates with two inputs complex gates with multiple inputs can save the area significantly. Insipred by this, on the basis of the 2-input gate model [15], our model includes complex gates. Our model gives a unified expression that can describe gates with 2 inputs, 3 inputs(e.g., XOR3, XNOR3, OR3, NOR3, AND3, and NAND3) and 4 inputs (e.g., MOAI1 and MAOI1).

Better S-box Implementations. We apply our method to many 4- and 5-bit S-boxes of popular ciphers such as RECTANGLE [17], PICCOLO [14], SKINNY [2], LBLOCK [16], KECCAK [3] and ASCON [5].

We manage to find an improved circuit of RECTANGLE's S-box with 18.00 GE cost which is better than `LIGHTER`'s and we can verify that the circuits of PICCOLO, SKINNY and LBLOCK's S-boxes have the optimal area cost under the 2-, 3- and 4-input gates we considered. In addition, due to the bit-sliced technique, our model is also useful in finding the implementation of the 5-bit S-boxes.

Organization of the paper. In Section 2, we first introduce some preliminary notions and recall some previous works on the implementation of S-box. We introduce our new model with the pre-computed algorithm and bit-sliced technique to search the optimal area implementation of an S-box in Section 3. In Section 4, we provide an comparison between our results and previous works. At the end, we conclude the paper in Section 5.

2 Preliminaries

In this section, we first present some definitions and notions used in this paper. Then, we briefly recall Stoffelen's SAT-based tool in [15].

2.1 Notations

Table 2. List of Boolean operators implemented by standard cell gates from the libraries. $\wedge, \vee, \oplus, \neg$ stand for logical and, or, exclusive or, not [9], respectively.

Operation	Function	Operation	Function
NAND	$(a, b) \to \neg(a \wedge b)$	XOR	$(a, b) \to a \oplus b$
NOR	$(a, b) \to \neg(a \vee b)$	XNOR	$(a, b) \to \neg(a \oplus b)$
AND	$(a, b) \to a \wedge b$	NAND3	$(a, b, c) \to \neg(a \wedge b \wedge c)$
OR	$(a, b) \to (a \vee b)$	NOR3	$(a, b) \to \neg(a \vee b \vee c)$
NOT	$a \to \neg a$	XOR3	$(a, b, c) \to (a \oplus b \oplus c)$
MAOI1	$(a, b, c, d) \to \neg((a \wedge b) \vee (\neg(c \vee d)))$	XNOR3	$(a, b, c) \to \neg(a \oplus b \oplus c)$
MOAI1	$(a, b, c, d) \to \neg((a \vee b) \wedge (\neg(c \wedge d)))$		

The combinatorial cell gates implement classical Boolean operations, whose functional behavior is shown in Table 2. In this paper, we use *logical connectives* to denote the types of operations, i.e., let $\wedge$, $\vee$, $\oplus$, $\neg$ denote AND, OR, XOR, NOT, respectively, and let $\uparrow$, $\downarrow$, $\leftrightarrow$ denote NAND, NOR, XNOR, respectively. The notations used in this paper are listed in Table 3.

2.2 Stoffelen's SAT-based Tool

The Boolean satisfiability problem (SAT) is the problem of determining whether there exists an evaluation for the binary variables such that the value of the given Boolean formula equals one. Through translating a problem into a SAT problem, we could then take the off-the-shelf solvers to solve this SAT problem, and finally get the corresponding answer to the original problem.

Since our tool can be regarded as an improved version of Stoffelen's SAT-based tool [15] that aims at finding the implementation with smallest number of GE rather than only the number of gates, we introduce the basic methods used in his tool. In [15], Stoffelen explores the feasibility of applying SAT solvers to optimize implementations of small S-boxes for the criteria including of the number of gates. He proposed a binary model to solve the following decision

Table 3. List of notations in this paper.

Notations	Definitions
K	K represents the number of gates.
G	G represents the area cost of a circuit.
x_i (resp. y_j)	Boolean variables, represent S-box inputs (resp. outputs).
q_{2i}	The i-th gate input , $q_{2i} \in \mathbb{F}_2$.
t_i	The i-th gate output, $t_i \in \mathbb{F}_2$.
a_i	Coefficient variables $a_i \in \mathbb{F}_2$ represent wiring between gates. (More details can refer to example 1.)
b_i	Variables $b_i \in \mathbb{F}_2$ determine the types of gates. (More details can refer to example 1.)
$Cost[i]$	The array $Cost[i]$ represents the cost of different gate operations.

problem: *Is there a circuit that implements an S-box* $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ *and that uses at most* K *logic operations?*

He uses a method in [4, 11] to transform the decision problem into a model. This model encodes each gate as an Algebraic Normal Form (ANF) equation and can judge the existence of solutions when given the number of gates. To get the smallest number of gates, it should exhaust K until finding the smallest one that there exists an implementation of an S-box.

As an example, we give a model of a decision problem whether there is a circuit implements an 2-bit toy S-box with 2 gates.

Example 1. Given a 2-bit S-box in Table 4 and we encode the model of this S-box as follows.

Table 4. Lookup table of the 2-bit S-box.

x	0	1	2	3
$S(x)$	3	2	0	1

Encode the input and output of the S-box. We encode the S-box as Boolean variables x_i and y_i.

$$
\begin{aligned}
&x_0 = 0,\ x_1 = 0,\ y_0 = 1,\ y_1 = 1; \quad //denote\ S(0) = 3\\
&x_2 = 0,\ x_3 = 1,\ y_2 = 1,\ y_3 = 0; \quad //denote\ S(1) = 2\\
&x_4 = 1,\ x_5 = 0,\ y_4 = 0,\ y_5 = 0; \quad //denote\ S(2) = 0\\
&x_6 = 1,\ x_7 = 1,\ y_6 = 0,\ y_7 = 1; \quad //denote\ S(3) = 1
\end{aligned}
$$

Then, for each x and $S(x)$, this model needs one set of equations as follows to represent a circuit with K gates and there are a total of 2^2 sets.

Encode a decision of choosing two inputs of a gate. The Boolean variables q_i represent the inputs of a gate. For example, q_0 and q_1 are two inputs of the gate t_0, while q_2 and q_3 are two inputs of the gate t_1.

$$\begin{aligned}
q_0 &= a_0 \cdot x_0 + a_1 \cdot x_1 \\
q_1 &= a_2 \cdot x_0 + a_3 \cdot x_1 \\
q_2 &= a_4 \cdot x_0 + a_5 \cdot x_1 + a_6 \cdot t_0 \\
q_3 &= a_7 \cdot x_0 + a_8 \cdot x_1 + a_9 \cdot t_0
\end{aligned}$$

One q_i must come from one of the S-box's inputs or the output of a previous gate. This constraint can be described as that only one of the variables a_i in an equation can be equal to 1.

$$\begin{aligned}
&a_0 \cdot a_1 = 0. \\
&a_2 \cdot a_3 = 0. \\
&a_4 \cdot a_5 = 0 \ AND \ a_4 \cdot a_6 = 0 \ AND \ a_5 \cdot a_6 = 0. \\
&a_7 \cdot a_8 = 0 \ AND \ a_7 \cdot a_9 = 0 \ AND \ a_8 \cdot a_9 = 0.
\end{aligned}$$

Encode the decision of choosing a type of gate. The variables b_i determine what kind of gate the t_i will represent, as can be seen in Table 5. When the value of the pattern $b_{3i}||b_{3i+1}||b_{3i+2}$ is different, t_i represents different kind of gate, such as AND, OR, XOR, NAND, NOR, and XNOR.

$$\begin{aligned}
t_0 &= b_0 \cdot q_0 \cdot q_1 + b_1 \cdot q_0 + b_1 \cdot q_1 + b_2 \\
t_1 &= b_2 \cdot q_2 \cdot q_3 + b_3 \cdot q_2 + b_3 \cdot q_3 + b_4
\end{aligned}$$

There are a total of K variables t_i to represent K different gates.
Encode the decision of choosing the output of the circuit. The Boolean variables y_i also represent the outputs of the circuit.

$$\begin{aligned}
y_0 &= a_{18} \cdot x_0 + a_{19} \cdot x_1 + a_{20} \cdot t_0 + a_{21} \cdot t_1 \\
y_1 &= a_{22} \cdot x_0 + a_{23} \cdot x_1 + a_{24} \cdot t_0 + a_{25} \cdot t_1
\end{aligned}$$

Similar to q_i, one y_i must come from one of the S-box's inputs or the output of a gate. This constraint can be described as that only one of the variables a_i in an equation can be equal to 1 too.

3 Optimizing Implementations for S-boxes

We measure the gate sizes in terms of Gate Equivalent (GE), which is a normalized ratio using the area of a 2-input NAND gate as a common reference.

Table 5. Encoding of different types of gates.

$b_{3i}\|\|b_{3i+1}\|\|b_{3i+2}$	Operations	Gate function
0 0 0	0	0
0 0 1	1	1
0 1 0	XOR	$q_{2i} \oplus q_{2i+1}$
0 1 1	XNOR	$q_{2i} \leftrightarrow q_{2i+1}$
1 0 0	AND	$q_{2i} \wedge q_{2i+1}$
1 0 1	NAND	$q_{2i} \uparrow q_{2i+1}$
1 1 0	OR	$q_{2i} \vee q_{2i+1}$
1 1 1	NOR	$q_{2i} \downarrow q_{2i+1}$

3.1 Main Idea of Our Model

In this section, we introduce how to improve Stoffelen's tool for optimizing the area of an S-box. Stoffelen's model can produce an implementation with a set of K gates and we denote this set as $\mathcal{I}$. We can add the cost of each gate up to obtain the area of this implementation. Let G denote the area of the implementation, we have

$$G = \sum_{g_i \in I} Cost_{g_i}, \tag{1}$$

where $Cost_{g_i}$ is the area of the gate g_i in $\mathcal{I}$. Since we want to search for an implementation of the S-box with G area, Equation 1 is naturally the objective function of our new model together with all equations in Stoffelen's model. This model can determine whether a circuit can implement an S-box with K gates and G area.

However, even if there exists a circuit, the area is not the smallest one. It needs to exhaust K and G and encode the corresponding decision problem to find the smallest area implementation by querying the SAT solver.

In this term, there are three limitations of Stoffelen's tool. Firstly, the NOT operation is not considered, because in his model a NOT gate is always redundant for it can always be incorperated into a new combinatorial gate. For example, a NOT gate and an AND gate can be combined into a NAND gate. However, if we want to consider the area, our model cannot ignore the NOT gate. In addition, his tool only covers 2-inputs gates, while the complex gates such as 3- and 4-inputs gates have a great effect on implementations. Secondly, the area costs of different gates are different and his model could not find the smallest number of GE of an S-box's implementation. Finally, as the number of gates increases, Stoffelen's model needs more variables, which results in a lower efficiency and the model does not work for 5-bit S-boxes and even some 4-bit S-boxes.

To overcome these limitations, we first re-encode the ANF equation of an gate including the NOT gate and 2-, 3-, 4-input gates. Then, we propose a new decision problem: *is there a circuit that implements an S-box so that the area cost at most G?* To solve this problem, we set an array to denote the area cost of different gates and give an algorithm to determine the upper and lower bounds of the K and G. In the end, we use a technique called bit-sliced to reduce the variables in our model and speed up the search.

3.2 Encode the NOT gate and Complex Gates

In this section, we re-encode the equation of a gate to include the NOT gate and 2-, 3-, 4-input gates. The 3-input gates include the AND3, OR3, XOR3, NAND3, NOR3 and XNOR3 gates while the MAOI1 and MOAI1 gates are two 4-input gates.

NOT gate. Firstly, we re-encode the gates equation from Stoffelen's model as follows to add the NOT gate.

$$t = b_0 \cdot q_0 \cdot q_1 + b_1 \cdot q_0 + b_1 \cdot q_1 + \boldsymbol{b_2 \cdot q_0} + b_3 \quad (2)$$

In this equation, when $b_2 = 0$, the patterns $b_0||b_1||b_3$ represent the same gates to the patterns Stoffelen's model, which can be seen in Table 6.

Table 6. Improve the encoding of different types of gates.

$\boldsymbol{b_0}\|\|\boldsymbol{b_1}\|\|\boldsymbol{b_2}\|\|\boldsymbol{b_3}$	**Operations**	**Gate function**
0 0 1 1	**NOT**	$\neg q_0$
0 1 0 0	XOR	$q_0 \oplus q_1$
0 1 0 1	XNOR	$q_0 \leftrightarrow q_1$
0 1 1 1	**NOT**	$\neg q_1$
1 0 0 0	AND	$q_0 \wedge q_1$
1 0 0 1	NAND	$q_0 \uparrow q_1$
1 1 0 0	OR	$q_0 \vee q_1$
1 1 0 1	NOR	$q_0 \downarrow q_1$

From Table 6, the patterns $b_0||b_1||b_2||b_3$ do not cover the whole space of $\mathbb{F}_2^4$. For example, when b_2 equals to 1, b_3 should equal to 1 and b_0 equal to 0. We describe this case as a constraint in our model to make sure that each pattern is corresponding to one gate.

$$Cst_1 = \{b_3 = 1 \textit{ and } b_0 = 0 | b_2 = 1\}.$$

However, more complex gates, such as 3-input and 4-input operations have a great effect on the number of GE. For example, two consecutive XOR gates can be replaced by a XOR3 gate and the XOR3 gate cost 4.67 GE which is smaller than two XOR gates.

3-input gates. We improve the Equation (2) to add the 3-input gates, such as AND3, NAND3, OR3, NOR3, XOR3, and XNOR3.

$$\begin{aligned} t =& \boldsymbol{b_0 \cdot q_0 \cdot q_1 \cdot q_2 + b_1 \cdot q_0 \cdot q_1 + b_1 \cdot q_0 \cdot q_2 + b_1 \cdot q_1 \cdot q_2 +} \\ & \boldsymbol{b_1 \cdot q_0 + b_1 \cdot q_1 + b_1 \cdot q_2 + b_2 \cdot q_0 + b_2 \cdot q_1 + b_2 \cdot q_2 +} \\ & b_3 \cdot q_0 \cdot q_1 + b_4 \cdot q_0 + b_4 \cdot q_1 + b_5 \cdot q_0 + b_6. \end{aligned} \tag{3}$$

This equation adds three b_i and one q_i to encode the 3-input gates. We propose the detail of the gate in Table 7.

Table 7. Encoding of different types of 2-input and 3-input gates.

$\boldsymbol{b_0 \| b_1 \| b_2 \| b_3 \| b_4 \| b_5 \| b_6}$							**Operations**	**Gate function**
0	0	0	0	0	1	1	NOT	$\neg q_0$
0	0	0	0	1	0	0	XOR	$q_0 \oplus q_1$
0	0	0	0	1	0	1	XNOR	$q_0 \leftrightarrow q_1$
0	0	0	0	1	1	1	NOT	$\neg q_1$
0	0	0	1	0	0	0	AND	$q_0 \wedge q_1$
0	0	0	1	0	0	1	NAND	$q_0 \uparrow q_1$
0	0	0	1	1	0	0	OR	$q_0 \vee q_1$
0	0	0	1	1	0	1	NOR	$q_0 \downarrow q_1$
1	0	0	0	0	0	0	AND3	$q_0 \wedge q_1 \wedge q_2$
1	0	0	0	0	0	1	NAND3	$\neg(q_0 \wedge q_1 \wedge q_2)$
1	1	0	0	0	0	0	OR3	$q_0 \vee q_1 \vee q_2$
1	1	0	0	0	0	1	NOR3	$\neg(q_0 \vee q_1 \vee q_2)$
0	0	1	0	0	0	0	XOR3	$q_0 \oplus q_1 \oplus q_2$
0	0	1	0	0	0	1	XNOR3	$\neg(q_0 \oplus q_1 \oplus q_2)$

Similarly, the patterns $b_0||b_1||b_2||b_3||b_4||b_5||b_6$ of this equation do not cover the whole space of $\mathbb{F}_2^7$. When $b_0 = b_1 = b_2 = 0$, the patterns $b_3||b_4||b_5||b_6$ represent the gates are the same as the 2-input ones. To make sure each pattern represents one gate, we add the following constraints in our model.

$$\begin{aligned} Cst_1 =& \{b_6 = 1 \textit{ and } b_3 = 0 | b_5 = 1\}. \\ Cst_2 =& \{b_2 = b_3 = b_4 = b_5 = 0 | b_0 = 1\}. \\ Cst_3 =& \{b_0 = 1 | b_1 = 1\}. \\ Cst_4 =& \{b_0 = b_1 = b_3 = b_4 = b_5 = 0 | b_2 = 1\}. \end{aligned}$$

4-input gates. The gate functions of the MAOI1 and MOAI1 4-input gates are listed in Table 8. It is easy to know that $MAOI1(a,b,c,d) = \neg MOAI1(a,b,c,d)$. We further improve Equation (3) to add the 4-input gates into our model. Firstly, we decompose the function of MAOI1 gate as follows

$$\begin{aligned}
&MAOI1(a,b,c,d)\\
=&\neg((a\wedge b)\vee(\neg(c\vee d)))\\
=&(\neg(a\wedge b))\wedge(c\vee d)\\
=&(b_0\cdot a\cdot b+b_0)\cdot(b_1\cdot c\cdot d+b_1\cdot c+b_1\cdot d)\\
=&b_0b_1\cdot abcd+b_0b_1\cdot abc+b_0b_1\cdot abd+b_0b_1\cdot cd+b_0b_1\cdot c+b_0b_1\cdot d\\
=&b_*\cdot abcd+b_*\cdot abc+b_*\cdot abd+b_*\cdot cd+b_*\cdot c+b_*\cdot d.
\end{aligned} \tag{4}$$

Then, we add one b_i and one q_i to encode all 2-, 3- and 4-input gates.

$$\begin{aligned}
t=&\boldsymbol{b_0\cdot q_0\cdot q_1\cdot q_2\cdot q_3+b_0\cdot q_0\cdot q_1\cdot q_2+}\\
&\boldsymbol{b_0\cdot q_0\cdot q_1\cdot q_3+b_0\cdot q_2\cdot q_3+b_0\cdot q_2+b_0\cdot q_3+}\\
&b_1\cdot q_0\cdot q_1\cdot q_2+b_2\cdot q_0\cdot q_1+b_2\cdot q_0\cdot q_2+b_2\cdot q_1\cdot q_2+\\
&b_2\cdot q_0+b_2\cdot q_1+b_2\cdot q_2+b_3\cdot q_0+b_3\cdot q_1+b_3\cdot q_2+\\
&b_4\cdot q_0\cdot q_1+b_5\cdot q_0+b_5\cdot q_1+b_6\cdot q_0+b_7.
\end{aligned} \tag{5}$$

In Table 8, we propose the details of the 4-input gate. Besides, to make sure each pattern represents one gate, we add one more constraint in our model.

Table 8. Encoding of different types of 4-inputgates.

$\boldsymbol{b_0 \,\|\, b_1\|\|b_2\|\|b_3\|\| b_4\|\|b_5\|\|b_6\|\|b_7}$								Operations	Gate function
1	0	0	0	0	0	0	0	MAOI1	$\neg((q_0\wedge q_1)\vee(\neg(q_2\vee q_3)))$
1	0	0	0	0	0	0	1	MOAI1	$\neg((q_0\vee q_1)\wedge(\neg(q_2\wedge q_3)))$

$$Cst_5=\{b_1=b_2=b_3=b_4=b_5=b_6=0|b_0=1\}.$$

In summary, Figure 1 gives the framework of our model and the number of the input variables q_i corresponding to each gate t_i in the model has become to 4 and a set of equations has a total of $4K$ inputs variables q_i.

We also use the decision problem whether there is a circuit implements an 2-bit toy S-box and that uses at most 3 logic operations as an example. The set of the equation re-encode as

$$\begin{aligned}
q_0=&a_0\cdot x_0+a_1\cdot x_1\\
q_1=&a_2\cdot x_0+a_3\cdot x_1
\end{aligned}$$

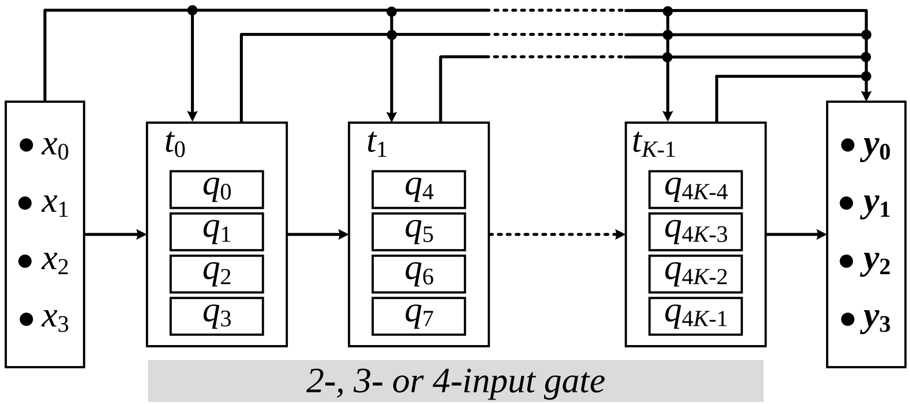

Fig. 1. Illustration of our model.

$$
\begin{aligned}
q_2 =& a_4 \cdot x_0 + a_5 \cdot x_1 \\
q_3 =& a_6 \cdot x_0 + a_7 \cdot x_1 \\
t_0 =& \boldsymbol{b_0 \cdot q_0 \cdot q_1 \cdot q_2 \cdot q_3 + b_0 \cdot q_0 \cdot q_1 \cdot q_2 +} \\
& \boldsymbol{b_0 \cdot q_0 \cdot q_1 \cdot q_3 + b_0 \cdot q_2 \cdot q_3 + b_0 \cdot q_2 + b_0 \cdot q_3 +} \\
& b_1 \cdot q_0 \cdot q_1 \cdot q_2 + b_2 \cdot q_0 \cdot q_1 + b_2 \cdot q_0 \cdot q_2 + b_2 \cdot q_1 \cdot q_2 + \\
& b_2 \cdot q_0 + b_2 \cdot q_1 + b_2 \cdot q_2 + b_3 \cdot q_0 + b_3 \cdot q_1 + b_3 \cdot q_2 + \\
& b_4 \cdot q_0 \cdot q_1 + b_5 \cdot q_0 + b_5 \cdot q_1 + b_6 \cdot q_0 + b_7. \\
\ldots & \\
y_0 =& a_{36} \cdot x_0 + a_{37} \cdot x_1 + a_{38} \cdot t_0 + a_{39} \cdot t_1 + a_{40} \cdot t_2 \\
y_1 =& a_{41} \cdot x_0 + a_{42} \cdot x_1 + a_{43} \cdot t_0 + a_{44} \cdot t_1 + a_{45} \cdot t_2
\end{aligned}
$$

It can be seen from the set of equations, the number of variables including a_i, q_i and b_i which has grown a lot.

3.3 Searching for the Implementation with the Smallest Area

As mentioned before, we propose a new decision problem: is there a circuit that implements an S-box with the area cost at most G GE? At first glance, it seems easy to solve this problem by slightly adjusting Stoffelen's tool. However, Stoffelen's SAT-based model needs to encode the problem based on a determined K. It could not determine the number of variables in a set of equations without knowing the K. On the other hand, it is simple to solve a sub-problem, whether a circuit can implement an S-box that uses determined K logic operations with G GE.

In this section, we first solve this sub-problem by Algorithm 1. Then we propose Algorithm 2 to determine the range of the search space to find the smallest number of GE step by step.

For the first step, to solve the sub-problem, we encode the area cost of different gates as an array $Cost[\,]$ in our model according to Table 1, Table 7 and Table 8. The indexes of the array are the different types of gates represented by the patterns $Gate_i = b_{7i}||b_{7i+1}||b_{7i+2}||b_{7i+3}||b_{7i+4}||b_{7i+5}||b_{7i+6}$. Meanwhile, the entries of the array represent the number of GE of different types of gates. Note that the Boolean vector can only represent integers, so we expand all the number of GE by 3 times simultaneously. For example, the AND gate costs 1.33GE, so $Cost[0bin00001000] = 0bin0100$. Next, we sum the cost of all gates and denote it as $G = Cost[Gate_0] + Cost[Gate_1] + ... + Cost[Gate_{K-1}]$ in our model. Seeing the pseudo-code of this model in Algorithm 1. Note that this algorithm could only solve the decision problem and return 0 or 1 when given the target area cost G and the number K of gates.

Even if there is a solution when giving the number of gates K and the target area cost G, it could not be the smallest number of GE. The second step is to determine the search space $\mathcal{V}(K, G)$ where the (K_{opt}, G_{opt}) of the global optimal implementation lie in. We can use the model in Section 3.2 to find an implementation with the smallest number of gates K_{low}, then we give a proposition.

Proposition 1. *K_{low} represents the smallest number of gates of an S-box's implementation. We set the area cost G_{up} of this implementation as the upper bound of the number of GE. Then, the range of $\mathcal{V}(K, G)$ is $K_{low} \leq K \leq G_{up}/1.00GE$ and $1.00GE \times K_{low} \leq G \leq G_{up}$.*

Proof. $1.00GE$ represents the lower area cost of the non-linear operation (e.g. NAND). Every implementation of an S-box needs several non-linear operations. Assuming that all K_{low} gates of an implementation are NAND, the area of this implementation must be the smallest one. Thus, the lower bound of G is $1.00GE \times K_{low}$. In the same way, if the number of gates in an implementation exceeds $G_{up}/1.00GE$, its area must be greater than G_{up}. □

Finally, we propose Algorithm 2 and utilize the Proposition 1 to find a circuit implementing an S-box with the smallest number of GE.

3.4 Bit-sliced technique

Bit-sliced techniques are widely used in the implementation and optimization of cryptographic primitives [2, 5, 6, 10, 17]. We transplanted the idea of bit-sliced into our model and provide a natural way to optimally encode the relation between inputs and outputs of the S-boxes.

As can be seen from Example 1, our model needs 2^n sets of equations to encode each input x and output $S(x)$ for an n-bit S-box. Although the coefficient variables a and b of each set of equations are the same, which determine the implementation circuit, more intermediate variables q and t are needed. To reduce the number of variables and then speed up our model, we use the bit-sliced technique as follows.

Algorithm 1: Solve the sub-problem: whether a circuit can implement an S-box uses determined (K) logic operations with (G) GE.

Input: K : Number of gates
G : Target area cost
$Sbox[]$: an n-bit to n-bit S-box

Output: If the sub-problem has a solution, it returns "1" and the implementation of this S-box or other case returns "0".

//Encode this sub-problem as an SAT-model with equtions.
$Counter_q \leftarrow 2K \cdot 2^n$
$Counter_t \leftarrow K \cdot 2^n$
$Counter_a \leftarrow 2 \times (n + (n + K - 2)) \times K/2 + n^2 + n \cdot K$
$Counter_b \leftarrow 4K$
$Cost[2^4] \leftarrow$ each area cost of operations in Table 1
for $x \leftarrow 0$ *to* $2^n - 1$ **do**
  $x = x_0||x_1||...||x_{n-1}$;
  $y = S(x) = y_0||y_1||...||y_{n-1}$;
  for $i \leftarrow 0$ *to* $K - 1$ **do**
    $q_{2i} \leftarrow$ one of S-box's inputs or outputs of previous gates;
    $q_{2i+1} \leftarrow$ one of S-box's inputs or outputs of previous gates;
    $q_{2i+2} \leftarrow$ one of S-box's inputs or outputs of previous gates;
    $q_{2i+3} \leftarrow$ one of S-box's inputs or outputs of previous gates;
    $t_i = ...$;
  end
  for $i \leftarrow 0$ *to* $n - 1$ **do**
    $y_i \leftarrow$ only one of S-box inputs or outputs of previous gates t;
  end
end
$total_{cost} \leftarrow$ sum of all the gates' area cost;
//Here is the end of the model.
if *Solve the model by STP, it returns "No Solution"* **then**
  return *0;*
end
else
  return *1 and the implementations of this S-box;*
end

Example 2. We give RECTANGLE's S-box and its corresponding truth table in Table 9.

Firstly, we re-encode every variables as a 16-bit Boolean vectorial variables instead of Boolean variables. For example, we use $x_0, x_1, \ldots, x_{63}$ to encode the inputs of the S-box and $y_0, y_1, \ldots, y_{63}$ to encode the outpus of the S-box in our original model. We re-encode them and only use 8 variables as

$$X_0 = \text{0x00ff},\ X_1 = \text{0x0f0f},\ X_2 = \text{0x3333},\ X_3 = \text{0x5555};$$
$$Y_0 = \text{0x369c},\ Y_1 = \text{0xe616},\ Y_2 = \text{0x96c5},\ Y_3 = \text{0x4bb4};$$

Algorithm 2: find the implementation of an S-box with the smallest number of GE.

Input: K_{low} : Gates' number of the optimal gate complexity implementation.
G_{up} : Total area cost of the optimal gate complexity implementation.
$Sbox[]$: an n-bit to n-bit S-box.

Output: The optimal GEC implementation and its area cost.

$K_{up} \leftarrow G_{up}$
$G_{low} \leftarrow K_{low}$
for $K \leftarrow K_{low}$ *to* K_{up} **do**
    for $G \leftarrow G_{up}$ *to* G_{low} **do**
        if *call the Algorithm 1 return 0 with the input (K,G,S)* **then**
            $G_{up} = G + 1$
            $K_{up} = G_{up}$
            break;
        end
    end
end
return (K_{up}, G_{up})

Table 9. Truth table of RECTANGLE S-box.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Hex
$S(x)$	6	5	12	10	1	14	7	9	11	0	3	13	8	15	4	2	-
x_0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0x00ff
x_1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0x0f0f
x_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0x3333
x_3	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0x5555
y_0	0	0	1	1	0	1	0	1	1	0	0	1	1	1	0	0	0x369c
y_1	1	1	1	0	0	1	1	0	0	0	0	1	0	1	1	0	0xe616
y_2	1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	1	0x96c5
y_3	0	1	0	0	1	0	1	1	1	0	1	1	0	1	0	0	0x4bb4

Then, we also use 16-bit vectorial Boolean variables A_i, Q_i, B_i and T_i to re-encode the Boolean variables a_i, q_i, b_i and t_i.

$$
\begin{aligned}
Q_0 =& A_0 \cdot X_0 + A_1 \cdot X_1 + A_2 \cdot X_2 + A_3 \cdot X_3 \\
Q_1 =& A_4 \cdot X_0 + A_5 \cdot X_1 + A_6 \cdot X_2 + A_7 \cdot X_3 \\
Q_2 =& A_8 \cdot X_0 + A_9 \cdot X_1 + A_{10} \cdot X_2 + A_{11} \cdot X_3 \\
Q_3 =& A_{12} \cdot X_0 + A_{13} \cdot X_1 + A_{14} \cdot X_2 + A_{15} \cdot X_3
\end{aligned}
$$

$$
\begin{aligned}
T_0 = & B_0 \cdot Q_0 \cdot Q_1 \cdot Q_2 \cdot Q_3 + Q_0 \cdot Q_0 \cdot Q_1 \cdot Q_2 + \\
& B_0 \cdot Q_0 \cdot Q_1 \cdot Q_3 + B_0 \cdot Q_2 \cdot Q_3 + B_0 \cdot Q_2 + B_0 \cdot Q_3 + \\
& B_1 \cdot Q_0 \cdot Q_1 \cdot Q_2 + B_2 \cdot Q_0 \cdot Q_1 + B_2 \cdot Q_0 \cdot Q_2 + B_2 \cdot Q_1 \cdot Q_2 + \\
& B_2 \cdot Q_0 + B_2 \cdot Q_1 + B_2 \cdot Q_2 + B_3 \cdot Q_0 + B_3 \cdot Q_1 + B_3 \cdot Q_2 + \\
& B_4 \cdot Q_0 \cdot Q_1 + B_5 \cdot Q_0 + B_5 \cdot Q_1 + B_6 \cdot Q_0 + B_7. \\
\ldots
\end{aligned}
$$

In this set of equations, we add more constraints on coefficient variables A_i and B_i as follows

$$
\begin{aligned}
A_i \in & 0x0000 \ , \ 0x1111, \\
B_i \in & 0x0000 \ , \ 0x1111.
\end{aligned}
$$

In conclusion, we only need 1 set of equations to encode RECTANGLE's S-box instead of 2^4 sets of equations in our original model above. The bit-sliced technique would immediately reduce the number of Q_i, T_i, X_i, and Y_i by a factor of 2^n and speed up the search. For more details about our model before and after the re-encoding, please refer to the code which are available online at https://github.com/Zhenyulu-cyber/Sample_implementation.

4 Applications to lightweight S-boxes

We now give our results related to small S-boxes. Our goal is to find the smallest circuits implementing those S-boxes with respect to the overall area. All of our experiments are running on AMD EPYC 7302 CPU 3.0Hz with 8-core. We use our tool and provide the details on the implementation of RECTANGLE's S-box in Table 10. In addition, some implementations of 4-bit and 5-bit S-boxes from well-known ciphers, such as PICCOLO, SKINNY, LBLOCK, KECCAK, and ASCON, are listed in Appendix A.

To highlight the strength of our tool, we compare our results with previous works in [9] and [15] under the UMC 180nm library which is a technology used in [9]. In Table 11, it can be seen that all of our results are better than Stoffelen's and this is expected as Stoffelen's tool simply minimizes the number of gate. Meanwhile, we find a circuit of RECTANGLE's S-box with 18.00GE cost which is better than `LIGHTER`'s and we can verify that the circuits of PICCOLO, SKINNY and LBLOCK's S-boxes have the optimal area cost under the 2-, 3- and 4-input gates we considered. In addition, due to the bit-sliced technique, our model can be used to find the implementation of 5-bit S-box. However, due to the expansion of the search space, we cannot guarantee that the searched implementation of 5-bit S-box is the optimal one.

Moreover, we also use the state-of-the-art synthesis tool Synopsys Design Compiler (DC) to synthesize lookup table (LUT) based implementation and

Table 10. The implementation of RECTANGLE's S-box.

a	b	c	d	**Operations**
$q_0 = x_0$;	$q_1 = x_1$;	$q_2 = 0$;	$q_3 = 0$;	$t_0 = NOR(a, b)$;
$q_4 = x_3$;	$q_5 = t_0$;	$q_6 = x_3$;	$q_7 = t_0$;	$t_1 = MOAI1(a, b, c, d)$;
$q_8 = x_2$;	$q_9 = t_1$;	$q_{10} = 0$;	$q_{11} = 0$;	$t_2 = NOR(a, b)$;
$q_{12} = x_0$;	$q_{13} = t_2$;	$q_{14} = x_0$;	$q_{15} = t_2$;	$t_3 = MOAI1(a, b, c, d)$;
$q_{16} = x_1$;	$q_{17} = t_3$;	$q_{18} = x_1$;	$q_{19} = t_3$;	$t_4 = MOAI1(a, b, c, d)$;
$q_{20} = x_1$;	$q_{21} = x_2$;	$q_{22} = x_1$;	$q_{23} = x_2$;	$t_5 = MOAI1(a, b, c, d)$;
$q_{24} = t_1$;	$q_{25} = t_5$;	$q_{26} = 0$;	$q_{27} = 0$;	$t_6 = AND(a, b)$
$q_{28} = t_5$;	$q_{29} = t_1$;	$q_{30} = t_5$;	$q_{31} = t_1$;	$t_7 = MOAI1(a, b, c, d)$;
$q_{32} = t_4$;	$q_{33} = t_7$;	$q_{34} = 0$;	$q_{35} = 0$;	$t_8 = NAND(a, b)$;
$q_{36} = t_6$;	$q_{37} = t_3$;	$q_{38} = t_6$;	$q_{39} = t_3$;	$t_9 = MOAI1(a, b, c, d)$;
$q_{40} = t_8$;	$q_{41} = t_1$;	$q_{42} = t_8$;	$q_{43} = t_1$;	$t_{10} = MOAI1(a, b, c, d)$;
$y_0 = t_7$;	$y_1 = t_9$;	$y_2 = t_4$;	$y_3 = t_{10}$;	GEC=18.00GE

equation based implementation circuits from three tools (e.g. ours, Stoffelen's [15] and LIGHTER [9]). We set the compiler being specifically instructed to optimize the circuit for area under the TSMC 90nm library. By comparing the output results of these algorithms, we measure the quality of the synthesis in the setting where area only should be minimized. We list the results in Table 12.

When using the DC synthesizer on the circuits produced by our tool (equation based implementation), the area is much better than the circuits produced by Stoffelen's tool (equation based implementation) and the circuits converted by DC synthesizers from the LUT. Especially the performance on RECTANGLE's S-box, the results from our tool is much better than `LIGHTER`.

Note that the choice of standard cell libraries used is almost irrelevant for our work as we are mainly interested in the quality of the area-optimized synthesis itself.

Table 11. Comparison of area-optimized on the UMC 180nm.

Sbox	`LIGHTER`	**[15]**	**Ours**			
	Area	Area	Area	Gate number	Optimal	Time
PICCOLO	13.00GE	16.66GE	13.00GE	8	$\surd$	1min
SKINNY	13.33GE	16.33GE	13.33GE	8	$\surd$	3min
RECTANGLE	18.33GE	25.66GE	18.00GE	11	-	43min
LBLOCK S_0	16.33GE	23GE	16.33GE	10	$\surd$	12min
KECCAK	-	-	17.66GE	13	-	6.66h
ASCON	-	-	28.66GE	15	-	4.66h

Table 12. Comparison of area-optimized on the TSMC 90nm.

Sbox	**TSMC 90nm Logic Process**			
	DC (from LUT)	**DC (from Ours)**	**DC (from [15])**	**DC (from LIGHTER)**
PICCOLO	18.25GE	11.25GE	11.25GE	11.25GE
SKINNY	23.00GE	11.00GE	11.00GE	11.00GE
RECTANGLE	23.00GE	16.25GE	18.25GE	18.00GE
LBLOCK S_0	17.50GE	14.25GE	14.75GE	14.25GE
KECCAK	17.00GE	16.50GE	-	-
ASCON	27.75GE	27.00GE	-	-

5 Conclusion and Future Work

In this article, we have described a new method to improve the implementation of lightweight cipher S-boxes. Our tool based on SAT-model could search for the optimal area implementation with 2, 3, and 4 inputs gates. It is very practical for cryptographic designers. There are still some weakness and future works that deserve to consider. For example, our tool can only apply to small S-boxes, e.g., 4-bit and 5-bit S-boxes. When the implementation of an S-box is complex, it is difficult to find the optimal implementation. The efficiency of our tool depends heavily on the size and complexity of S-boxes. So, a future work is to reduce the search space and speed up finding the optimal implementation.

Acknowledgements

We thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. This work is supported by the National Natural Science Foundation of China (Grant No. 62032014), the National Key Research and Development Program of China (Grant No. 2018YFA0704702), the Major Scientific and Technological Innovation Project of Shandong Province, China (Grant No. 2019JZZY010133), the Major Basic Research Project of Natural Science Foundation of Shandong Province, China (Grant No. ZR202010220025), the Program of Qilu Young Scholars (Grant No. 61580082063088) of Shandong University, and National Natural Science Foundation of China (Grant No. 62002204).

Appendix A Implementation of Some S-boxes

In this section, we give the implementations of several Sboxes mapped on the UMC 180nm standard cell libraries used in this paper.

Table 13. The implementation of PICCOLO's S-box.

a	b	c	d	**Operations**
$q_0 = x_2$;	$q_1 = x_3$;	$q_2 = 0$;	$q_3 = 0$;	t_0 = OR(a,b);
$q_4 = x_0$;	$q_5 = t_0$;	$q_6 = x_0$;	$q_7 = t_0$;	t_1 = MOAI1(a,b,c,d);
$q_8 = x_1$;	$q_9 = t_1$;	$q_{10} = 0$;	$q_{11} = 0$;	t_2 = NOR(a,b);
$q_{12} = x_1$;	$q_{13} = x_2$;	$q_{14} = 0$;	$q_{15} = 0$;	t_3 = OR(a,b);
$q_{16} = x_2$;	$q_{17} = t_2$;	$q_{18} = x_2$;	$q_{19} = t_2$;	t_4 = MOAI1(a,b,c,d);
$q_{20} = x_3$;	$q_{21} = t_3$;	$q_{22} = x_3$;	$q_{23} = t_3$;	t_5 = MOAI1(a,b,c,d);
$q_{24} = t_1$;	$q_{25} = t_5$;	$q_{26} = 0$;	$q_{27} = 0$;	t_6 = OR(a,b)
$q_{28} = x_1$;	$q_{29} = t_6$;	$q_{30} = x_1$;	$q_{31} = t_6$;	t_7 = MOAI1(a,b,c,d);
$y_0 = t_7$;	$y_1 = t_4$;	$y_2 = t_5$;	$y_3 = t_1$;	GEC=13.00GE

Table 14. The implementation of SKINNY's S-box.

a	b	c	d	**Operations**
$q_0 = x_2$;	$q_1 = x_3$;	$q_2 = 0$;	$q_3 = 0$;	t_0 = OR(a,b);
$q_4 = x_1$;	$q_5 = x_2$;	$q_6 = 0$;	$q_7 = 0$;	t_1 = OR(a,b);
$q_8 = x_3$;	$q_9 = t_1$;	$q_{10} = x_3$;	$q_{11} = t_1$;	t_2 = MOAI1(a,b,c,d);
$q_{12} = x_0$;	$q_{13} = t_0$;	$q_{14} = x_0$;	$q_{15} = t_0$;	t_3 = MOAI1(a,b,c,d);
$q_{16} = x_1$;	$q_{17} = t_3$;	$q_{18} = 0$;	$q_{19} = 0$;	t_4 = OR(a,b);
$q_{20} = t_2$;	$q_{21} = t_3$;	$q_{22} = 0$;	$q_{23} = 0$;	t_5 = OR(a,b);
$q_{24} = x_1$;	$q_{25} = t_5$;	$q_{26} = x_1$;	$q_{27} = t_5$;	t_6 = MOAI1(a,b,c,d);
$q_{28} = x_2$;	$q_{29} = t_4$;	$q_{30} = x_2$;	$q_{31} = t_4$;	t_7 = MOAI1(a,b,c,d);
$y_0 = t_6$;	$y_1 = t_7$;	$y_2 = t_2$;	$y_3 = t_3$;	GEC=13.33GE

Table 15. The implementation of LBLOCK's S-box.

a	b	c	d	**Operations**
$q_0 = x_2$;	$q_1 = x_3$;	$q_2 = 0$;	$q_3 = 0$;	t_0 = OR(a,b);
$q_4 = x_0$;	$q_5 = t_0$;	$q_6 = x_0$;	$q_7 = t_0$;	t_1 = MOAI1(a,b,c,d);
$q_8 = x_1$;	$q_9 = t_1$;	$q_{10} = x_1$;	$q_{11} = t_1$;	t_2 = MOAI1(a,b,c,d);
$q_{12} = x_2$;	$q_{13} = t_2$;	$q_{14} = 0$;	$q_{15} = 0$;	t_3 = NAND(a,b);
$q_{16} = x_0$;	$q_{17} = t_3$;	$q_{18} = x_0$;	$q_{19} = t_3$;	t_4 = MOAI1(a,b,c,d);
$q_{20} = x_3$;	$q_{21} = t_4$;	$q_{22} = x_3$;	$q_{23} = t_4$;	t_5 = MOAI1(a,b,c,d);
$q_{24} = t_2$;	$q_{25} = t_5$;	$q_{26} = 0$;	$q_{27} = 0$;	t_6 = NOR(a,b)
$q_{28} = x_3$;	$q_{29} = t_6$;	$q_{30} = x_3$;	$q_{31} = t_6$;	t_7 = MOAI1(a,b,c,d);
$q_{32} = t_5$;	$q_{33} = t_7$;	$q_{34} = 0$;	$q_{35} = 0$;	t_8 = NAND(a,b);
$q_{36} = x_2$;	$q_{37} = t_8$;	$q_{38} = x_2$;	$q_{39} = t_8$;	t_9 = MOAI1(a,b,c,d);
$y_0 = t_2$;	$y_1 = t_5$;	$y_2 = t_9$;	$y_3 = t_7$;	GEC=16.33GE

Table 16. The implementation of KECCAK's S-box.

a	b	c	d	**Operations**
$q_0 = x_2$;	$q_1 = 0$;	$q_2 = 0$;	$q_3 = 0$;	t_0 = NOT(a);
$q_4 = x_4$;	$q_5 = 0$;	$q_6 = 0$;	$q_7 = 0$;	t_1 = NOT(a);
$q_8 = x_1$;	$q_9 = 0$;	$q_{10} = 0$;	$q_{11} = 0$;	t_2 = NOT(a);
$q_{12} = x_3$;	$q_{13} = t_1$;	$q_{14} = 0$;	$q_{15} = 0$;	t_3 = OR(a,b);
$q_{16} = x_2$;	$q_{17} = t_3$;	$q_{18} = x_2$;	$q_{19} = t_3$;	t_4 = MOAI1(a,b,c,d);
$q_{20} = x_3$;	$q_{21} = t_0$;	$q_{22} = 0$;	$q_{23} = 0$;	t_5 = NAND(a,b);
$q_{24} = x_0$;	$q_{25} = t_2$;	$q_{26} = 0$;	$q_{27} = 0$;	t_6 = OR(a,b)
$q_{28} = x_4$;	$q_{29} = t_6$;	$q_{30} = x_4$;	$q_{31} = t_6$;	t_7 = MOAI1(a,b,c,d);
$q_{32} = x_1$;	$q_{33} = t_5$;	$q_{34} = x_1$;	$q_{35} = t_5$;	t_8 = MOAI1(a,b,c,d);
$q_{36} = x_2$;	$q_{37} = t_2$;	$q_{38} = 0$;	$q_{39} = 0$;	t_9 = NAND(a,b);
$q_{40} = x_0$;	$q_{41} = t_9$;	$q_{42} = x_0$;	$q_{43} = t_9$;	t_{10} = MOAI1(a,b,c,d);
$q_{44} = x_0$;	$q_{45} = t_1$;	$q_{46} = 0$;	$q_{47} = 0$;	t_{11} = NAND(a,b);
$q_{48} = x_3$;	$q_{49} = t_{11}$;	$q_{50} = x_3$;	$q_{51} = t_{11}$;	t_{12} = MOAI1(a,b,c,d);
$y_0 = t_{10}$; $y_1 = t_8$;	$y_2 = t_4$;	$y_3 = t_{12}$;	$y_4 = t_7$;	GEC=17.66GE

References

1. Banik, S., Funabiki, Y., Isobe, T.: More results on shortest linear programs. Cryptology ePrint Archive, Report 2019/856 (2019), https://ia.cr/2019/856
2. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. Lecture Notes in Computer Science,

Table 17. The implementation of ASCON's S-box.

a	b	c	d	**Operations**
$q_0 = x_1$;	$q_1 = x_2$;	$q_2 = x_1$;	$q_3 = x_2$;	t_0 = MOAI1(a,b,c,d);
$q_4 = x_0$;	$q_5 = x_4$;	$q_6 = x_0$;	$q_7 = x_4$;	t_1 = MOAI1(a,b,c,d);
$q_8 = x_1$;	$q_9 = t_0$;	$q_{10} = 0$;	$q_{11} = 0$;	t_2 = NOR(a,b);
$q_{12} = x_3$;	$q_{13} = x_4$;	$q_{14} = x_3$;	$q_{15} = x_4$;	t_3 = MAOI1(a,b,c,d);
$q_{16} = t_1$;	$q_{17} = t_2$;	$q_{18} = t_1$;	$q_{19} = t_2$;	t_4 = MOAI1(a,b,c,d);
$q_{20} = x_3$;	$q_{21} = t_0$;	$q_{22} = 0$;	$q_{23} = 0$;	t_5 = NAND(a,b);
$q_{24} = x_1$;	$q_{25} = t_5$;	$q_{26} = x_1$;	$q_{27} = t_5$;	t_6 = MOAI1(a,b,c,d)
$q_{28} = x_0$;	$q_{29} = t_3$;	$q_{30} = 0$;	$q_{31} = 0$;	t_7 = NOR(a,b);
$q_{32} = x_4$;	$q_{33} = t_3$;	$q_{34} = 0$;	$q_{35} = 0$;	t_8 = AND(a,b);
$q_{36} = t_0$;	$q_{37} = t_8$;	$q_{38} = t_0$;	$q_{39} = t_8$;	t_9 = MAOI1(a,b,c,d);
$q_{40} = x_1$;	$q_{41} = t_4$;	$q_{42} = 0$;	$q_{43} = 0$;	t_{10} = NOR(a,b);
$q_{44} = t_0$;	$q_{45} = t_7$;	$q_{46} = t_0$;	$q_{47} = t_7$;	t_{11} = MAOI1(a,b,c,d);
$q_{48} = t_3$;	$q_{49} = t_{10}$;	$q_{50} = t_3$;	$q_{51} = t_{10}$;	t_{12} = MOAI1(a,b,c,d);
$q_{51} = t_4$;	$q_{52} = t_{12}$;	$q_{53} = t_4$;	$q_{54} = t_{12}$;	t_{13} = MAOI1(a,b,c,d);
$q_{55} = t_4$;	$q_{56} = t_6$;	$q_{57} = t_4$;	$q_{58} = t_6$;	t_{14} = MAOI1(a,b,c,d);
$y_0 = t_{12}$;	$y_1 = t_{14}$;	$y_2 = t_9$;	$y_3 = t_{11}$; $y_4 = t_{13}$;	GEC=28.66GE

vol. 9815, pp. 123–153. Springer (2016). https://doi.org/10.1007/978-3-662-53008-5_5, https://doi.org/10.1007/978-3-662-53008-5_5

3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 313–314. Springer (2013)
4. Courtois, N., Mourouzis, T., Hulme, D.: Exact logic minimization and multiplicative complexity of concrete algebraic and cryptographic circuits
5. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1. 2. Submission to the CAESAR Competition (2016)
6. Goudarzi, D., Jean, J., Kölbl, S., Peyrin, T., Rivain, M., Sasaki, Y., Sim, S.M.: Pyjamask: Block cipher and authenticated encryption with highly efficient masked implementation. IACR Trans. Symmetric Cryptol. **2020**(S1), 31–59 (2020). https://doi.org/10.13154/tosc.v2020.iS1.31-59, https://doi.org/10.13154/tosc.v2020.iS1.31-59
7. Hlavicka, J., Fiser, P.: Boom-a heuristic boolean minimizer. In: IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281). pp. 439–442 (2001). https://doi.org/10.1109/ICCAD.2001.968667
8. Inc., V.S.: 0.18μm vip standard cell library tape out ready, part number: Umcl18g212t3, process:umc logic 0.18 μm generic ii technology: 0.18μm (July 2004)
9. Jean, J., Peyrin, T., Sim, S.M., Tourteaux, J.: Optimizing implementations of lightweight building blocks. IACR Trans. Symmetric Cryptol. **2017**(4), 130–168 (2017). https://doi.org/10.13154/tosc.v2017.i4.130-168, https://doi.org/10.13154/tosc.v2017.i4.130-168
10. Kwon, H., Koleva, B., Schnädelbach, H., Benford, S.: "it's not yet A gift": Understanding digital gifting. In: Lee, C.P., Poltrock, S.E., Barkhuus,

L., Borges, M., Kellogg, W.A. (eds.) Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing, CSCW 2017, Portland, OR, USA, February 25 - March 1, 2017. pp. 2372–2384. ACM (2017). https://doi.org/10.1145/2998181.2998225, https://doi.org/10.1145/2998181.2998225
11. Mourouzis, T.: Optimizations in algebraic and differential cryptanalysis. Ph.D. thesis, UCL (University College London) (2015)
12. NIST.: Submission requirements and evaluation criteria for the lightweight cryptography standardization process (2018), https://csrc.nist.gov/projects/lightweight-cryptography
13. Rudell, R.L.: Multiple-valued logic minimization for pla synthesis. Tech. Rep. UCB/ERL M86/65, EECS Department, University of California, Berkeley (Jun 1986), http://www2.eecs.berkeley.edu/Pubs/TechRpts/1986/734.html
14. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An ultra-lightweight blockcipher. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6917, pp. 342–357. Springer (2011). https://doi.org/10.1007/978-3-642-23951-9_23, https://doi.org/10.1007/978-3-642-23951-9_23
15. Stoffelen, K.: Optimizing s-box implementations for several criteria using SAT solvers. In: Peyrin, T. (ed.) Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9783, pp. 140–160. Springer (2016). https://doi.org/10.1007/978-3-662-52993-5_8, https://doi.org/10.1007/978-3-662-52993-5_8
16. Wu, W., Zhang, L.: Lblock: A lightweight block cipher. In: López, J., Tsudik, G. (eds.) Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6715, pp. 327–344 (2011). https://doi.org/10.1007/978-3-642-21554-4_19, https://doi.org/10.1007/978-3-642-21554-4_19
17. Zhang, W., Bao, Z., Lin, D., Rijmen, V., Yang, B., Verbauwhede, I.: RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. Sci. China Inf. Sci. **58**(12), 1–15 (2015). https://doi.org/10.1007/s11432-015-5459-7, https://doi.org/10.1007/s11432-015-5459-7