Optimizing S-box Implementations Using SAT Solvers: Revisited

Fuxin Zhang[1,2] and Zhenyu Huang[1,2]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{zhangfuxin,huangzhenyu}@iie.ac.cn

Abstract. We propose a new method to encode the problems of optimizing S-box implementations into SAT problems. By considering the inputs and outputs of gates as Boolean functions, the fundamental idea of our method is representing the relationships between these inputs and outputs according to their algebraic normal forms. Based on this method, we present several encoding schemes for optimizing S-box implementations according to various criteria, such as multiplicative complexity, bitslice gate complexity, gate complexity, and circuit depth complexity. The experimental results of these optimization problems show that, compared to the encoding method proposed in FSE 2016, which represents these relationships between Boolean functions by their truth tables, our new encoding method can significantly accelerate the subsequent solving process by 2-100 times for the majority of instances. To further improve the solving efficiency, we propose several strategies to eliminate the redundancy of the derived equation system and break the symmetry of the solution space. We apply our method in the optimizations of the S-boxes used in Ascon, ICEPOLE, PRIMATEs, Keccak/Ketje/Keyak, Joltik/Piccolo, LAC, Minalpher, Prøst, and RECTANGLE. We achieve some new improved implementations and narrow the range of the optimal values for different optimization criteria of these S-boxes.

Keywords: S-box · SAT solver · Circuit optimization · Algebraic normal form.

1 Introduction

As one of the main primitives in symmetric-key cryptography, S-boxes play a pivotal role in providing nonlinearity and confusion. Efficient implementations of S-boxes are of paramount importance in enhancing the overall performance of symmetric-key algorithms. Depending on the requirements of the specific platform, the optimization of S-box implementations involves one or multiple criteria, such as throughput, latency, power consumption, memory usage, and resistance against side-channel attacks.

Courtois et al. were the first to introduce SAT solvers for optimizing S-box implementations [CMH13]. The main idea of their method is encoding the decision problems related to S-box implementations into SAT problems, which are subsequently solved by off-the-shelf SAT solvers. At FSE 2016 [Sto16], Stoffelen generalized this method and proposed specific schemes to encode optimization problems based on various criteria, such as multiplicative complexity, bitslice gate complexity, gate complexity, and circuit depth complexity, into SAT problems. Moreover, some optimized implementations for lightweight S-boxes used in Ascon [DEMS16], ICEPOLE [lMGH+15], PRIMATEs [ABB+], Keccak [BPVA+11]/Ketje [BDP+]/Keyak [BDP+16], Joltik [JNP15]/Piccolo [SIH+11], LAC [ZWW+14], Minalpher [STA+14], Prøst [KLL+14] and RECTANGLE [ZBL+14], were presented in [Sto16]. Stoffelen's method was further extended to search for S-box implementations with the smallest area [LWH+21, FWL+21], as well as the lowest AND-depth

and AND-count jointly [BMD+20].

In addition to the SAT-based method, some heuristic search-based methods have been developed for optimizing S-box implementations [Osv00, BMP13]. In [JPST17], based on a graph-based meet-in-the-middle search algorithm, Jean et al. proposed an automated tool named LIGHTER, which can generate an efficient implementation of a lightweight S-box given a certain set of available instructions and their corresponding costs. Utilizing the implementation model of LIGHTER, a more efficient and versatile platform named PEIGEN was presented in [BGLS19]. In [Ras22], Rasoolzadeh investigated the latency complexity of S-boxes, and proposed algorithms for determining the latency complexity and finding the circuits with the minimum latency complexity of a given S-box.

Compared to the heuristic search-based methods, an advantage of the SAT-based methods is the theoretical guarantee of obtaining optimal results. Specifically, for the problem of implementing an S-box based on an optimization criterion with a value of k, if the SAT solver returns `UNSAT`(unsatisfiable) for $k = k_1$ and returns `SAT`(satisfiable) for $k = k_2$ with $k_2 = k_1 + 1$, then k_2 is the optimal value for this criterion. However, the SAT solver cannot always return the results within a reasonable time (especially for the `UNSAT` instances), hence there is often a gap between k_1 and k_2 for many instances. Our motivation is to accelerate the solving process of the SAT-based method, thus bridging the gap between k_1 and k_2. For this purpose, we focus on modifying the way of encoding the problems of optimizing S-box implementations into SAT problems.

Our contribution. We propose a new method to encode the decision problems associated with the implementations of S-boxes based on different optimization criteria, such as multiplicative complexity, bitslice gate complexity, gate complexity, and circuit depth complexity, into SAT problems. Compared to the encoding method introduced in FSE 2016 [Sto16], our new encoding method can significantly reduce the running time of the subsequent solving process. For the majority of instances originated from the implementations of various 4-bit and 5-bit S-boxes used in Ascon, ICEPOLE, Joltik/Piccolo, Keccak/Ketje/Keyak, LAC, Minalpher, Prøst, and RECTANGLE, our method can accelerate the solving processes by approximately 2-100 times.

To further improve our encoding method, we introduce several strategies aimed at eliminating the redundancy within the derived equation system and breaking the symmetry of the solution space. Experimental results show that these improvements lead to additional acceleration in solving all `UNSAT` instances and the majority of `SAT` instances.

We achieve some new optimized implementations for the S-boxes used in Ascon, ICEPOLE, Joltik/Piccolo, Keccak/Ketje/Keyak, LAC, Minalpher, Prøst, and RECTANGLE. Moreover, we narrow the range of the possible optimal values for different optimization criteria of these S-boxes. Especially, we determine the multiplicative complexity of the S-box used in PRIMATEs, the gate complexities of the S-box and S-box^{-1} used in RECTANGLE, as well as the optimal widths of the low-depth implementations of the S-boxes used in Keccak/Ketje/Keyak, LAC, and RECTANGLE.

2 Preliminaries

2.1 Boolean Functions and Boolean Equations

Let $\mathbb{F}_2$ be the finite field with only two elements $\{0, 1\}$, and $\mathbb{F}_2^n$ denotes the n-dimensional vector space over $\mathbb{F}_2$. A n-variable Boolean function $f(x_1, x_2, \ldots, x_n)$ is a function mapping $\mathbb{F}_2^n$ to $\mathbb{F}_2$. Let B_n be the set of n-variable Boolean functions. A Boolean function $f \in B_n$ can be represented by a truth table, which is a list of all 2^n vectors together with their function values.

A Boolean function can also be represented as an element in the Boolean polynomial ring $\mathbf{R}_2 = \mathbb{F}_2[x_1, x_2, \ldots, x_n]/\langle x_1^2 + x_1, x_2^2 + x_2, \ldots, x_n^2 + x_n\rangle$, and has a unique polynomial

form:

$$f(x_1, x_2, \ldots, x_n) = \sum_{u=\{u_1,u_2,\ldots,u_n\}\in\mathbb{F}_2^n} a_u \prod_{i=1}^{n} x_i^{u_i}.$$

This form is called the Algebraic Normal Form (ANF) of a Boolean function. A Boolean function in form is called a Boolean polynomial. In this paper, when we say a general Boolean polynomial we mean a Boolean polynomial whose 2^n coefficients are all variables, and when we say a specific Boolean polynomial we mean a Boolean polynomial whose 2^n coefficients take specific Boolean values.

A vectorial Boolean function is a map from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$:

$$(x_1, x_2 \ldots, x_n) \to (f_1(x_1, x_2, \ldots, x_n), f_2(x_1, x_2, \ldots, x_n), \ldots, f_m(x_1, x_2, \ldots, x_n)),$$

where each $f_i(x_1, x_2, \ldots, x_n)$ is a Boolean function. An S-box with n-bit input and m-bit output can be represented by a vectorial Boolean function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$.

A (Boolean) circuit is a directed acyclic graph where the inputs and the gates are the nodes, and the edges correspond to the Boolean-valued wires. The most common gates used in a circuit are `XOR`, `AND`, `NOT`, and `OR`. Each of them corresponds to a specific vectorial Boolean function over $\mathbb{F}_2$.

If a circuit implements a given S-box, then by setting the input nodes as variables $\{x_1, x_2, \ldots, x_n\}$, it outputs the vectorial Boolean function corresponding to the S-box. Note that, in this circuit implementation, the input and output of each gate can be described as Boolean functions with variables $\{x_1, x_2, \ldots, x_n\}$, and the coefficients of their ANFs will have some relations. For example, if the gate is from the set $\{$`XOR`, `AND`, `NOT`$\}$, then we have the following relations.

- `XOR`: Suppose $f = \sum_{u_i\in\mathbb{F}_2} a_{u_1,\ldots,u_n} \prod_{i=1}^{n} x_i^{u_i}, g = \sum_{u_i\in\mathbb{F}_2} b_{u_1,\ldots,u_n} \prod_{i=1}^{n} x_i^{u_i}$ are the inputs of an `XOR` gate, and $h = \sum_{u_i\in\mathbb{F}_2} c_{u_1,\ldots,u_n} \prod_{i=1}^{n} x_i^{u_i}$ is the output, then $h = f+g$ and we have:

$$c_{u_1,\ldots,u_n} = a_{u_1,\ldots,u_n} + b_{u_1,\ldots,u_n},\ \forall (u_1, \ldots, u_n) \in \mathbb{F}_2^n. \tag{1}$$

- `AND`: Suppose $f = \sum_{u_i\in\mathbb{F}_2} a_{u_1,\ldots,u_n} \prod_{i=1}^{n} x_i^{u_i}, g = \sum_{v_i\in\mathbb{F}_2} b_{v_1,\ldots,v_n} \prod_{i=1}^{n} x_i^{v_i}$ are the inputs of an `AND` gate, and $h = \sum_{w_i\in\mathbb{F}_2} c_{w_1,\ldots,w_n} \prod_{i=1}^{n} x_i^{w_i}$ is the output, then $h = f\cdot g$ and we have:

$$c_{w_1,\ldots,w_n} = \sum_{\max(u_i,v_i)=w_i, 1\le i\le n} a_{u_1,\ldots,u_n} \cdot b_{v_1,\ldots,v_n},\ \forall (w_1, \ldots, w_n) \in \mathbb{F}_2^n. \tag{2}$$

- `NOT`: If the input of a `NOT` gate is $f = \sum_{u_i\in\mathbb{F}_2} a_{u_1,\ldots,u_n} \prod_{i=1}^{n} x_i^{u_i}$, and the output of the gate is $g = \sum_{u_i\in\mathbb{F}_2} b_{u_1,\ldots,u_n} \prod_{i=1}^{n} x_i^{u_i}$, then $g = f + 1$ and we have:

$$b_{0,\ldots,0} = a_{0,\ldots,0} + 1, \text{and } \ b_{u_1,\ldots,u_n} = a_{u_1,\ldots,u_n},\ \text{if } (u_1, \ldots, u_n) \neq (0, \ldots, 0). \tag{3}$$

More generally, suppose $f_1, f_2, \ldots, f_k$ are some general Boolean polynomials, $g_1, g_2, \ldots, g_s$ some specific Boolean polynomials, and H is either a general or a specific Boolean polynomial. It is easy to see that given an equation

$$H(f_1, f_2, \ldots, f_k, g_1, g_2, \ldots, g_s) = 0,$$

similarly to generating Equation (1), (2) and (3), by comparing the coefficients of different monomials, we can derive some equations with respect to the coefficients of these general Boolean polynomials. In the following paragraphs of this paper, we use

$$H(f_1, f_2, \ldots, f_k, g_1, g_2, \ldots, g_s) \overset{c}{=} 0$$

to denote these coefficient equations generated from the original polynomial equation.

Equations for the at-most-one constraint: Suppose $a_1, a_2, \ldots, a_k$ are some Boolean variables, then the constraint that there is at most one nonzero element in $\{a_1, a_2, \ldots, a_k\}$ can be depicted by the following Boolean polynomial equations:

$$a_i a_j = 0, \text{ for all } 1 \leq i < j \leq k$$

In the followings, we use the notation $\texttt{AtMost}_1(a_1, a_2, \ldots, a_k)$ to denote the above Boolean polynomial equations for the at-most-one constraint.

Equations for the exactly-one constraint: Suppose $a_1, a_2, \ldots, a_k$ are some Boolean variables, then the constraint that there is exactly one nonzero element in $\{a_1, a_2, \ldots, a_k\}$ can be depicted by the following Boolean polynomial equations:

$$\begin{cases} a_1 + a_2 + \cdots + a_k = 1 \\ a_i a_j = 0, \text{ for all } 1 \leq i < j \leq k \end{cases}$$

In the followings, we use the notation $\texttt{Exactly}_1(a_1, a_2, \ldots, a_k)$ to denote the above Boolean polynomial equations for the exactly-one constraint.

2.2 Optimization Criteria

Since an S-box is equivalent to a vectorial Boolean function, the S-box implementation is sometimes mentioned as the Boolean function implementation. The S-box implementation in software and hardware can be optimized according to various criteria. In this paper, we consider the S-box implementations using gates with fan-in at most two, and focus on the optimization criteria addressed in [Sto16]. Here we reference their definitions from [Sto16]:

Multiplicative Complexity (MC). The multiplicative complexity of an S-box is defined as the minimum number of `AND` gates required to implement the S-box using the gates from the set {`XOR`, `AND`, `NOT`}. We shall also call the multiplicative complexity of a given circuit the actual number of `AND` gates involved in the circuit. Reducing the MC is useful in protecting against side-channel attacks, since the cost of the masking for side-channel analysis countermeasure depends on the MC. Moreover, it is important in multi-party computation and fully homomorphic encryption, where linear operations almost come for free [ARS+15].

Bitslice Gate Complexity (BGC). The bitslice gate complexity of an S-box is defined as the minimum number of gates in {`XOR`, `OR`, `AND`, `NOT`} required to implement the S-box. Reducing the BGC leads to efficient bitsliced software implementations. Note that, `NAND`, `NOR`, and `XNOR` are not allowed under the BGC measure, since on the most common CPU architectures, no instruction can immediately compute these gates.

Gate Complexity (GC). The gate complexity of an S-box is defined as the minimum number of logic gates required to implement the S-box. In addition to bitslice gates {`XOR`, `OR`, `AND`, `NOT`}, {`NAND`, `NOR`, `XNOR`} are now permitted. From a low GC implementation, one can obtain an efficient hardware implementation, although the different area costs of different gates still need to be considered.

Circuit Depth Complexity. For a circuit implementing an S-box, its depth is defined as the length of the longest path from the input gate to the output gate. In hardware implementations, circuit depth is highly relevant to the latency of the circuit. Here, we only consider the circuits consisting of gates from the set {`XOR`, `OR`, `AND`, `NOT`, `NAND`, `NOR`, `XNOR`}.

The procedures of using SAT solvers to solve these different optimization problems are similar. Here, we take the problem of finding an implementation with the optimal MC as an example.

- For any fixed k, the problem of determining whether there is a circuit that implements an S-box $\mathcal{S}$ with at most k `AND` gates, which is called the *multiplicative complexity decision problem*, can be converted to a SAT problem. Then this SAT problem can be solved by an off-the-shelf SAT solver. If the solver returns `SAT` (satisfiable) for some k and returns `UNSAT` (unsatisfiable) for $k-1$, then it is proven that k is the minimum number of `AND` gates required to implement $\mathcal{S}$. Moreover, in this case, from a `SAT` result, we can construct a circuit that implements S with k `AND` gates.

Similarly, circuit optimizations based on different criteria correspond to different decision problems. For encoding such a decision problem into a SAT problem, we can first encode it into a system of Boolean polynomial equations, then convert this system into a CNF (Conjunctive Normal Form) expression, which serves as the input of SAT solvers. The conversion process can be accomplished by an ANF-to-CNF converter, such as Bosphorus [CSCM19]. Therefore, we focus on the problem of encoding the decision problems into Boolean equation systems.

3 Two Encoding Methods

We use the following toy example to illustrate our encoding method and its difference with the one proposed in [Sto16].

Example 1. Implementing an S-box : $(x_1, x_2, x_3) \to (x_1x_2 + x_3, x_2x_3 + x_1, x_1x_3 + x_2)$ with gates in $\{\texttt{AND}, \texttt{XOR}, \texttt{NOT}\}$, and using at most two `AND` gates.

First, we present a scheme for encoding the above problem into a Boolean equation system based on our encoding method. We use general Boolean polynomials X_1, X_2, and X_3 to denote the circuit inputs, and use general Boolean polynomials Y_1, Y_2, and Y_3 to denote the circuit outputs. In this case, if a circuit implements this S-box, the following coefficient equations must hold.

$$X_1 \overset{c}{=} x_1,\ X_2 \overset{c}{=} x_2,\ X_3 \overset{c}{=} x_3,\ Y_1 \overset{c}{=} x_1x_2 + x_3,\ Y_2 \overset{c}{=} x_2x_3 + x_1,\ Y_3 \overset{c}{=} x_1x_3 + x_2$$

Suppose the two inputs and one output of the first `AND` gate are denoted by general Boolean polynomials Q_1, Q_2, and T_1 respectively. Then Q_1 and Q_2 should be constructed from the circuit inputs and previous gates outputs by applying some `XOR` and `NOT` gates. Since no `AND` gate was applied before, we can express Q_1 and Q_2 by some affine combinations of the circuit inputs. It means there exist some $a_i \in \mathbb{F}_2$, $i \in \{1, 2, \ldots, 8\}$ such that

$$\begin{aligned} Q_1 &= a_1 + a_2X_1 + a_3X_2 + a_4X_3, \\ Q_2 &= a_5 + a_6X_1 + a_7X_2 + a_8X_3, \\ T_1 &= Q_1 \cdot Q_2. \end{aligned} \tag{4}$$

Now we denote the two inputs and one output of the second `AND` gate by general Boolean polynomial Q_3, Q_4, and T_2 respectively. Similarly to Q_1 and Q_2, Q_3 and Q_4 can be expressed as an affine combination of X_1, X_2, X_3, and T_1. Hence, there are some $a_i \in \mathbb{F}_2$, $i \in \{9, 10, \ldots, 18\}$ such that

$$\begin{aligned} Q_3 &= a_9 + a_{10}X_1 + a_{11}X_2 + a_{12}X_3 + a_{13}T_1, \\ Q_4 &= a_{14} + a_{15}X_1 + a_{16}X_2 + a_{17}X_3 + a_{18}T_1, \\ T_2 &= Q_3 \cdot Q_4. \end{aligned} \tag{5}$$

Similarly, the circuit outputs can be expressed as some affine combinations of X_1, X_2, X_3, T_1, and T_2, thus there are some $a_i \in \mathbb{F}_2$, $i \in \{19, 20, \ldots, 36\}$ such that

$$\begin{aligned} Y_1 &= a_{19} + a_{20}X_1 + a_{21}X_2 + a_{22}X_3 + a_{23}T_1 + a_{24}T_2, \\ Y_2 &= a_{25} + a_{26}X_1 + a_{27}X_2 + a_{28}X_3 + a_{29}T_1 + a_{30}T_2, \\ Y_3 &= a_{31} + a_{32}X_1 + a_{33}X_2 + a_{34}X_3 + a_{35}T_1 + a_{36}T_2. \end{aligned} \tag{6}$$

Then by setting these a_i's and the coefficients of these general Boolean polynomials as unknowns, we can generate the following Boolean equation system:

$$\mathcal{G} = \begin{cases} X_1 \overset{c}{=} x_1,\ X_2 \overset{c}{=} x_2,\ X_3 \overset{c}{=} x_3 \\ Q_1 \overset{c}{=} a_1 + a_2X_1 + a_3X_2 + a_4X_3 \\ Q_2 \overset{c}{=} a_5 + a_6X_1 + a_7X_2 + a_8X_3 \\ Q_3 \overset{c}{=} a_9 + a_{10}X_1 + a_{11}X_2 + a_{12}X_3 + a_{13}T_1 \\ Q_4 \overset{c}{=} a_{14} + a_{15}X_1 + a_{16}X_2 + a_{17}X_3 + a_{18}T_1 \\ T_1 \overset{c}{=} Q_1 \cdot Q_2,\ T_2 \overset{c}{=} Q_3 \cdot Q_4 \\ Y_1 \overset{c}{=} a_{19} + a_{20}X_1 + a_{21}X_2 + a_{22}X_3 + a_{23}T_1 + a_{24}T_2 \\ Y_2 \overset{c}{=} a_{25} + a_{26}X_1 + a_{27}X_2 + a_{28}X_3 + a_{29}T_1 + a_{30}T_2 \\ Y_3 \overset{c}{=} a_{31} + a_{32}X_1 + a_{33}X_2 + a_{34}X_3 + a_{35}T_1 + a_{36}T_2 \\ Y_1 \overset{c}{=} x_1x_2 + x_3,\ Y_2 \overset{c}{=} x_2x_3 + x_1,\ Y_3 \overset{c}{=} x_1x_3 + x_2 \end{cases} \tag{7}$$

Obviously, a solution of this system corresponds to a circuit that implements the given S-box with at most two `AND` gates.[1] In this solution, the values of the coefficient variables determine the ANFs of different wires in the circuit, and from the values of $a_1, a_2, \ldots, a_{36}$, one can easily construct the sub-circuits that can generate these `AND` gate inputs by `XOR` and `NOT` gates. The number of `XOR` and `NOT` gates used to generate these `AND` gate inputs can be optimized by solving the *shortest linear straight-line program* problem as mentioned in [Sto16].

Now, based on the method proposed in [Sto16], we present another scheme to encode the same problem into Boolean equation systems. First, suppose we have a circuit that implements the given S-box with 2 `AND` gates. If the inputs of the circuit take some specific Boolean values $\{x_1^0, x_2^0, x_3^0\}$, then the outputs of the circuit are some fixed Boolean values $\{y_1^0, y_2^0, y_3^0\}$, where $\{x_1^0, x_2^0, x_3^0\}$ and $\{y_1^0, y_2^0, y_3^0\}$ correspond to a row in the truth table of the S-box. In this case, the input and output wires of all gates will also take some specific Boolean values. It is easy to see that if we substitute the variables X_i, Q_i, T_i, and Y_i with these specific values, Equations (4), (5), and (6) hold for some assignments of these a_i's. Note that, if the inputs of the circuit take any other values, then the wires of this circuit will also take some other fixed values. If we substitute the variables X_i, Q_i, T_i, and Y_i with these new values, Equations (4), (5), and (6) still hold for the same assignments of a_i's.

Actually, we can construct a Boolean equation system $\mathcal{F}$ which consists of 2^3 subsystems $\mathcal{F}_i\,(0 \le i \le 2^3 - 1)$ with the same structure, and $\mathcal{F}_i$ is generated by setting the input of the S-box as (i_0, i_1, i_2), where $i_0i_1i_2$ is the binary expression of i. Moreover, we have

[1] If $a_{24}, a_{30}, a_{36} = 0$, then T_2 is not used in the following gates. This means the `AND` gate generating T_2 is redundant, which induces a circuit implementing the S-box with one `AND` gate.

$$\mathcal{F}_i = \begin{cases} x_{i,1} = i_0,\ x_{i,2} = i_1,\ x_{i,3} = i_2 \\ q_{i,1} = a_1 + a_2 x_{i,1} + a_3 x_{i,2} + a_4 x_{i,3} \\ q_{i,2} = a_5 + a_6 x_{i,1} + a_7 x_{i,2} + a_8 x_{i,3} \\ q_{i,3} = a_9 + a_{10} x_{i,1} + a_{11} x_{i,2} + a_{12} x_{i,3} + a_{13} t_{i,1} \\ q_{i,4} = a_{14} + a_{15} x_{i,1} + a_{16} x_{i,2} + a_{17} x_{i,3} + a_{18} t_{i,1} \\ t_{i,1} = q_{i,1} \cdot q_{i,2},\ t_{i,2} = q_{i,3} \cdot q_{i,4} \\ y_{i,1} = a_{19} + a_{20} x_{i,1} + a_{21} x_{i,2} + a_{22} x_{i,3} + a_{23} t_{i,1} + a_{24} t_{i,2} \\ y_{i,2} = a_{25} + a_{26} x_{i,1} + a_{27} x_{i,2} + a_{28} x_{i,3} + a_{29} t_{i,1} + a_{30} t_{i,2} \\ y_{i,3} = a_{31} + a_{32} x_{i,1} + a_{33} x_{i,2} + a_{34} x_{i,3} + a_{35} t_{i,1} + a_{36} t_{i,2} \\ y_{i,1} = i_0 i_1 + i_2,\ y_{i,2} = i_1 i_2 + i_0,\ y_{i,3} = i_0 i_2 + i_1 \end{cases} \tag{8}$$

Apparently, a solution of $\mathcal{F}$ corresponds to a circuit that implements the given S-box with at most two `AND` gates. In this solution, the values of $x_{i,j}, y_{i,j}, q_{i,j}, t_{i,j}$ are equal to the values taken by the corresponding wires when the circuit input is (i_2, i_1, i_0).

It is easy to see that, for these two encoding schemes, their *basic encoding frameworks*, which define the relations among gate inputs, outputs, and a_i's, are the same. The difference is how we generate Boolean equations from these relations. In our method, the equations are generated based on the algebraic expressions of the vectorial Boolean functions corresponding to the gate inputs and outputs. In comparison, in the method proposed in [Sto16], the equations are generated based on the truth tables of these vectorial Boolean functions. Therefore, we call our encoding method the *algebraic expression method*, and call the method proposed in [Sto16] the *truth table method*.

For two encoding schemes, which are obtained by the algebraic expression method and the truth table method respectively, if their encoding frameworks are the same, then the two equation systems generated by them have the same number of variables and the same number of equations. For example, for this toy problem, in the first scheme, each "$\overset{c}{=}$" equation corresponds to 2^3 Boolean equations, since there are 2^3 coefficients for a Boolean polynomial with 3 variables. Moreover, each general Boolean polynomial corresponds to 2^3 Boolean variables. Therefore, the system $\mathcal{G}$ has $12 \cdot 2^3 + 36$ variables and $15 \cdot 2^3$ equations. For the second scheme, there are 2^3 subsystems and each $\mathcal{F}_i$ contains 36 common variables, 12 special variables, and 15 equations. Therefore, $\mathcal{F}$ also has $12 \cdot 2^3 + 36$ variables and $15 \cdot 2^3$ Boolean equations.

Although the numbers of variables and equations for the systems generated by these two schemes are the same, the corresponding reduced equations, which are obtained by applying simple substitutions to eliminate intermediate variables, have different sizes. We observed that, in most cases, the reduced equation system generated by the algebraic expression method is simpler than the one generated by the truth table method. This can be illustrated by the following toy example.

Example 2. Implementing the 2-to-1 Boolean function: $f : (x_1, x_2) \to x_1 x_2 + x_1 + x_2$ with {`AND`,`XOR`,`NOT`} gates, and using 1 `AND` gate.

Based on the algebraic expression method and the truth table method, we can generate two Boolean equation systems, and both of them have 34 variables and 28 equations. For the algebraic expression method, we have to solve the system:

$$\begin{cases} c_1 = 0, c_2 = 1, c_3 = 0, c_4 = 0, c_5 = 0, c_6 = 0, c_7 = 1, c_8 = 0 \\ c_9 = a_1 + a_2 c_1 + a_3 c_5, c_{10} = a_2 c_2 + a_3 c_6, c_{11} = a_2 c_3 + a_3 c_7, c_{12} = a_2 c_4 + a_3 c_8 \\ c_{13} = a_4 + a_5 c_1 + a_6 c_5, c_{14} = a_5 c_2 + a_6 c_6, c_{15} = a_5 c_3 + a_6 c_7, c_{16} = a_5 c_4 + a_6 c_8 \\ c_{17} = c_9 c_{13}, c_{18} = c_{10} c_{14} + c_{10} c_{13} + c_9 c_{14}, c_{19} = c_{11} c_{15} + c_{11} c_{13} + c_9 c_{15} \\ c_{20} = c_9 c_{16} + c_{10} c_{15} + c_{10} c_{16} + c_{11} c_{14} + c_{11} c_{16} + c_{12} c_{13} + c_{12} c_{14} + c_{12} c_{15} + c_{12} c_{16} \\ c_{21} = a_7 + a_8 c_1 + a_9 c_5 + a_{10} c_{17}, c_{22} = a_8 c_2 + a_9 c_6 + a_{10} c_{18} \\ c_{23} = a_8 c_3 + a_9 c_7 + a_{10} c_{19}, c_{24} = a_8 c_4 + a_9 c_8 + a_{10} c_{20} \\ c_{21} = 0, c_{22} = 1, c_{23} = 1, c_{24} = 1. \end{cases}$$

For the truth table method, we have to solve the system:

$$\begin{cases} x_{1,1}=0,\ x_{1,2}=0, \\ q_{1,1}=a_1+a_2x_{1,1}+a_3x_{1,2},\ q_{1,2}=a_4+a_5x_{1,1}+a_6x_{1,2} \\ t_{1,1}=q_{1,1}q_{1,2} \\ y_{1,1}=a_7+a_8x_{1,1}+a_9x_{1,2}+a_{10}t_{1,1}, \\ y_{1,1}=0 \\ x_{2,1}=0,\ x_{2,2}=1, \\ q_{2,1}=a_1+a_2x_{2,1}+a_3x_{2,2},\ q_{2,2}=a_4+a_5x_{2,1}+a_6x_{2,2} \\ t_{2,1}=q_{2,1}q_{2,2} \\ y_{2,1}=a_7+a_8x_{2,1}+a_9x_{2,2}+a_{10}t_{2,1}, \\ y_{2,1}=1 \\ x_{3,1}=1,\ x_{3,2}=0, \\ q_{3,1}=a_1+a_2x_{3,1}+a_3x_{3,2},\ q_{3,2}=a_4+a_5x_{3,1}+a_6x_{3,2} \\ t_{3,1}=q_{3,1}q_{3,2} \\ y_{3,1}=a_7+a_8x_{3,1}+a_9x_{3,2}+a_{10}t_{3,1}, \\ y_{3,1}=1 \\ x_{4,1}=1,\ x_{4,2}=1, \\ q_{4,1}=a_1+a_2x_{4,1}+a_3x_{4,2},\ q_{4,2}=a_4+a_5x_{4,1}+a_6x_{4,2} \\ t_{4,1}=q_{4,1}q_{4,2} \\ y_{4,1}=a_7+a_8x_{4,1}+a_9x_{4,2}+a_{10}t_{4,1}, \\ y_{4,1}=1. \end{cases}$$

After simple substitution, the reduced system for the algebraic expression method is:

$$\begin{cases} a_7+a_{10}a_1a_4=0,\ a_{10}a_2a_6+a_{10}a_3a_5=1, \\ a_8+a_{10}a_2a_5+a_{10}a_2a_4+a_{10}a_1a_5=1, \\ a_9+a_{10}a_3a_6+a_{10}a_3a_4+a_{10}a_1a_6=1. \end{cases}$$

In comparison, the reduced system for the truth table method is:

$$\begin{cases} a_7+a_{10}a_1a_4=0, \\ a_7+a_9+a_{10}a_1a_4+a_{10}a_1a_6+a_{10}a_3a_4+a_{10}a_3a_6=1, \\ a_7+a_8+a_{10}a_1a_4+a_{10}a_1a_5+a_{10}a_2a_4+a_{10}a_2a_5=1, \\ a_7+a_8+a_9+a_{10}a_1a_4+a_{10}a_1a_5+a_{10}a_1a_6+a_{10}a_2a_4 \\ \quad +a_{10}a_2a_5+a_{10}a_2a_6+a_{10}a_3a_4+a_{10}a_3a_5+a_{10}a_3a_6=1. \end{cases}$$

Apparently, the reduced systems generated by the algebraic expression method are simpler. Since the complexity of the reduced systems often determines the difficulty of solving the original system, we conjecture that the system generated by the algebraic expression method can be probably solved faster, and our experimental results shown in Section 4.5 confirm this conjecture.

4 Algebraic Encoding Schemes for Circuit Optimizations

For the sake of simplicity, in the following paragraphs, we call an encoding scheme obtained by the algebraic expression method, an *algebraic expression scheme*, and call an encoding scheme obtained by the truth table method, a *truth table scheme*. According to Section 3, from a truth table scheme, one can easily construct an algebraic expression scheme by using the same encoding framework. Therefore, we can convert the truth table schemes proposed in [Sto16] to algebraic expression schemes. However, we found that there are some minor issues with the basic encoding frameworks proposed in [Sto16], which will lead to incorrect results in certain cases. For this reason, we have slightly modified these flawed encoding frameworks to ensure that the algebraic expression schemes proposed in this section can work well in any case. In the following parts, we will present these

algebraic expression schemes, and show their advantages by comparing the time to solve the equations generated by these schemes and those from [Sto16] for specific problems.

4.1 An Encoding Scheme for MC Optimizations

In this subsection, we present an algebraic expression scheme for MC optimizations. This algebraic expression scheme uses the same encoding framework proposed in [Sto16] with slightly modification, since we observed that there is a flaw in this encoding framework.

Given an S-box $\mathcal{S} : (x_0, \ldots, x_{n-1}) \to (S_0(x_0, \ldots, x_{n-1}), \ldots, S_{m-1}(x_0, \ldots, x_{n-1}))$, our decision problem is determining *whether $\mathcal{S}$ can be implemented with at most k `AND` gates.* We follow the notations used in Section 3:

- X_i is the general Boolean polynomial corresponding to the circuit input;
- Y_i is the general Boolean polynomial corresponding to the circuit output;
- Q_i is the general Boolean polynomial corresponding to the gate input;
- T_i is the general Boolean polynomial corresponding to the gate output;
- a_i is the variable that describes the expressions of Q_i and Y_i w.r.t. X_i and T_i.

Then the Boolean equation system can be generated as follows.

1) $\forall i \in \{0, \ldots, n-1\}, \forall j \in \{0, \ldots, m-1\}$, $X_i \overset{c}{=} x_i$, $Y_j \overset{c}{=} S_j(x_0, \ldots, x_{n-1})$. These equations encode the circuit inputs and outputs.

2) $\forall i \in \{0, \ldots, 2k-1\}$,

$$Q_i \overset{c}{=} a_{l_i} + \left(\sum_{j=0}^{n-1} a_{l_i+j+1} \cdot X_j\right) + \left(\sum_{j=0}^{\lfloor \frac{i}{2} \rfloor - 1} a_{l_i+n+j+1} \cdot T_j\right),$$

where $l_i = i(n+1) + \left\lfloor \frac{i^2-2i+1}{4} \right\rfloor$. These coefficient equations encode that Q_i, the inputs of the `AND` gates, can be expressed as an affine combination of the circuit inputs and previous `AND` gate outputs. Here a_l corresponds to a possible `NOT` gate.

3) $\forall i \in \{0, \ldots, k-1\}$, $T_i \overset{c}{=} Q_{2i} \cdot Q_{2i+1}$. These equations encode the k `AND` gates.

4) $\forall i \in \{0, \ldots, m-1\}$,

$$Y_i \overset{c}{=} a_s + \left(\sum_{j=0}^{n-1} a_{s+j+1} \cdot X_j\right) + \left(\sum_{j=0}^{k-1} a_{s+n+j+1} \cdot T_j\right), \tag{9}$$

where $s_i = 2k(n+1) + k(k-1) + i(n+k+1)$. These equations encode that the S-box outputs can be expressed as an affine combination of the S-box inputs and all `AND` gate outputs.

Remark 1. If we directly use the encoding framework proposed in [Sto16] , Equation (9) should be rewritten as

$$Y_i \overset{c}{=} \left(\sum_{j=0}^{n-1} a_{s+j+1} \cdot X_j\right) + \left(\sum_{j=0}^{k-1} a_{s+n+j+1} \cdot T_j\right). \tag{10}$$

Compared to Equation (9), this equation does not contain a_s, which encodes a possible `NOT` gate. However, we observed that Equation (10) may lead to an incorrect result when the ANF of the S-box contains the constant term 1. For example, the S-box $(x_1, x_2, x_3) \to (x_1x_2 + x_2x_3 + 1, x_2x_3 + x_1x_3, x_1x_2 + x_1x_3)$ has multiplicative complexity 2, since it can be implemented by the following steps:

- $q_1 = x_1 + x_3$; $t_1 = q_1 \cdot x_2 = x_1x_2 + x_2x_3$; $q_2 = x_2 + x_3$; $t_2 = x_1 \cdot q_2 = x_1x_2 + x_1x_3$; $y_1 = t_1 + 1$; $y_2 = t_1 + t_2$; $y_3 = t_2$.

However, if we utilize the algebraic expression scheme based on Equation (10) or the truth table scheme proposed in [Sto16] with setting $k = 2$, the SAT solver will output `UNSAT`. Therefore, to avoid potentially incorrect results, in our proposed scheme we modify the equations for Y_i as Equation (9).

4.2 An Encoding Scheme for BGC optimizaitons

Given an S-box $\mathcal{S} : (x_0, \ldots, x_{n-1}) \rightarrow (S_0(x_0, \ldots, x_{n-1}), \ldots, S_{m-1}(x_0, \ldots, x_{n-1}))$, the BGC decision problem is determining *whether $\mathcal{S}$ can be implemented with at most k bitslice gates.* Let X_i, Y_i, Q_i, and a_i be defined as in the MC decision problem, then we can encode this decision problem into a Boolean equation system as follows.

1) $\forall i \in \{0, \ldots, n-1\}, \forall j \in \{0, \ldots, m-1\}$, $X_i \stackrel{c}{=} x_i$, $Y_j \stackrel{c}{=} S_j(x_0, \ldots, x_{n-1})$. These equations encode the circuit inputs and outputs.

2) $\forall i \in \{0, \ldots, k-1\}$,

$$T_i \stackrel{c}{=} b_{3i} \cdot Q_{2i} \cdot Q_{2i+1} + b_{3i+1} \cdot Q_{2i} + b_{3i+1} \cdot Q_{2i+1} + b_{3i+2} \cdot Q_{2i} + b_{3i+2}.$$

These equations encode the k gates from the set {`AND`, `OR`, `XOR`, `NOT`}, where Boolean variables $b_{3i}, b_{3i+1}, b_{3i+2}$ determine the type of the gate. Table 1 shows the correspondence between the values of $(b_{3i}, b_{3i+1}, b_{3i+2})$ and the types of gates.

3) $\forall i \in \{0, \ldots, k-1\}$, $b_{3i} \cdot b_{3i+2} = 0$, $b_{3i+1} \cdot b_{3i+2} = 0$. These equations prevent $(b_{3i}, b_{3i+1}, b_{3i+2})$ from taking the values $(0, 1, 1)$, $(1, 0, 1)$, $(1, 1, 1)$, ensuring that only `AND`, `OR`, `XOR`, and `NOT` are allowed in this circuit.

4) $\forall i \in \{0, \ldots, 2k-1\}$,

$$Q_i \stackrel{c}{=} \left(\sum_{j=0}^{n-1} a_{l_i+j} \cdot X_j\right) + \left(\sum_{j=0}^{\lfloor \frac{i}{2} \rfloor - 1} a_{l_i+n+j} \cdot T_j\right), \texttt{AtMost}_1(a_{l_i}, \ldots, a_{l_i+n+\lfloor \frac{i}{2} \rfloor - 1}),$$

where $l_i = ni + \left\lfloor \frac{i^2-2i+1}{4} \right\rfloor$. These equations encode that Q_i, the gate input, can be the constant 0, an S-box input, or a previous gate output.

5) $\forall i \in \{0, \ldots, m-1\}$:

$$Y_i \stackrel{c}{=} \left(\sum_{j=0}^{n-1} a_{s_i+j} \cdot X_j\right) + \left(\sum_{j=0}^{k-1} a_{s_i+n+j} \cdot T_j\right), \texttt{AtMost}_1(a_{s_i}, \ldots, a_{s_i+n+k-1}),$$

where $s_i = 2kn + k(k-1) + i(n+k)$. These equations encode that Y_i, the S-box output, can be the constant 0, an S-box inputs, or a gate output.

Table 1: Gate types for different $(b_{3i}, b_{3i+1}, b_{3i+2})$

$(b_{3i}, b_{3i+1}, b_{3i+2})$	Expression for T_i	Gate represented by T_i
(0,0,0)	0	0
(0,0,1)	$Q_{2i} + 1$	$\texttt{NOT}(Q_{2i})$
(0,1,0)	$Q_{2i} + Q_{2i+1}$	$\texttt{XOR}(Q_{2i}, Q_{2i+1})$
(0,1,1)	$Q_{2i+1} + 1$	Prevented
(1,0,0)	$Q_{2i} \cdot Q_{2i+1}$	$\texttt{AND}(Q_{2i}, Q_{2i+1})$
(1,0,1)	$Q_{2i} \cdot Q_{2i+1} + Q_{2i} + 1$	Prevented
(1,1,0)	$Q_{2i} \cdot Q_{2i+1} + Q_{2i} + Q_{2i+1}$	$\texttt{OR}(Q_{2i}, Q_{2i+1})$
(1,1,1)	$Q_{2i} \cdot Q_{2i+1} + Q_{2i+1} + 1$	Prevented

4.3 An Encoding Scheme for GC Optimizations

The encoding scheme for GC optimizations is similar to that for BGC optimizations. There are only two differences:

1) The gate outputting T_i is encoded by a different expression:

$$\forall i \in \{0, \ldots, k-1\},\ T_i \stackrel{c}{=} b_{3i} \cdot Q_{2i} \cdot Q_{2i+1} + b_{3i+1} \cdot Q_{2i} + b_{3i+1} \cdot Q_{2i+1} + b_{3i+2}.$$

These equations encode k gates in {XOR, XNOR, AND, OR, NAND, NOR}, where Boolean variables $b_{3i}, b_{3i+1}, b_{3i+2}$ determine the type of the gate. The correspondence between the values of $(b_{3i}, b_{3i+1}, b_{3i+2})$ and the types of gates is presented in Table 2. Note that, the NOT gate is not in this table, but it can be implemented by other gates with some specific inputs. For example, if $Q_{2i} = Q_{2i+1}$, $\texttt{NAND}(Q_{2i}, Q_{2i+1}) = \texttt{NOT}(Q_{2i})$, and if $Q_{2i+1} = 0$, $\texttt{XNOR}(Q_{2i}, Q_{2i+1}) = \texttt{NOT}(Q_{2i})$. An explanation for the necessity of the NOT gate for GC optimizations can be found in Remark 2.

Table 2: Gate types for different $(b_{3i}, b_{3i+1}, b_{3i+2})$

$(b_{3i}, b_{3i+1}, b_{3i+2})$	Expression for T_i	Gate represented by T_i
(0,0,0)	0	0
(0,0,1)	1	1
(0,1,0)	$Q_{2i} + Q_{2i+1}$	$\texttt{XOR}(Q_{2i}, Q_{2i+1})$
(0,1,1)	$Q_{2i} + Q_{2i+1} + 1$	$\texttt{XNOR}(Q_{2i}, Q_{2i+1})$
(1,0,0)	$Q_{2i} \cdot Q_{2i+1}$	$\texttt{AND}(Q_{2i}, Q_{2i+1})$
(1,0,1)	$Q_{2i} \cdot Q_{2i+1} + 1$	$\texttt{NAND}(Q_{2i}, Q_{2i+1})$
(1,1,0)	$Q_{2i} \cdot Q_{2i+1} + Q_{2i} + Q_{2i+1}$	$\texttt{OR}(Q_{2i}, Q_{2i+1})$
(1,1,1)	$Q_{2i} \cdot Q_{2i+1} + Q_{2i} + Q_{2i+1} + 1$	$\texttt{NOR}(Q_{2i}, Q_{2i+1})$

2) There is no constraint for $b_{3i}, b_{3i+1}, b_{3i+2}$, since all types of gates are allowed.

Remark 2. For some problems, by using NOT gates, one can achieve implementations with lower GC. For example, consider the problem of implementing the following 3-bit S-box: $(x_1, x_2, x_3) \to (x_1x_2 + x_2, x_1x_3 + x_3, x_2x_3)$. It is easy to check that without NOT gates, the optimal implementation has GC 5. In comparison, with NOT gate, the optimal implementation has GC 4: $t_1 = x_1 + 1$, $t_2 = t_1 \cdot x_2$, $t_3 = t_1 \cdot x_3$, $t_4 = x_2 \cdot x_3$, $y_1 = t_2$, $y_2 = t_3$, $y_3 = t_4$. Theoretically, we only need to consider the NOT gates in the first depth layer of a circuit for GC optimization. Actually, if the input of a NOT gate is the output of a previous gate G, we can remove this NOT gate by modifying G. For example if G is an AND gate, then we can change G to be a NAND gate. By this way, without changing the GC of a circuit, one can remove all NOT gates with depth ≥ 2 in this circuit. However, in our scheme for GC optimizations, determining a gate in the first depth layer is costly, hence we suppose any gate in the circuit could be a NOT gate.

4.4 An Encoding Scheme for Depth Optimizations

The decision problem for circuit depth complexity differs slightly from the former ones. We can divide a circuit into different layers based on the depth of each gate. Note that, if only the depth of a circuit is bounded, the number of gates in each layer can be arbitrarily large, and it is not possible to encode a problem with an infinite number of operations into a fixed Boolean equation system. For this reason, the following decision problem was considered in [Sto16]. Given an S-box $\mathcal{S} : (x_0, \ldots, x_{n-1}) \to (S_0(x_0, \ldots, x_{n-1}), \ldots, S_{m-1}(x_0, \ldots, x_{n-1}))$, *determine whether there is a circuit implementing* $\mathcal{S}$ *with depth* $\leq k$*, and using at most* w *gates in each layer.*

In the encoding framework proposed in [Sto16], the following property was deemed to be valid.

- For a fan-in 2 gate in the i-th layer, any of its inputs should be the circuit input or the output of a gate in some layer with depth less than i.

However, based on this property, one may not always obtain a gate with depth i. For example, if the inputs of this gate are the outputs of a gate with depth $i-2$, then this gate has depth $i-1$. Therefore, when encoding under this property, a gate that was originally intended to be in the i-th layer may actually be located in some previous layer. In this case, the number of gates in this previous layer may exceed the width bound w. More precisely, the solver may output a solution that corresponds to a circuit containing a layer with more than w gates. For this reason, we modify this property as follows.

- For a fan-in 2 gate in the i-th layer, its inputs should be the circuit input or the output of a gate in some layer with depth less than i. Moreover, **at least one bit of its inputs should be the output of a gate in the $(i-1)$-th layer.**

Under this property, the Boolean equation system can be generated as follows.

1) $\forall i \in \{0,\ldots,n-1\}, \forall j \in \{0,\ldots,m-1\},\ X_i \overset{c}{=} x_i,\ Y_j \overset{c}{=} S_j(x_0,\ldots,x_{n-1})$. These equations encode the circuit inputs and outputs.

2) $\forall i \in \{0,\ldots,kw-1\}$,

$$T_i \overset{c}{=} b_{3i} \cdot Q_{2i} \cdot Q_{2i+1} + b_{3i+1} \cdot Q_{2i} + b_{3i+1} \cdot Q_{2i+1} + b_{3i+2}.$$

These equations encode the kw gates, with the type of gates being encoded by $(b_{3i}, b_{3i+1}, b_{3i+2})$ as shown in Table 2. Similarly, the `NOT` gate can be implemented by other fan-in 2 gates with some specific inputs.

3) $\forall i \in \{0,\ldots,2w-1\}, Q_i \overset{c}{=} \sum_{j=0}^{n-1} a_{ni+j} \cdot X_j,\ \texttt{AtMost}_1(a_{ni},\ldots,a_{ni+n-1})$. These equations encode that the input of a gate in first layer can be the constant 0 or an S-box input.

4) $\forall i \in \{w,\ldots,kw-1\}$,

$$\begin{aligned} Q_{2i} &\overset{c}{=} \left(\sum_{j=0}^{n-1} a_{l_i+j} \cdot X_j\right) + \left(\sum_{j=0}^{vw-1} a_{l_i+n+j} \cdot T_j\right), \texttt{AtMost}_1(a_{l_i},\ldots,a_{l_i+n+vw-1}), \\ Q_{2i+1} &\overset{c}{=} \left(\sum_{j=0}^{w-1} a_{l_i+n+vw+j} \cdot T_{(v-1)w+j}\right), \texttt{Exactly}_1(a_{l_i+n+vw},\ldots,a_{l_i+n+(v+1)w-1}), \end{aligned}$$

where $v = \left\lfloor \frac{i}{w} \right\rfloor, l_i = i(n+vw+w) + nw - \frac{v^2+v+2}{2}w^2$. Here Q_{2i} and Q_{2i+1} are two inputs of a gate in the $(v+1)$-th layer of the circuit. These equations encode that Q_{2i} can be the constant 0, or an input of the S-box, while Q_{2i+1} can only be the output of a gate in the v-th layer. Note that, if Q_{2i+1} equals zero, this gate may be a `NOT` gate with depth less than $v+1$. Thus, in order to avoid this case, $\{a_{l_i+n+vw}, a_{l_i+n+vw+1},\ldots,a_{l_i+n+(v+1)w-1}\}$ should satisfy the exactly-one constraint.

5) $\forall i \in \{0,\ldots,m-1\}$,

$$Y_i \overset{c}{=} \left(\sum_{j=0}^{n-1} a_{s_i+j} \cdot X_j\right) + \left(\sum_{j=0}^{kw-1} a_{s_i+n+j} \cdot T_j\right), \texttt{AtMost}_1(a_{s_i},\ldots,a_{s_i+n+kw-1}),$$

where $s_i = kw(\frac{kw}{2} + n + \frac{w}{2}) + w(n-w) + i(n+kw)$. These equations encode that Y_i, the S-box outputs, can be the constant 0, an S-box input, or a gate output.

4.5 Algebraic Expression Method v.s. Truth Table Method

In this subsection, we present the experimental comparison between the algebraic expression method and the truth table method by contrasting the time of the solving process. Our experiments focus on optimizing the implementations of the S-boxes considered in [Sto16] for different criteria. These S-boxes are, the S-boxes used in Ascon [DEMS16], ICEPOLE [lMGH+15], PRIMATEs [ABB+], Keccak [BPVA+11]/Ketje [BDP+]/Keyak [BDP+16], Joltik [JNP15]/Piccolo [SIH+11], LAC [ZWW+14], Minalpher [STA+14], Prøst [KLL+14], RECTANGLE [ZBL+14], and their inverses, if they exist and are used in decryption.

Table 3: The time of the solving process for MC and BGC optimizations

Result	S-box	Size	MC			BGC		
			k	Truth	Alge.	k	Truth	Alge.
SAT	Ascon	5×5	5	7.4 s	**1.4 s**	16	8477 s	**1813 s**
	ICEPOLE	5×5	6	31.2 s	**4.8 s**	26	25262 s	**5357 s**
	Keccak	5×5	5	3.7 s	**1.2 s**	13	19.7 s	75.5 s
	PRIMATEs	5×5	7	85.4 s	**3.6 s**	27	#	**5840 s**
	PRIMATEs^{-1}	5×5	9	9234 s	**1750 s**	-	-	-
	Joltik	4×4	4	0.91 s	**0.28 s**	10	41.8 s	**27.9 s**
	Joltik^{-1}	4×4	4	0.72 s	**0.27 s**	10	157 s	**77.9 s**
	LAC	4×4	4	1.0 s	**0.25 s**	11	91.3 s	**66.2 s**
	Minalpher	4×4	5	2.0 s	**0.85 s**	18	#	**5022 s**
	Prøst	4×4	4	1.4 s	**0.25 s**	8	3.9 s	4.4 s
	RECTANGLE	4×4	4	0.53 s	**0.24 s**	12	606 s	**98.7 s**
	RECTANGLE^{-1}	4×4	4	1.8 s	**0.22 s**	12	34.7 s	81.1 s
UNSAT	Ascon	5×5	4	104 s	**0.42 s**	10	647 s	**215 s**
	ICEPOLE	5×5	5	#	**51.4 s**	10	229 s	**132 s**
	Keccak	5×5	4	215 s	**0.45 s**	9	231 s	**72.8 s**
	PRIMATEs	5×5	5	120936 s	**142 s**	10	385 s	**182 s**
	PRIMATEs^{-1}	5×5	6	#	**658 s**	10	436 s	**101 s**
	Joltik	4×4	3	2.3 s	**0.08 s**	9	1076 s	**193 s**
	Joltik^{-1}	4×4	3	2.3 s	**0.07 s**	9	897 s	**205 s**
	LAC	4×4	3	2.4 s	**0.07 s**	9	504 s	**120 s**
	Minalpher	4×4	4	99.8 s	**0.17 s**	9	240 s	**62.6 s**
	Prøst	4×4	3	2.2 s	**0.07 s**	7	4.5 s	**3.9 s**
	RECTANGLE	4×4	3	2.6 s	**0.04 s**	9	500 s	**128 s**
	RECTANGLE^{-1}	4×4	3	2.4 s	**0.06 s**	9	514 s	**148 s**

In our experiments, we chose Bosphorus [CSCM19] as the ANF-to-CNF converter, and Kissat [BF22] as the SAT solver. Since Bosphorus contains a process of simplifying the equations and the entire converting process may take more time than the SAT problem-solving process, the timings presented in this paper are the sum of the running times of Bosphorus and Kissat. Our experimental platforms are a PC with a 3.4GHz Intel i7-6700 CPU and a workshop with a 2.9GHz AMD 3990X CPU. For each optimization problem, timings for different methods were obtained on the same platform, using only one thread.

In Tables 3 and 4, the timings for solving different instances are presented. The instances for the algebraic expression method were generated based on the schemes presented in Section 4. For a fair comparison, the instances for the truth table method should be generated based on the same basic encoding frameworks, hence we modify the code given in [Sto16] accordingly and generated the instances for the truth table method by the modified code.

In these tables, the columns labeled as "Alge." present the timings for solving the equation systems generated by the algebraic expression method, while the columns labeled

Table 4: The time of the solving process for GC and depth optimizations

Result	S-box	GC			Depth		
		k	Truth	Alge.	(k, w)	Truth	Alge.
SAT	Ascon	15	7303 s	**398 s**	(3, 6)	1024 s	**37.1 s**
	ICEPOLE	21	#	**27303 s**	(4, 7)	#	**2057 s**
	Keccak	13	230 s	**21.8 s**	(2, 10)	86.0 s	**5.1 s**
	PRIMATEs	26	116710 s	**34554 s**	(4, 8)	11190 s	**542 s**
	Joltik	8	3.3 s	**3.1 s**	(4, 2)	2.0 s	**1.4 s**
	Joltik^{-1}	8	18.1 s	**8.8 s**	(4, 3)	57.1 s	**24.5 s**
	LAC	10	45.4 s	**20.4 s**	(3, 6)	140 s	**22.5 s**
	Minalpher	16	#	**916 s**	(4, 5)	27031 s	**396 s**
	Prøst	8	4.6 s	**1.8 s**	(4, 3)	22.9 s	**10.9 s**
	RECTANGLE	11	626 s	**76.5 s**	(3, 6)	241 s	**86.9 s**
	RECTANGLE^{-1}	11	283 s	**86.5 s**	(3, 6)	252 s	**61.9 s**
UNSAT	Ascon	8	3496 s	**34.1 s**	(2, 8)	358 s	**6.3 s**
	ICEPOLE	8	57605 s	**221 s**	(3, 10)	1298 s	**505 s**
	Keccak	8	1991 s	**93.9 s**	(2, 7)	52934 s	**1621 s**
	PRIMATEs	8	118807 s	**41.9 s**	(2, 15)	865 s	**43.4 s**
	PRIMATEs^{-1}	8	1215 s	**13.4 s**	(3, 10)	1571 s	**377 s**
	Joltik	7	27.0 s	**13.8 s**	(3, 10)	436 s	**101 s**
	Joltik^{-1}	7	37.8 s	**13.1 s**	(3, 10)	566 s	**136 s**
	LAC	7	52.0 s	**10.9 s**	(2, 10)	127 s	**9.6 s**
	Minalpher	7	797 s	**8.6 s**	(3, 10)	988 s	**233 s**
	Prøst	7	19.1 s	**16.1 s**	(3, 10)	749 s	**132 s**
	RECTANGLE	8	9096 s	**357 s**	(2, 10)	517 s	**8.0 s**
	RECTANGLE^{-1}	8	4658 s	**323 s**	(2, 10)	145 s	**7.1 s**

as "Truth" present the timings for solving the equation systems generated by the truth table method. Here "#" means running over 2 days without output. For MC, BGC, and GC, the columns labeled as "k" list the maximum number of gates allowed in the decision problems we introduced before. For depth, in the column labeled as "(k, w)", the first number is the depth bound, and the second one is the maximum number of gates allowed in each layer.

Based on these experimental results, it is evident that the algebraic expression method accelerates the solving process for all instances except three cases for BGC. Moreover, for the majority of instances, the acceleration rate is approximately 2 to 100 times.

5 Eliminating Redundancy and Breaking Symmetry

In this section, we further improve the efficiency of the solving process by modifying the basic encoding frameworks. We consider two types of techniques: eliminating redundancy and breaking symmetry.

- For eliminating redundancy, we aim to eliminate redundant variables in the derived equations and prevent the occurrence of redundant gates. Here, redundant variables are variables that can be set to some fixed Boolean values (usually 0) without affecting the satisfiability of the equations. In a circuit, redundant gates refer to gates that can be removed directly from the circuit without affecting the outputs of the circuit.
- For breaking symmetry, we aim to break the symmetry of the solution space, which arises from the commutativity of the binary operations corresponding to the fan-in 2 gates.

In the following, we show how to apply such techniques to different optimization problems.

We should notice that for our implementation problems, solving UNSAT instances is much harder than solving SAT instances, since for a SAT instance, there are a lot of solutions, and the SAT solver only needs to return 1 solution. Moreover, the solving time for SAT instances is not very stable, since slightly modifying the equations may vary the solving time. For these reasons, our primary goal is to reduce the solving time for UNSAT instances.

5.1 Improvements for MC Optimizations

Eliminating Redundant Variables. In the scheme proposed in Section 4.1, Q_i and Y_i are encoded as follows:

$$Q_i \overset{c}{=} a_{l_i} + \left(\sum_{j=0}^{n-1} a_{l_i+j+1} \cdot X_j\right) + \left(\sum_{j=0}^{\lfloor \frac{i}{2} \rfloor - 1} a_{l_i+n+j+1} \cdot T_j\right),$$
$$Y_i \overset{c}{=} a_{s_i} + \left(\sum_{j=0}^{n-1} a_{s_i+j+1} \cdot X_j\right) + \left(\sum_{j=0}^{k-1} a_{s_i+n+j+1} \cdot T_j\right).$$

In these equations, a_{l_i} and a_{s_i} encode possible NOT gates. The following proposition demonstrates that if the ANF of the S-box has no constant term, then we can remove a_l and a_s in these equations without affecting the satisfiability of the system. The proof of this proposition can be found in Appendix A.

Proposition 1. *Let $\mathcal{S}' : (x_0, \ldots, x_{n-1}) \to (S'_0(x_0, \ldots, x_{n-1}), \ldots, S'_{m-1}(x_0, \ldots, x_{n-1}))$ be an S-box with MC k, and the ANF of each $S'_i(x_0, \ldots, x_{n-1})$ has no constant term. Then there exists a circuit implementing $\mathcal{S}'$ has MC k and only contains AND and XOR gates.*

For any S-box $\mathcal{S} : (x_0, \ldots, x_{n-1}) \to (S_0(x_0, \ldots, x_{n-1}), \ldots, S_{m-1}(x_0, \ldots, x_{n-1}))$. Suppose $\mathcal{S}'$ is derived from $\mathcal{S}$ by removing the constant term of each S_i. It is easy to see that $\mathcal{S}$ and $\mathcal{S}'$ have the same MC, and based on an implementation of S' one can easily obtain an implementation of S by adding some NOT gates to generate the constant term 1. Therefore, we can equivalently consider the MC decision problem for $\mathcal{S}'$. Then, based on Proposition 1, we can modify the encoding scheme by rewriting the equations for Q_i and Y_i as

$$Q_i \overset{c}{=} \left(\sum_{j=0}^{n-1} a_{l_i+j} \cdot X_j\right) + \left(\sum_{j=0}^{\lfloor \frac{i}{2} \rfloor - 1} a_{l_i+n+j} \cdot T_j\right),$$
$$Y_i \overset{c}{=} \left(\sum_{j=0}^{n-1} a_{s_i+j} \cdot X_j\right) + \left(\sum_{j=0}^{k-1} a_{s_i+n+j} \cdot T_j\right).$$

By this modification, we can eliminate $2k + m$ redundant variables, when the decision problem is for testing k.

Avoiding Redundant Gates. Here we consider the occurrences of redundant AND gates. For our encoding scheme introduced before, two types of redundant AND gates may appear.

1) The first type is the gates having trivial outputs, and we call this type of gates *trivial gates*. If the input of a circuit are variables, the output of a gate should be a specific Boolean polynomial. A trivial output means this specific Boolean polynomial is equal to the constant 0, constant 1, or one of the gate inputs.

2) The second type is the gates having useless outputs, and we call this type of gates *useless gates*. If T_i is the output of an AND gate, we say T_i is useless if its value does not affect the final outputs of the circuit. Specifically, it means T_i is neither an input of any subsequent gate nor an output of the circuit.

By adding some equations corresponding to the constraints that at least one of some a_i's should be nonzero, we can avoid these two kinds of redundant gates[2]. However, our

[2]A specific introduction of how to avoid these redundant gates can be found in Appendix B

experimental results show that these at-least-one constraints will slightly decrease the efficiency of the subsequent solving process. Therefore, for MC optimizations, we choose to keep these two types of redundant gates. But for optimization problems based on other criteria, we observed that avoiding the trivial gates, which could be achieved by adding some much simpler equations, can accelerate the subsequent solving processes, and we will introduce this in the following subsections.

Breaking Symmetry. We consider the symmetry derived from the commutativity of the multiplication operation. In our modified scheme, the two inputs of the AND gate that outputs T_i are encoded as

$$Q_{2i} = \left(\sum_{j=0}^{n-1} a_{l_{2i}+j} \cdot X_j\right) + \left(\sum_{j=0}^{i-1} a_{l_{2i}+n+j} \cdot T_j\right),$$
$$Q_{2i+1} = \left(\sum_{j=0}^{n-1} a_{l_{2i}+n+i+j} \cdot X_j\right) + \left(\sum_{j=0}^{i-1} a_{l_{2i}+2n+i+j} \cdot T_j\right).$$

Let $N = n+i$ and $l = l_{2i}$. Suppose $\mathcal{A} = \{(\alpha_0, \ldots, \alpha_{N-1}), (\beta_0, \ldots, \beta_{N-1})\}$ is an assignment for variables $a_l, \ldots, a_{l+N-1}$, and $a_{l+N}, \ldots, a_{l+2N-1}$, with $(\alpha_0, \ldots, \alpha_{N-1}) \neq (\beta_0, \ldots, \beta_{N-1})$. Let $\mathcal{A}' = \{(\beta_0, \ldots, \beta_{N-1}), (\alpha_0, \ldots, \alpha_{N-1})\}$ be another assignment, then we have $\mathcal{A} \neq \mathcal{A}'$. Obviously, $\mathcal{A}$ and $\mathcal{A}'$ will result in the same T_i, since $Q_{2i} \cdot Q_{2i+1} = Q_{2i+1} \cdot Q_{2i}$. In this case, we say $\mathcal{A}$ and $\mathcal{A}'$ are a pair of symmetric assignments. It is easy to check that there are $2^{N-1}(2^N - 1)$ pairs of symmetric assignments, and symmetry breaking aims to exclude one assignment from each of these $2^{N-1}(2^N - 1)$ pairs of symmetric assignments.

We can completely break this symmetry by imposing the constraint $(a_l, \ldots, a_{l+N-1}) \succeq (a_{l+N}, \ldots, a_{l+2N-1})$, where "$\succeq$" is a total order, such as the lexicography order "$\succeq_{lex}$", for N-dimension Boolean vectors. Then, an assignment $\mathcal{A}' = \{(\beta_0, \ldots, \beta_{N-1}), (\alpha_0, \ldots, \alpha_{N-1})\}$ with $(\beta_0, \ldots, \beta_{N-1}) \preceq (\alpha_0, \ldots, \alpha_{N-1})$ will be an invalid assignment. Hence, in the subsequent solving process, the searching branches containing this partial assignment are pruned early. Theoretically, this early pruning may reduce the running time of the SAT solver, if its computational cost is lower than that of determining the invalidity of assignments containing $\mathcal{A}'$ in the original system.

If we use the lexicography order, then we can encode the constraint $(a_l, \ldots, a_{l+N-1}) \succeq_{lex} (a_{l+N}, \ldots, a_{l+2N-1})$ to the following equations.

- $(a_l + 1)a_{l+N} = 0$, which encodes $a_l \geq a_{l+N}$.
- For all $0 \leq j \leq N-2$, $\prod_{i=0}^{j}(a_{l+i} + a_{l+N+i} + 1)(a_{l+j+1} + 1)a_{l+N+j+1} = 0$, which encodes: if $a_l = a_{l+N}, \ldots, a_{l+j} = a_{l+N+j}$, then $a_{l+j+1} \geq a_{l+N+j+1}$.

By adding all these equations into the equation system, we can exclude one assignment from each of the $2^{N-1}(2^N - 1)$ pairs of symmetric assignments. However, since there are some complicated ones in these equations, if all these equations are added, the size of the system will significantly increase, which will undesirably slow down the subsequent solving process. To address this issue, we consider eliminating a part of these symmetric assignments by just adding some simple equations.

In these equations, the first equation $(a_l + 1)a_{l+N} = 0$ can exclude assignments with the form $\{(0, *, \ldots, *), (1, *, \ldots, *)\}$. There are $2^{2(N-1)}$ assignments having this form, and each of them contains in a pair of symmetric assignments. These pairs account for $1/(2 - \frac{1}{2^{N-1}}) > 1/2$ of all pairs of symmetric assignments. Hence, by adding this equation for each pair of Q_{2i} and Q_{2i+1}, we can eliminate more than $1/2$ of the symmetric assignments we aim to eliminate. Based on our experiments, we observed that compared to the strategy of adding more equations to exclude more symmetric assignments, this strategy provides better acceleration.

Experimental Results. By applying the above strategies for eliminating redundant variables and breaking partial symmetry, we achieve an improved encoding scheme. In

Table 5, we compare this improved scheme and the basic scheme based on the time to solve the UNSAT instances in Table 3. Experimental results show that the improved encoding scheme can accelerate the solving process for all these UNSAT instances. For instances that are not considered "too easy" (with solving time exceeding 0.5 seconds), the acceleration rate is approximately 2.

Table 5: Comparison of the basic scheme and the improved scheme for MC optimizations

UNSAT instances							
S-box	k	Basic	Improved	S-box	k	Basic	Improved
Ascon	4	0.42 s	**0.40 s**	Joltik^{-1}	3	0.07 s	**0.04 s**
ICEPOLE	5	51.4 s	**22.7 s**	LAC	3	0.07 s	**0.04 s**
Keccak	4	0.45 s	**0.39 s**	Minalpher	4	0.17 s	**0.16 s**
PRIMATEs	5	142 s	**96.1 s**	Prøst	3	0.07 s	**0.03 s**
PRIMATEs^{-1}	6	658 s	**239 s**	RECTANGLE	3	0.04 s	**0.03 s**
Joltik	3	0.08 s	**0.04 s**	RECTANGLE^{-1}	3	0.06 s	**0.04 s**

5.2 Improvements for BGC Optimizations

Eliminating Redundant Variables. In the encoding scheme proposed in Section 4.2, the following equations encode that the output Y_i is equal to either a circuit input or a gate output.

$$Y_i \overset{c}{=} \left(\sum_{j=0}^{n-1} a_{s_i+j} \cdot X_j\right) + \left(\sum_{j=0}^{k-1} a_{s_i+n+j} \cdot T_j\right),\ \texttt{AtMost}_1(a_{s_i}, \ldots, a_{s_i+n+k-1})$$

Note that, in a practical problem, the ANF of an output of a S-box is always not equal to the ANF of any input of the S-box. Therefore, we can modify these equations as follows,

$$Y_i \overset{c}{=} \sum_{j=0}^{k-1} a_{s_i+j} \cdot T_j,\ \texttt{AtMost}_1(a_{s_i}, \ldots, a_{s_i+k-1}),$$

By this modification, we can eliminate $n \cdot m$ variables.

Avoiding Redundant Gates. Similarly to MC optimizations, for BGC optimizations, the achieved circuits may contain two types of redundant gates: trivial gates and useless gates. To avoid useless gates, as in MC optimizations, we should add at-least-one constraints for certain a_j's, but this will decrease the efficiency of the subsequent solving process as observed in our experiments. Therefore, we focus on avoiding the trivial gates.

In our encoding scheme, the inputs of a fan-in 2 gate that outputs T_i are Q_{2i} and Q_{2i+1}. This gate is trivial, when $Q_{2i} = Q_{2i+1}$, $Q_{2i} = 0$, or $Q_{2i+1} = 0$. The following equations encode Q_{2i} and Q_{2i+1} in our scheme.

$$Q_{2i} \overset{c}{=} \left(\sum_{j=0}^{n-1} a_{l_{2i}+j} \cdot X_j\right) + \left(\sum_{j=0}^{i-1} a_{l_{2i}+n+j} \cdot T_j\right),\ \texttt{AtMost}_1(a_{l_{2i}}, \ldots, a_{l_{2i}+n+i-1}),$$
$$Q_{2i+1} \overset{c}{=} \left(\sum_{j=0}^{n-1} a_{l_{2i+1}+j} \cdot X_j\right) + \left(\sum_{j=0}^{i-1} a_{l_{2i+1}+n+j} \cdot T_j\right),\ \texttt{AtMost}_1(a_{l_{2i+1}}, \ldots, a_{l_{2i+1}+n+i-1}),$$

Thus, to ensure $Q_{2i} \neq 0$ and $Q_{2i+1} \neq 0$, we only need to replace $\texttt{AtMost}_1$ with $\texttt{Exactly}_1$ in the above equations. Moreover, when $Q_{2i} \neq 0$, to ensure $Q_{2i} \neq Q_{2i+1}$, we can impose the constrain that for any $0 \leq j \leq n+i-1$, $a_{l_{2i}+j}$ and $a_{l_{2i+1}+j}$ cannot be 1 simultaneously. For this purpose, we can add the following equations into the system.

$$a_{l_{2i}+j} a_{l_{2i+1}+j} = 0, \text{ for all } 0 \leq j \leq n+i-1.$$

Additionally, if the gate outputting T_i is a NOT gate, Q_{2i+1} is a free input. In this case, to guarantee this gate being nontrivial, we only need $Q_{2i} \neq 0$.

Breaking Symmetry. Similarly to MC optimizations, to break the symmetry derived from the commutativity of the binary operation with inputs Q_{2i} and Q_{2i+1}, one can impose the constraint $(a_{l_{2i}}, \ldots, a_{l_{2i}+n+i-1}) \succeq_{lex} (a_{l_{2i+1}}, \ldots, a_{l_{2i+1}+n+i-1})$. We denote this constraint by $Q_{2i} \succeq_{lex} Q_{2i+1}$ for simplicity. Note that, we can combine the constraint $Q_{2i} \succeq_{lex} Q_{2i+1}$ with the constraint $Q_{2i} \neq Q_{2i+1}$ (this prevents the trivial gates with two same inputs) to the constraint $Q_{2i} \succ_{lex} Q_{2i+1}$. When $Q_{2i} \neq 0$, this constraint can be encoded as the following equations.

$$a_{l_{2i}+j} a_{l_{2i+1}} = 0,\ a_{l_{2i}+j} a_{l_{2i+1}+1} = 0,\ \ldots,\ a_{l_{2i}+j} a_{l_{2i+1}+j} = 0,\ 0 \leq \forall j \leq n+i-1.$$

Note that, if $(a_{l_{2i}}, \ldots, a_{l_{2i}+n+i-2}, a_{l_{2i}+n+i-1}) = (0, \ldots, 0, 1)$, the only valid assignment for $(a_{l_{2i+1}}, \ldots, a_{l_{2i+1}+n+i-1})$ in the above equations is $(0, \ldots, 0)$. This contradicts the constraint $Q_{2i+1} \neq 0$. From our experiments, we observed that compared to modifying the constraint $Q_{2i} \succ Q_{2i+1}$, removing the constraint $Q_{2i+1} \neq 0$ is a better choice. Therefore, in our final improved encoding scheme, we impose the constrains $Q_{2i} \succ Q_{2i+1}$ and $Q_{2i} \neq 0$, to avoid the majority of trivial gates and completely break the symmetry derived from commutativity.

Experimental Results. In Table 6, we present the comparison of the performances of our basic encoding scheme and the improved encoding scheme. We can observe that for all `UNSAT` instances, the solving process can be accelerated by approximately 2 to 6 times. Moreover, for most `SAT` instances, the solving process also has a moderate acceleration.

Table 6: Comparison of the basic scheme and the improved scheme for BGC optimizations

S-box	`SAT` instances			`UNSAT` instances		
	k	Basic	Improved	k	Basic	Improved
Ascon	16	1813 s	**299 s**	10	215 s	**118 s**
ICEPOLE	26	5357 s	**2046 s**	10	132 s	**19.2 s**
Keccak	13	75.5 s	**17.8 s**	9	72.8 s	**34.1 s**
PRIMATEs	27	5840 s	**1900 s**	10	182 s	**65.4 s**
PRIMATEs^{-1}	-	-	-	10	101 s	**50.6 s**
Joltik	10	27.9 s	**17.4 s**	9	193 s	**117 s**
Joltik^{-1}	10	77.9 s	84.6 s	9	205 s	**86.8 s**
LAC	11	66.2 s	**65.3 s**	9	120 s	**52.4 s**
Minalpher	18	5022 s	**3397 s**	9	62.6 s	**15.7 s**
Prøst	8	4.4 s	**3.6 s**	7	3.9 s	**2.4 s**
RECTANGLE	12	98.7 s	**62.9 s**	9	128 s	**54.4 s**
RECTANGLE^{-1}	12	81.1 s	104 s	9	148 s	**52.0 s**

5.3 Improvements for GC Optimizations

Since the only difference of the encoding schemes for BGC optimizations and GC optimizations is the way of encoding different gates, we use the strategies introduced in last subsection. Specifically, we apply the following three strategies:

1) Assuming $Y_i \neq X_j$ for any $0 \leq i \leq m-1, 0 \leq j \leq n-1$; 2) Adding the constraint $Q_{2i} \neq 0$; 3) Adding the constraint $Q_{2i} \succ_{lex} Q_{2i+1}$.

Note that, after this modification, Q_{2i+1} can still be equal to 0, thus a `NOT` gate can be applied as $\texttt{XNOR}(Q_{2i}, Q_{2i+1})$ with $Q_{2i+1} = 0$.

Table 7 presents the performances of our basic and improved schemes. It is evident that the improved scheme accelerates the solving process for all `UNSAT` and `SAT` instances.

It is surprising that the acceleration rate reaches approximately 40 to 200 for some UNSAT instances.

Table 7: Comparison of the basic scheme and the improved scheme for GC optimizations

S-box	SAT instances			UNSAT instances		
	k	Basic	Improved	k	Basic	Improved
Ascon	15	398 s	**91.5 s**	8	34.1 s	**11.6 s**
ICEPOLE	21	27303 s	**18631 s**	8	221 s	**1.4 s**
Keccak	13	21.8 s	**19.9 s**	8	93.9 s	**6.5 s**
PRIMATEs	26	34554 s	**4570 s**	8	41.9 s	**1.4 s**
PRIMATEs^{-1}	-	-	-	8	13.4 s	**1.5 s**
Joltik	8	3.1 s	**2.4 s**	7	13.8 s	**3.0 s**
Joltik^{-1}	8	8.8 s	**4.5 s**	7	13.1 s	**2.1 s**
LAC	10	20.4 s	**12.6 s**	7	10.9 s	**1.7 s**
Minalpher	16	916 s	**800 s**	7	8.6 s	**0.42 s**
Prøst	8	1.8 s	**1.7 s**	7	16.1 s	**3.1 s**
RECTANGLE	11	76.5 s	**9.6 s**	8	357 s	**5.5 s**
RECTANGLE^{-1}	11	86.5 s	**10.6 s**	8	323 s	**7.4 s**

5.4 Improvements for Depth Optimizations

Similarly, we apply the following three improving strategies:

1) Assuming $Y_i \neq X_j$ for any $0 \leq i \leq m-1, 0 \leq j \leq n-1$; 2) For the first layer, adding the constraint $Q_{2i+1} \neq 0$; 3) Adding the constraint $Q_{2i} \prec_{lex} Q_{2i+1}$[3].

Note that, after applying these strategies, a NOT can only be implemented by an XNOR gate with inputs 0 and Q_{2i+1}, therefore we cannot impose the constraint $Q_{2i} \neq 0$. Moreover, to be consistent with the property that Q_{2i} can be equal to 0 and $Q_{2i+1} \neq 0$, we use the constraint $Q_{2i} \prec_{lex} Q_{2i+1}$ instead of $Q_{2i} \succ_{lex} Q_{2i+1}$.

The experimental results in Table 8 show that for all UNSAT instances and the majority of SAT instances, the solving process can be accelerated moderately.

Table 8: Comparison of the basic scheme and the improved scheme for depth optimizations

S-box	SAT instances			UNSAT instances		
	(k, w)	Basic	Improved	(k, w)	Basic	Improved
Ascon	(3, 6)	37.1 s	82.1 s	(2, 8)	6.3 s	**3.9 s**
ICEPOLE	(4, 7)	2057 s	**1360 s**	(3, 10)	505 s	**203 s**
Keccak	(2, 10)	5.1 s	**3.6 s**	(2, 7)	1621 s	**42.1 s**
PRIMATEs	(4, 8)	542 s	**274 s**	(2, 15)	43.4 s	**26.2 s**
PRIMATEs^{-1}	-	-	-	(3, 10)	377 s	**341 s**
Joltik	(4, 2)	1.4 s	**1.4 s**	(3, 10)	101 s	**55.8 s**
Joltik^{-1}	(4, 3)	24.5 s	**17.8 s**	(3, 10)	136 s	**92.0 s**
LAC	(3, 6)	22.5 s	**7.9 s**	(2, 10)	9.6 s	**7.6 s**
Minalpher	(4, 5)	396 s	**388 s**	(3, 10)	233 s	**128 s**
Prøst	(4, 3)	10.9 s	**3.9 s**	(3, 10)	132 s	**76.9 s**
RECTANGLE	(3, 6)	86.9 s	**15.9 s**	(2, 10)	8.0 s	**4.3 s**
RECTANGLE^{-1}	(3, 6)	61.9 s	**10.1 s**	(2, 10)	7.1 s	**4.0 s**

[3]Here we compare the last w a_i's in Q_{2i} with the w a_i's in Q_{2i+1}

6 New Results for Optimized S-box Implementations

Based on our improved encoding schemes, we achieved some new results for optimizing the implementations of the S-boxes introduced in Section 4.5. We present these results in this section, and the corresponding detailed implementations are given in Appendix C [4]

Here we denote the range of the optimal value for a criterion (MC, BGC, GC) as $[k_1, k_2]$. For example, let k be the value we test in the MC decision problem. When we say the MC of an S-box is in $[k_1, k_2]$, it means the SAT-solver returns `UNSAT` when $k = k_1 - 1$, and returns `SAT` when $k = k_2$.

Multiplicative Complexity. The MCs of the S-boxes for Ascon, ICEPOLE, Keccak/Ketje/Keyak, Joltik/Piccolo, Joltik^{-1}/Piccolo^{-1}, LAC, Minalpher, Prøst, RECTANGLE, RECTANGLE^{-1} have already been determined in [Sto16]. Considering the issue mentioned in Remark 1, which may cause potentially incorrect results, we verified the correctness of these results by our method. Moreover, as shown in Table 9, we determined the MC of the PRIMATEs S-box and narrowed the value range of the MC of the PRIMATEs^{-1} S-box.

Table 9: Multiplicative complexities of PRIMATEs and PRIMATEs^{-1}

S-box	[Sto16]	This paper
PRIMATEs	$\in [6, 7]$	7
PRIMATEs^{-1}	$\in [6, 10]$	$\in [7, 8]$

Bitslice Gate Complexity. For the S-boxes of Joltik/Piccolo, Joltik^{-1}/Piccolo^{-1}, LAC, and Prøst, their BGCs have been determined in [Sto16]. We verified these results, and obtained some new BGC results for other S-boxes. These new results are summarized in Table 10.

Table 10: Bitslice gate complexities of S-boxes

S-box	[Sto16]	This paper	S-box	[Sto16]	This paper
Ascon	-	$\in [12, 16]$	Keccak	≤ 13	$\in [12, 13]$
ICEPOLE	-	$\in [12, 26]$	Minalpher	$\in [11, \infty]$	$\in [12, 17]$
PRIMATEs	-	$\in [12, 27]$	RECTANGLE	$\in [10, 12]$	$\in [11, 12]$
PRIMATEs^{-1}	-	$\in [12, 45]$			

Gate Complexity. The GCs of the S-boxes for Joltik/Piccolo, Joltik^{-1}/Piccolo^{-1}, LAC, and Prøst have been determined in [Sto16]. Here, we determined the GCs of RECTANGLE and RECTANGLE^{-1}, and obtained some new results for other S-boxes. Table 11 presents these new results.

Table 11: Gate complexities of S-boxes

S-box	[Sto16]	This paper	S-box	[Sto16]	This paper
Ascon	-	$\in [12, 15]$	Keccak	-	$\in [11, 13]$
ICEPOLE	-	$\in [12, 21]$	Minalpher	-	$\in [11, 14]$
PRIMATEs	-	$\in [12, 23]$	RECTANGLE	$\in [10, 11]$	11
PRIMATEs^{-1}	-	$\in [12, 43]$	RECTANGLE^{-1}	$\in [10, 11]$	11

[4] For the BGC and GC optimizations of PRIMATEs^{-1} , the solver cannot return a `SAT` result in a reasonable time. The BGC-45 and GC-43 implementations are constructed from the MC-8 implementation.

Depth complexity. In comparison to the results presented in [Sto16], we achieved new low-depth implementations and `UNSAT` boundaries for the 4-bit Minalpher S-box and all 5-bit S-boxes. We also refined the `UNSAT` boundaries for some 4-bit S-boxes (from LAC, RECTANGLE, RECTANGLE^{-1}). Especially, we can determine the optimal widths for implementing some S-boxes (from Ascon, Keccak, LAC, RECTANGLE, RECTANGLE^{-1}) with low depth.

Table 12: Circuit depth complexities of S-boxes. k is the depth of the circuit and w is the width of the circuit.

S-box	(k, w) for `SAT` results		(k, w) for `UNSAT` bounds		Optimal width
	[Sto16]	This paper	[Sto16]	This paper	
Ascon	-	$(3, 6)$	-	$(3, 5)$	✓
ICEPOLE	-	$(4, 7)$	-	$(4, 4)$	×
PRIMATEs	-	$(4, 7)$	-	$(4, 4)$	×
Keccak	-	$(2, 10)$	-	$(2, 9)$	✓
Minalpher	-	$(4, 5)$	-	$(4, 3)$	×
LAC	$(3, 6)$	$(3, 6)$	$(3, 4)$	**$(3, 5)$**	✓
RECTANGLE	$(3, 6)$	$(3, 6)$	$(3, 4)$	**$(3, 5)$**	✓
RECTANGLE^{-1}	$(3, 6)$	$(3, 6)$	$(3, 4)$	**$(3, 5)$**	✓

7 Conclusion

We revisit the problem of optimizing S-box implementations with SAT solvers. For accelerating the SAT problem-solving process, we propose the algebraic expression method for encoding the circuit optimization problems into SAT problems. Experimental results show that, in most cases, compared to the truth table method introduced in FSE 2016, the algebraic expression method can accelerate the subsequent solving process by approximately 2 to 100 times. To further improve the solving efficiency, we propose several strategies to remove redundant variables, avoid redundant gates, and break the symmetry of the solution space. Based on these further improved encoding schemes, we obtain some new optimized implementations for the S-boxes used in Ascon, ICEPOLE, PRIMATEs, Keccak, Minalpher, and RECTANGLE. We also narrow the value ranges of their MC, BGC, GC, and their optimal circuit width for low-depth implementations. It is easy to see that our method can be easily combined with the SAT-based algorithms for optimizing circuit area and AND-depth proposed in previous works.

References

[ABB+] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. Primates v1. 02. submission to caesar. 2016.

[ARS+15] Martin R Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for mpc and fhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 430–454. Springer, 2015.

[BDP+] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Caesar submission: Ketje v1, march 2014. *URL http://competitions.cr.yp.to/round1/ketjev11.pdf*.

[BDP+16] Guido Bertoni, Joan Daemen, Michaël Peeters, GV Assche, and RV Keer. Keyak v2. *Submission to the CAESAR Competition*, 2016.

[BF22] Armin Biere and Mathias Fleury. Gimsatul, isasat and kissat entering the sat competition 2022. *Proc. of SAT Competition*, pages 10–11, 2022.

[BGLS19] Zhenzhen Bao, Jian Guo, San Ling, and Yu Sasaki. PEIGEN - a platform for evaluation, implementation, and generation of s-boxes. *IACR Trans. Symmetric Cryptol.*, 2019(1):330–394, 2019.

[BMD+20] Begül Bilgin, Lauren De Meyer, Sébastien Duval, Itamar Levi, and François-Xavier Standaert. Low AND depth and efficient inverses: a guide on s-boxes for low-latency masking. *IACR Trans. Symmetric Cryptol.*, 2020(1):144–184, 2020.

[BMP13] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *J. Cryptol.*, 26(2):280–312, 2013.

[BPVA+11] Guido Bertoni, Michaël Peeters, Gilles Van Assche, et al. The keccak reference. 2011.

[CMH13] Nicolas Courtois, Theodosis Mourouzis, and Daniel Hulme. Exact logic minimization and multiplicative complexity of concrete algebraic and cryptographic circuits. *Int. J. Adv. Intell. Syst*, 6(3):165–176, 2013.

[CSCM19] Davin Choo, Mate Soos, Kian Ming A Chai, and Kuldeep S Meel. Bosphorus: Bridging anf and cnf solvers. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 468–473. IEEE, 2019.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1. 2. *Submission to the CAESAR Competition*, 5(6):7, 2016.

[FWL+21] Yanhong Fan, Weijia Wang, Zhihu Li, Zhenyu Lu, Siu-Ming Yiu, and Meiqin Wang. Forced independent optimized implementation of 4-bit s-box. In Joonsang Baek and Sushmita Ruj, editors, *Information Security and Privacy - 26th Australasian Conference, ACISP 2021, Virtual Event, December 1-3, 2021, Proceedings*, volume 13083 of *Lecture Notes in Computer Science*, pages 151–170. Springer, 2021.

[JNP15] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Joltik v1. 3. *CAESAR Round*, 2, 2015.

[JPST17] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing implementations of lightweight building blocks. *IACR Trans. Symmetric Cryptol.*, 2017(4):130–168, 2017.

[KLL+14] Elif Bilge Kavun, Martin M Lauridsen, Gregor Leander, Christian Rechberger, Peter Schwabe, and Tolga Yalçın. Prøst v1. *CAESAR Round*, 1, 2014.

[lMGH+15] Pawe l Morawiecki, Kris Gaj, Ekawat Homsirikamol, Krystian Matusiewicz, Josef Pieprzyk, Marcin Rogawski, Marian Srebrny, and Marcin Wójcik. Icepole v2. *CAESAR submission: http://competitions.cr.yp.to/round2/icepolev2.pdf*, 2015.

[LWH+21] Zhenyu Lu, Weijia Wang, Kai Hu, Yanhong Fan, Lixuan Wu, and Meiqin Wang. Pushing the limits: Searching for implementations with the smallest area for lightweight s-boxes. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings*, volume 13143 of *Lecture Notes in Computer Science*, pages 159–178. Springer, 2021.

[Osv00] Dag Arne Osvik. Speeding up serpent. In *The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, New York, USA*, pages 317–329. National Institute of Standards and Technology,, 2000.

[Ras22] Shahram Rasoolzadeh. Low-latency boolean functions and bijective s-boxes. *IACR Trans. Symmetric Cryptol.*, 2022(3):403–447, 2022.

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: an ultra-lightweight blockcipher. In *International workshop on cryptographic hardware and embedded systems*, pages 342–357. Springer, 2011.

[STA+14] Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, Mitsuru Matsui, and Shoichi Hirose. Minalpher v1. *CAESAR Round*, 1, 2014.

[Sto16] Ko Stoffelen. Optimizing s-box implementations for several criteria using sat solvers. In *International Conference on Fast Software Encryption*, pages 140–160. Springer, 2016.

[ZBL+14] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. Rectangle: a bit-slice ultra-lightweight block cipher. *Suitable for Multiple Platforms*, 2014.

[ZWW+14] Lei Zhang, Wenling Wu, Yanfeng Wang, Shengbao Wu, and Jian Zhang. Lac: A lightweight authenticated encryption cipher. *Submitted to the CAESAR competition*, 2014.

A Proof of Proposition 1

First, we need the following lemma.

Lemma 1. *Let $\mathcal{S} : (x_1, \ldots, x_n) \to (S_1, \ldots, S_m)$ be an S-box with MC k. There exists a circuit, which implements $\mathcal{S}$ with MC k, satisfying that the* `NOT` *gates in this circuit all locate after the last* `AND` *gate.*

Proof. Let $\mathcal{C}_0$ be a circuit that implements $\mathcal{S}$ with k `AND` gates. Denote its outputs by $y_1, y_2, \ldots, y_m$. Suppose $\texttt{AND}_i$ is the i-th `AND` gate in $\mathcal{C}_0$. We denote its inputs as p_i and q_i, and its output as t_i. Then we have

$$p_i = a_i + L_i(x_1, \ldots, x_n, t_1, \ldots, t_{i-1})$$

for some a_i and linear function L_i. If $a_i = 0$, this is a linear combination of these x_j's and t_j's, and for simplicity, we say p_i is a linear combination. In this case, p_i can be generated from x_j's and t_j's by only using `XOR` gates, hence we say `NOT` gate is not used for generating p_i. If $a_1 = 1$, we say p_i is an affine combination. In this case, one should use at least one `NOT` gate to generate p_i from x_j's and t_j's, hence we say `NOT` gate is used for generating p_i The same terminology applies to q_i and y_i.

Now we prove the following assertion: one can remove the `NOT` gates used for generating p_i and q_i by modifying the `XOR` and `NOT` gates following $\texttt{AND}_i$, while preserving the circuit outputs.

If `NOT` gates are used for generating p_i or q_i, we have the following two cases.

1) Both p_i and q_i are affine combinations. We have

$$p_i = p_i' + 1,\ q_i = q_i' + 1,\ t_i = p_i q_i = p_i' q_i' + p_i' + q_i' + 1,$$

where $p_i' = U_i(x_1, \ldots, x_n, t_1, \ldots, t_{i-1})$, $q_i' = V_i(x_1, \ldots, x_n, t_1, \ldots, t_{i-1})$, and U_i, V_i are two linear functions. We can change the inputs of $\texttt{AND}_i$ to p_i' and q_i' by removing the `NOT` gates used to generate p_i and q_i. Then, the output of $\texttt{AND}_i$ will be $t_i' = p_i' q_i'$. Note that in $\mathcal{C}_0$, t_i is used to generate the inputs of some $\texttt{AND}_j$ with $j > i$, or generate the circuit outputs. For any case, t_i is used to generate some affine combination $L(x_1, \ldots, x_n, t_1, \ldots, t_i, \ldots, t_s) + a$. Note that

$$t_i = t_i' + p_i' + q_i' + 1 = t_i' + U_i(x_1, \ldots, x_n, t_1, \ldots, t_{i-1}) + V_i(x_1, \ldots, x_n, t_1, \ldots, t_{i-1}) + 1.$$

By substituting t_i with this expression in L, we have

$$L(x_1, \ldots, x_n, t_1, \ldots, t_i, \ldots, t_s) + a = L'(x_1, \ldots, x_n, t_1, \ldots, t_i', \ldots, t_s) + a' \tag{11}$$

for some a' and linear function L'.

Equation (11) means that we can modify the `XOR` and `NOT` gates following $\texttt{AND}_i$ to preserve the circuit outputs.

2) One of p_i and q_i is an affine combination. Without loss of generality, suppose p_i is an affine combination. Then, we have

$$p_i = p_i' + 1,\ t_i = p_i q_i = p_i' q_i + q_i,$$

where $p_i' = U_i(x_1, \ldots, x_n, t_1, \ldots, t_{i-1})$, $q_i = V_i(x_1, \ldots, x_n, t_1, \ldots, t_{i-1})$, and U_i, V_i are two linear functions.

By removing the `NOT` gate used to generate p_i, we can change the inputs of $\texttt{AND}_i$ to p_i' and q_i. Then the output of $\texttt{AND}_i$ will be $t_i' = p_i' q_i$, and we have

$$t_i = t_i' + V_i(x_1, \ldots, x_n, t_1, \ldots, t_{i-1}).$$

Similarly, for any affine combination $L(x_1, \ldots, x_n, t_1, \ldots, t_i, \ldots, t_s) + a$, we can substitute t_i with the above expression, and obtain an equivalent affine combination $L'(x_1, \ldots, x_n, t_1, \ldots, t'_i, \ldots, t_s) + a'$. This proves our assertion.

Based on the above assertion, we can recursively reconstruct the circuit as follows. For i from 1 to k, remove the `NOT` gates used to generate the inputs of $\texttt{AND}_i$ and modify the following `XOR` and `NOT` gates accordingly. After the above process, the `NOT` gates only appear after $\texttt{AND}_k$. Note that, we only modify the `XOR` and `NOT` gates, hence the new circuit still has k `AND` gates. This proves the lemma. □

Proof of Proposition 1. According to Lemma 1, we can construct a circuit $\mathcal{C}$ that implements $\mathcal{S}$ with multiplicative complexity k, and all `NOT` gates in $\mathcal{C}$ appear after the last `AND` gate. Suppose t_i is the output of the i-th `AND` gate, then the ANF of t_i has no constant term, for any $1 \leq i \leq k$. Since y_i, the i-th bit of the circuit output, can be written as $y_i = L(x_1, \ldots, x_n, t_1, \ldots, t_k) + a_i$, where $a_i \in \mathbb{F}_2$ and L is a linear function. By substituting t_i with its ANF in L, we have $y_i = T(x_1, \ldots, x_n) + a_i$, where T is a Boolean polynomial without a constant term. This induces that $a_i = 0$. Therefore, we can construct y_i from $(x_1, \ldots, x_n, t_1, \ldots, t_k)$ only by `XOR` gates. This proves the proposition.

B Attempts in Eliminating Redundant Gates for MC Optimizations

1) **Trivial gates**. In our modified scheme (without `NOT` gates), the two inputs of an `AND` gate are encoded as

$$Q_{2i} = \left(\sum_{j=0}^{n-1} a_{l_{2i}+j} \cdot X_j\right) + \left(\sum_{j=0}^{i-1} a_{l_{2i}+n+j} \cdot T_j\right),$$
$$Q_{2i+1} = \left(\sum_{j=0}^{n-1} a_{l_{2i+1}+j} \cdot X_j\right) + \left(\sum_{j=0}^{i-1} a_{l_{2i+1}+n+j} \cdot T_j\right).$$

Obviously, if $a_{l_{2i}} = 0, \ldots, a_{l_{2i}+n+i-1} = 0$ or $a_{l_{2i+1}} = 0, \ldots, a_{l_{2i+1}+n+i-1} = 0$, which means $Q_{2i} = 0$ or $Q_{2i+1} = 0$, this `AND` gate will be a trivial gate. Moreover, if $a_{l_{2i}} = a_{l_{2i+1}}, \ldots, a_{l_{2i}+n+i-1} = a_{l_{2i+1}+n+i-1}$, which means $Q_{2i} = Q_{2i+1}$, this `AND` gate will also be a trivial gate.

2) **Useless gates.** For our encoding scheme, if T_i is useless, then it is irrelevant to any Q_r $(r > 2i + 1)$ and any Y_t $(0 \leq t \leq m - 1)$. In our scheme, Q_r and Y_t are encoded as follows,

$$Q_r = \left(\sum_{j=0}^{n-1} a_{l_r+j} \cdot X_j\right) + \left(\sum_{j=0}^{\lfloor \frac{r}{2} \rfloor - 1} a_{l_r+n+j} \cdot T_j\right),$$
$$Y_t = \left(\sum_{j=0}^{n-1} a_{s_t+j} \cdot X_j\right) + \left(\sum_{j=0}^{k-1} a_{s_t+n+j} \cdot T_j\right).$$

If T_i is irrelevant to Q_r and Y_t, we have $a_{l_r+n+i} = a_{s_t+n+i} = 0$.

We use $\texttt{AtLeast}_1(S)$ to denote the equations corresponding to the constraint that there is at least one nonzero element in a variable set S. Then, to avoid trivial gates, we can impose the constraints $Q_{2i} \neq 0$, $Q_{2i+1} \neq 0$, and $Q_{2i} \neq Q_{2i+1}$. This can be done by adding $\texttt{AtLeast}_1(a_{l_{2i}}, \ldots, a_{l_{2i}+n+i-1})$, $\texttt{AtLeast}_1(a_{l_{2i+1}}, \ldots, a_{l_{2i+1}+n+i-1})$, $\texttt{AtLeast}_1(a_l + a_{l+n+i}, \ldots, a_{l+n+i-1} + a_{l+2n+2i-1})$, for all $0 \leq i \leq k - 1$, into the equation system. Moreover, we can add, $\texttt{AtLeast}_1(a_{l_{2i+2}+n+i}, \ldots, a_{l_{2k-1}+n+i}, a_{s_0+n+i}, \ldots, a_{s_{k-1}+n+i})$ for all $0 \leq i \leq k - 1$, into the equation system, to avoid the useless gates.

However, our experiments show that adding either of these two groups of equations will slightly decrease the efficiency of the subsequent solving process[5]. Therefore, for MC optimizations, we choose to keep these two types of redundant gates.

[5]We have tested two ways of encoding $\texttt{AtLeastOne}(a_1, a_2, \ldots, a_s)$: 1) directly adding $a_1 \vee a_2 \vee \cdots \vee a_s$ into the input of the SAT solver; 2) adding $(a_1 + 1)(a_2 + 1) \cdots (a_s + 1) = 0$ into the equation system.

C New Optimized S-box Implementations

Here we present some new optimized S-box implementations achieved by our method. In these implementations, *logical connectives* are used to denote different gates, i.e., let $\wedge$, $\vee$, $\oplus$, $\neg$ denote `AND`, `OR`, `XOR`, `NOT`, respectively. Moreover, we use x_0 and y_0 to denote the most significant bit of the S-box input x and the S-box output y, respectively.

C.1 MC Optimized Implementation

PRIMATEs^{-1}
$k = 8$

$$
\begin{aligned}
q_1 &= x_0 \oplus x_1 \\
t_0 &= x_1 \wedge q_1 \\
u_0 &= x_4 \oplus t_0 \\
u_1 &= x_2 \oplus t_0 \\
u_2 &= \neg x_1 \\
q_2 &= x_1 \oplus u_0 \\
q_3 &= u_1 \oplus u_2 \\
t_1 &= q_2 \wedge q_3 \\
u_3 &= x_3 \oplus q_3 \\
u_4 &= x_2 \oplus q_1 \\
q_4 &= u_3 \oplus u_4 \\
q_5 &= x_4 \oplus u_3 \\
t_2 &= q_4 \wedge q_5 \\
u_5 &= x_4 \oplus t_2 \\
q_6 &= u_5 \oplus u_2 \\
q_7 &= x_1 \oplus q_5 \\
t_3 &= q_6 \wedge q_7 \\
u_6 &= x_2 \oplus q_7 \\
q_8 &= t_3 \oplus u_6 \\
u_7 &= q_6 \oplus q_8 \\
q_9 &= q_4 \oplus u_7 \\
t_4 &= q_8 \wedge q_9 \\
u_8 &= x_3 \oplus t_1 \\
q_{10} &= u_8 \oplus u_1 \\
q_{11} &= t_4 \oplus u_8 \\
t_5 &= q_{10} \wedge q_{11} \\
u_9 &= x_2 \oplus t_2 \\
q_{14} &= t_5 \oplus u_9 \\
q_{15} &= t_4 \oplus u_5 \\
u_{10} &= t_0 \oplus q_{15} \\
u_{11} &= u_8 \oplus u_{10} \\
q_{12} &= \neg u_{11} \\
u_{12} &= q_{15} \oplus q_{14} \\
q_{13} &= u_4 \oplus u_{12} \\
t_6 &= q_{12} \wedge q_{13} \\
t_7 &= q_{14} \wedge q_{15} \\
u_{13} &= t_7 \oplus u_9 \\
u_{14} &= t_6 \oplus u_7 \\
u_{15} &= u_3 \oplus q_8 \\
u_{16} &= u_{14} \oplus u_{15} \\
y_0 &= u_{13} \oplus u_{15} \\
y_1 &= u_8 \oplus u_{13} \\
y_2 &= u_{10} \oplus u_{16} \\
y_4 &= q_{13} \oplus u_{14} \\
y_3 &= y_4 \oplus u_{11}
\end{aligned}
$$

C.2 BGC Optimized Implementations

Ascon

$k = 16$

$$
\begin{aligned}
t_0 &= x_0 \oplus x_4 \\
t_1 &= \neg x_4 \\
t_2 &= t_1 \vee x_3 \\
t_3 &= x_1 \oplus x_2 \\
t_4 &= x_3 \oplus x_2 \\
t_5 &= x_3 \oplus x_4 \\
t_6 &= t_0 \vee x_1 \\
t_7 &= x_0 \vee t_5 \\
t_8 &= t_4 \vee t_3 \\
y_1 &= t_0 \oplus t_8 \\
y_3 &= t_3 \oplus t_7 \\
t_{11} &= x_2 \wedge t_3 \\
t_{12} &= t_6 \oplus t_5 \\
y_2 &= t_3 \oplus t_2 \\
y_0 &= t_{12} \oplus t_{11} \\
y_4 &= t_0 \oplus t_{12}
\end{aligned}
$$

ICEPOLE

$k = 26$

$$
\begin{aligned}
t_0 &= x_4 \wedge x_3 \\
t_1 &= x_1 \vee x_0 \\
t_2 &= x_4 \oplus x_1 \\
t_3 &= \neg x_1 \\
t_4 &= x_4 \wedge x_0 \\
t_5 &= x_3 \oplus t_2 \\
t_6 &= x_2 \oplus x_0 \\
t_7 &= \neg x_3 \\
t_8 &= x_1 \oplus t_4 \\
t_9 &= x_4 \oplus x_0 \\
t_{10} &= x_0 \oplus t_3 \\
t_{11} &= x_3 \wedge x_2 \\
t_{12} &= t_{11} \oplus x_0 \\
t_{13} &= x_2 \vee x_1 \\
t_{14} &= t_0 \oplus t_6 \\
t_{15} &= t_7 \vee t_{12} \\
t_{16} &= t_2 \oplus t_{13} \\
t_{17} &= t_{15} \oplus t_{13} \\
t_{18} &= t_{17} \wedge t_{10} \\
t_{19} &= t_{18} \vee t_9 \\
t_{20} &= t_5 \oplus t_{19} \\
y_2 &= t_{14} \oplus t_{19} \\
y_3 &= t_{20} \oplus t_8 \\
y_1 &= t_{12} \oplus t_{20} \\
y_0 &= t_{16} \oplus t_{19} \\
y_4 &= t_{19} \oplus t_1
\end{aligned}
$$

PRIMATEs

$k = 27$

$$
\begin{aligned}
t_0 &= x_2 \wedge x_3 \\
t_1 &= x_0 \wedge x_2 \\
t_2 &= x_0 \wedge x_3 \\
t_3 &= t_0 \oplus x_4 \\
t_4 &= x_1 \oplus t_1 \\
t_5 &= x_0 \wedge x_4 \\
t_6 &= x_0 \oplus x_3 \\
t_7 &= t_6 \wedge x_4 \\
t_8 &= x_1 \vee x_0 \\
t_9 &= t_1 \oplus t_2 \\
t_{10} &= t_5 \oplus t_8 \\
t_{11} &= x_2 \oplus x_3 \\
t_{12} &= x_2 \wedge t_3 \\
t_{13} &= t_4 \wedge x_2 \\
t_{14} &= t_9 \oplus t_3 \\
t_{15} &= t_{11} \oplus t_{14} \\
t_{16} &= t_7 \oplus x_0 \\
t_{17} &= t_{12} \oplus t_{14} \\
t_{18} &= x_1 \wedge t_{11} \\
y_2 &= t_{12} \oplus t_{10} \\
t_{20} &= \neg t_4 \\
t_{21} &= t_{12} \oplus t_{18} \\
y_3 &= t_{13} \oplus t_{16} \\
t_{23} &= x_1 \vee x_4 \\
y_4 &= t_{20} \oplus t_{17} \\
y_0 &= t_{15} \oplus t_{23} \\
y_1 &= t_6 \oplus t_{21}
\end{aligned}
$$

PRIMATEs^{-1}

$k = 45$

See C.1

Minalpher

$k = 17$

$$
\begin{aligned}
t_0 &= \neg x_3 \\
t_1 &= x_0 \vee t_0 \\
t_2 &= t_1 \oplus x_2 \\
t_3 &= t_2 \oplus x_0 \\
t_4 &= t_2 \oplus x_3 \\
t_5 &= t_4 \oplus x_1 \\
t_6 &= t_5 \oplus t_1 \\
t_7 &= x_3 \oplus t_3 \\
t_8 &= t_7 \vee x_1 \\
t_9 &= t_6 \wedge t_4 \\
t_{10} &= t_3 \wedge t_8 \\
y_3 &= t_9 \oplus t_2 \\
y_2 &= t_{10} \vee t_9 \\
t_{13} &= t_6 \wedge x_0 \\
t_{14} &= t_0 \oplus t_{10} \\
y_0 &= t_9 \oplus t_8 \\
y_1 &= t_{14} \vee t_{13}
\end{aligned}
$$

C.3 GC Optimized Implementations

Ascon

$k = 15$

$$
\begin{aligned}
t_0 &= x_4 \oplus x_3 \\
t_1 &= x_2 \oplus x_3 \\
t_2 &= x_1 \oplus x_2 \\
t_3 &= \neg(x_0 \oplus x_4) \\
t_4 &= \neg(t_0 \vee x_0) \\
t_5 &= \neg(t_3 \oplus t_2) \\
t_6 &= t_5 \vee x_1 \\
t_7 &= x_4 \wedge t_0 \\
t_8 &= \neg(x_1 \wedge t_3) \\
t_9 &= t_2 \vee t_1 \\
y_3 &= \neg(t_4 \oplus t_2) \\
y_4 &= \neg(t_8 \oplus t_0) \\
y_0 &= t_6 \oplus t_0 \\
y_1 &= \neg(t_3 \oplus t_9) \\
y_2 &= \neg(t_2 \oplus t_7)
\end{aligned}
$$

ICEPOLE

$k = 21$

$$
\begin{aligned}
t_0 &= \neg(x_4 \oplus x_2) \\
t_1 &= \neg(x_4 \oplus x_1) \\
t_2 &= t_0 \vee x_3 \\
t_3 &= x_0 \wedge x_1 \\
t_4 &= \neg(x_3 \oplus t_3) \\
t_5 &= \neg(x_2 \wedge x_3) \\
t_6 &= \neg(x_0 \oplus x_2) \\
t_7 &= \neg(t_4 \oplus t_5) \\
t_8 &= \neg(x_0 \wedge t_1) \\
t_9 &= \neg(x_1 \vee t_6) \\
t_{10} &= t_0 \wedge t_6 \\
t_{11} &= \neg(t_1 \wedge t_{10}) \\
t_{12} &= \neg(t_7 \oplus x_4) \\
t_{13} &= \neg(t_2 \oplus t_7) \\
t_{14} &= t_{11} \wedge t_4 \\
y_2 &= t_{14} \oplus t_{13} \\
t_{16} &= t_{14} \oplus x_3 \\
y_0 &= \neg(t_9 \oplus t_{16}) \\
y_4 &= t_{16} \oplus t_1 \\
y_1 &= t_{12} \oplus y_4 \\
y_3 &= t_{14} \oplus t_8
\end{aligned}
$$

Keccak/Ketje/Keyak

$k = 13$

$$
\begin{aligned}
t_0 &= x_3 \wedge x_4 \\
t_1 &= \neg x_2 \\
t_2 &= \neg x_0 \\
t_3 &= \neg(x_1 \wedge t_2) \\
t_4 &= \neg(t_2 \vee x_4) \\
t_5 &= \neg(t_1 \oplus t_0) \\
t_6 &= \neg(t_1 \wedge x_3) \\
y_1 &= \neg(x_1 \oplus t_6) \\
t_8 &= \neg(t_1 \vee x_1) \\
y_2 &= x_4 \oplus t_5 \\
y_4 &= \neg(x_4 \oplus t_3) \\
y_0 &= t_8 \oplus x_0 \\
y_3 &= t_4 \oplus x_3
\end{aligned}
$$

PRIMATEs

$k = 23$

$$
\begin{aligned}
t_0 &= x_3 \oplus x_2 \\
t_1 &= \neg(x_4 \oplus x_3) \\
t_2 &= \neg(x_2 \vee t_1) \\
t_3 &= x_1 \wedge t_0 \\
t_4 &= \neg(x_3 \oplus t_2) \\
t_5 &= x_1 \oplus x_4 \\
t_6 &= x_0 \oplus x_4 \\
t_7 &= \neg(t_5 \vee x_0) \\
t_8 &= x_2 \wedge t_6 \\
t_9 &= t_0 \vee x_3 \\
y_2 &= t_7 \oplus t_4 \\
t_{11} &= t_5 \wedge x_1 \\
t_{12} &= x_0 \wedge t_0 \\
t_{13} &= t_5 \oplus x_0 \\
t_{14} &= t_{12} \oplus t_{11} \\
t_{15} &= t_1 \wedge t_{13} \\
t_{16} &= t_2 \oplus t_3 \\
t_{17} &= \neg(t_8 \oplus t_{12}) \\
t_{18} &= t_{15} \oplus t_{14} \\
y_0 &= t_9 \oplus t_{14} \\
y_1 &= t_6 \oplus t_{16} \\
y_3 &= t_{18} \oplus t_3 \\
y_4 &= t_{17} \oplus t_5
\end{aligned}
$$

PRIMATEs^{-1}

$k = 43$

See C.1

Minalpher

$k = 14$

$$
\begin{aligned}
t_0 &= \neg(x_2 \wedge x_3) \\
t_1 &= x_1 \oplus t_0 \\
t_2 &= t_1 \vee x_0 \\
t_3 &= \neg(x_2 \oplus t_2) \\
t_4 &= \neg(t_3 \oplus x_3) \\
t_5 &= \neg(x_3 \oplus t_1) \\
t_6 &= t_4 \wedge t_5 \\
y_3 &= \neg(t_3 \oplus t_6) \\
t_8 &= t_6 \oplus t_1 \\
y_0 &= x_0 \oplus t_4 \\
t_{10} &= t_8 \wedge y_0 \\
y_2 &= t_5 \oplus t_{10} \\
t_{12} &= \neg(y_2 \wedge y_3) \\
y_1 &= \neg(t_{12} \oplus t_8)
\end{aligned}
$$

C.4 Depth Optimized Implementations

Ascon

$\boldsymbol{k = 3, w = 6}$

Layer 1

$t_0 = x_2 \oplus x_1$
$t_1 = x_1 \oplus x_3$
$t_2 = \neg(x_3 \oplus x_4)$
$t_3 = \neg(x_0 \oplus x_4)$
$t_4 = x_2 \vee x_1$
$t_5 = x_4 \wedge x_3$

Layer 2

$t_6 = t_0 \oplus t_5$
$t_7 = \neg(t_2 \wedge t_3)$
$t_8 = \neg(t_4 \oplus t_2)$
$t_9 = t_1 \vee t_0$
$t_{10} = \neg(x_1 \vee t_3)$
$t_{11} = x_1 \wedge t_3$

Layer 3

$y_0 = t_{10} \oplus t_8$
$y_1 = \neg(t_3 \oplus t_9)$
$y_2 = \neg(x_4 \oplus t_6)$
$y_3 = t_6 \oplus t_7$
$y_4 = \neg(t_2 \oplus t_{11})$

ICEPOLE

$\boldsymbol{k = 4, w = 7}$

Layer 1

$t_0 = \neg(x_4 \oplus x_1)$
$t_1 = \neg(x_4 \wedge x_3)$
$t_2 = x_1 \oplus x_3$
$t_3 = \neg(x_3 \oplus x_0)$
$t_4 = \neg x_1$
$t_5 = \neg(x_2 \oplus x_0)$
$t_6 = x_4 \oplus x_2$

Layer 2

$t_7 = \neg(x_2 \vee t_2)$
$t_8 = t_1 \oplus t_6$
$t_9 = \neg(x_0 \vee t_4)$
$t_{10} = t_2 \vee t_6$
$t_{11} = \neg(t_5 \wedge t_0)$
$t_{12} = x_4 \vee t_3$
$t_{13} = x_2 \wedge t_4$

Layer 3

$t_{15} = \neg(t_{10} \vee t_{11})$
$t_{16} = x_4 \oplus t_9$
$t_{17} = t_{12} \oplus t_{11}$
$t_{18} = \neg(t_1 \oplus t_{12})$
$t_{19} = \neg(x_0 \oplus t_{13})$
$t_{20} = \neg(t_7 \oplus t_{13})$

Layer 4

$y_0 = \neg(t_{15} \oplus t_{19})$
$y_1 = t_{20} \oplus t_{15}$
$y_2 = \neg(t_8 \oplus t_{15})$
$y_3 = \neg(t_{18} \oplus t_{15})$
$y_4 = t_{16} \oplus t_{15}$

Keccak/Ketje/Keyak

$\boldsymbol{k = 2, w = 10}$

Layer 1

$t_0 = \neg(x_1 \vee x_2)$
$t_1 = x_2 \wedge x_3$
$t_2 = x_3 \wedge x_4$
$t_3 = \neg(x_4 \wedge x_0)$
$t_4 = \neg(x_3 \oplus x_1)$
$t_5 = \neg(x_0 \oplus x_1)$
$t_6 = x_2 \oplus x_4$
$t_7 = x_0 \vee x_1$
$t_8 = x_3 \oplus x_0$
$t_9 = \neg(x_4 \oplus x_0)$

Layer 2

$y_0 = t_5 \oplus t_0$
$y_1 = \neg(t_4 \oplus t_1)$
$y_2 = t_6 \oplus t_2$
$y_3 = \neg(t_8 \oplus t_3)$
$y_4 = \neg(t_7 \oplus t_9)$

PRIMATEs

$\boldsymbol{k = 4, w = 7}$

Layer 1

$t_0 = \neg(x_1 \vee x_4)$
$t_1 = x_0 \oplus x_2$
$t_2 = \neg(x_1 \oplus x_4)$
$t_3 = x_1 \wedge x_2$
$t_4 = x_3 \oplus x_0$
$t_5 = x_2 \oplus x_3$
$t_6 = x_4 \oplus x_3$

Layer 2

$t_7 = \neg(x_0 \wedge t_2)$
$t_8 = x_2 \wedge t_1$
$t_9 = \neg(x_3 \wedge t_1)$
$t_{10} = x_2 \wedge t_6$
$t_{11} = \neg(t_6 \oplus t_0)$
$t_{12} = \neg(x_4 \wedge t_4)$
$t_{13} = \neg(x_1 \wedge t_5)$

Layer 3

$t_{14} = x_1 \oplus t_{10}$
$t_{15} = \neg(t_7 \vee t_8)$
$t_{16} = \neg(x_4 \oplus t_9)$
$t_{17} = t_{10} \oplus t_{13}$
$t_{18} = t_1 \oplus t_8$
$t_{19} = t_9 \oplus t_8$
$t_{20} = \neg(t_3 \oplus t_{12})$

Layer 4

$y_0 = \neg(t_{11} \oplus t_{19})$
$y_1 = \neg(t_4 \oplus t_{17})$
$y_2 = t_{14} \oplus t_{15}$
$y_3 = t_{20} \oplus t_{18}$
$y_4 = \neg(t_{14} \oplus t_{16})$

Joltik/Piccolo

$\boldsymbol{k = 4, w = 2}$

Layer 1

$t_0 = \neg(x_1 \vee x_2)$
$t_1 = \neg(x_1 \vee x_0)$

Layer 2

$y_0 = x_3 \oplus t_1$
$y_1 = x_0 \oplus t_0$

Layer 3

$t_4 = x_2 \vee y_0$
$t_5 = y_0 \vee y_1$

Layer 4

$y_2 = x_1 \oplus t_4$
$y_3 = \neg(x_2 \oplus t_5)$

Joltik^{-1}/Piccolo^{-1}

$k = 4, w = 3$

Layer 1

$t_0 = x_0 \vee x_1$
$t_1 = \neg(x_2 \wedge x_1)$
$t_2 = x_0 \oplus x_2$

Layer 2

$t_3 = x_3 \oplus t_2$
$t_4 = t_0 \wedge t_1$
$y_2 = \neg(x_3 \oplus t_0)$

Layer 3

$t_6 = t_2 \vee y_2$
$t_7 = t_3 \wedge y_2$
$t_8 = x_0 \vee y_2$

Layer 4

$y_0 = \neg(x_1 \oplus t_6)$
$y_1 = x_2 \oplus t_8$
$y_3 = t_4 \oplus t_7$

LAC

$k = 3, w = 6$

Layer 1

$t_0 = x_0 \oplus x_3$
$t_1 = \neg(x_3 \vee x_1)$
$t_2 = \neg(x_3 \oplus x_2)$
$t_3 = \neg(x_1 \oplus x_0)$
$t_4 = x_2 \wedge x_1$
$t_5 = x_1 \vee x_0$

Layer 2

$t_6 = \neg(x_0 \vee t_1)$
$t_7 = x_1 \wedge t_2$
$t_8 = \neg(t_0 \wedge t_2)$
$t_9 = t_3 \vee t_4$
$t_{10} = \neg(x_3 \wedge t_5)$
$t_{11} = t_2 \vee t_2$

Layer 3

$y_0 = \neg(t_8 \oplus t_{10})$
$y_1 = t_9 \oplus t_6$
$y_2 = \neg(t_0 \oplus t_7)$
$y_3 = \neg(t_5 \oplus t_{11})$

Minalpher

$k = 4, w = 5$

Layer 1

$t_0 = \neg(x_2 \wedge x_1)$
$t_1 = \neg(x_1 \vee x_2)$
$t_2 = \neg(x_2 \oplus x_3)$
$t_3 = x_1 \wedge x_3$
$t_4 = x_3 \wedge x_2$

Layer 2

$t_5 = x_0 \oplus t_2$
$t_6 = \neg(x_0 \wedge t_0)$
$t_7 = \neg(x_3 \vee t_1)$
$t_8 = x_1 \oplus t_4$
$t_9 = \neg(x_2 \oplus t_3)$

Layer 3

$t_{10} = t_6 \oplus t_9$
$t_{11} = \neg(t_5 \oplus t_8)$
$t_{12} = \neg(x_0 \wedge t_8)$
$t_{13} = \neg(t_5 \vee t_9)$
$t_{14} = t_3 \oplus t_6$

Layer 4

$y_0 = t_{12} \oplus t_{11}$
$y_1 = \neg(t_{13} \oplus t_{14})$
$y_2 = t_{10} \oplus t_{12}$
$y_3 = \neg(t_7 \vee t_{13})$

Prøst

$k = 4, w = 3$

Layer 1

$t_0 = x_0 \wedge x_1$
$t_1 = x_2 \wedge x_1$
$t_2 = \neg(x_3 \wedge x_0)$

Layer 2

$t_3 = x_0 \vee t_1$
$y_0 = x_2 \oplus t_0$
$t_5 = \neg(t_1 \oplus t_2)$

Layer 3

$t_6 = \neg(x_3 \oplus t_5)$
$t_7 = x_1 \oplus t_5$
$t_8 = \neg(x_3 \wedge y_0)$

Layer 4

$y_3 = \neg(t_7 \oplus t_8)$
$y_1 = t_2 \oplus t_6$
$y_2 = \neg(t_3 \oplus t_8)$

RECTANGLE

$k = 3, w = 6$

Layer 1

$t_0 = \neg(x_3 \oplus x_2)$
$t_1 = x_2 \oplus x_1$
$t_2 = x_2 \vee x_0$
$t_3 = \neg(x_1 \oplus x_0)$
$t_4 = x_3 \vee x_1$
$t_5 = \neg(x_0 \oplus x_3)$

Layer 2

$t_6 = \neg(t_2 \oplus t_5)$
$t_7 = \neg(x_1 \vee t_3)$
$t_8 = t_4 \wedge t_0$
$t_9 = \neg(x_2 \vee t_0)$
$t_{10} = t_3 \vee t_5$
$t_{11} = t_1 \wedge t_2$

Layer 3

$y_0 = \neg(t_{10} \oplus t_{11})$
$y_1 = \neg(t_8 \oplus t_7)$
$y_2 = \neg(x_1 \oplus t_6)$
$y_3 = \neg(t_3 \oplus t_9)$

RECTANGLE^{-1}

$k = 3, w = 6$

Layer 1

$t_0 = \neg(x_2 \oplus x_1)$
$t_1 = x_0 \oplus x_3$
$t_2 = \neg(x_2 \wedge x_1)$
$t_3 = \neg(x_2 \wedge x_3)$
$t_4 = x_1 \oplus x_0$
$t_5 = x_2 \vee x_1$

Layer 2

$t_6 = t_5 \wedge t_4$
$t_7 = t_0 \wedge t_4$
$t_8 = \neg(x_0 \vee t_1)$
$t_9 = \neg(t_2 \wedge t_1)$
$t_{10} = \neg(x_3 \wedge t_4)$
$t_{11} = t_3 \wedge t_1$

Layer 3

$y_0 = \neg(t_7 \oplus t_{11})$
$y_1 = t_0 \oplus t_8$
$y_2 = t_0 \oplus t_{10}$
$y_3 = t_6 \oplus t_9$