IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2022, No. 1, pp. 158–182. DOI:10.46586/tosc.v2022.i1.158-182

Towards Low-Latency Implementation of Linear Layers

Qun Liu[1,2], Weijia Wang[1,2], Yanhong Fan[1,2], Lixuan Wu[1,2], Ling Sun[1,2] and Meiqin Wang(✉)[1,2,3]

[1] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China
[2] School of Cyber Science and Technology, Shandong University, Qingdao, China
[3] Quan Cheng Shandong Laboratory, Jinan, China
{qunliu,fanyh,lixuanwu}@mail.sdu.edu.cn, {wjwang,lingsun,mqwang}@sdu.edu.cn

Abstract. Lightweight cryptography features a small footprint and/or low computational complexity. Low-cost implementations of linear layers usually play an important role in lightweight cryptography. Although it has been shown by Boyar et al. that finding the optimal implementation of a linear layer is a Shortest Linear Program (SLP) problem and NP-hard, there exist a variety of heuristic methods to search for near-optimal solutions. This paper considers the low-latency criteria and focuses on the heuristic search of lightweight implementation for linear layers. Most of the prior approach iteratively combines the inputs (of linear layers) to reach the output, which can be regarded as the *forward* search. To better adapt the low-latency criteria, we propose a new framework of *backward* search that attempts to iteratively split every output (into an XORing of two bits) until all inputs appear. By bounding the time of splitting, the new framework can find a sub-optimal solution with a minimized depth of circuits.
We apply our new search algorithm to linear layers of block ciphers and find many low-latency candidates for implementations. Notably, for AES Mixcolumns, we provide an implementation with 103 XOR gates with a depth of 3, which is among the best hardware implementations of the AES linear layer. Besides, we obtain better implementations in XOR gates for 54.3% of 4256 Maximum Distance Separable (MDS) matrices proposed by Li et al. at FSE 2019. We also achieve an involutory MDS matrix (in $\mathbf{M}_4(\mathrm{GL}(8,\mathbb{F}_2))$) whose implementation uses the lowest number (i.e., 86, saving 2 from the state-of-the-art result) of XORs with the minimum depth.

Keywords: Lightweight cryptography · Linear layers · Low latency · AES

1 Introduction

In recent years, lightweight cryptography has been applied in many fields, such as the Internet of Things (IoTs) and Radio-Frequency IDentification (RFID) tags. Their security has been the central area of focus for researchers because various restrictions lead to new security threats [DGB19]. Generally, lightweight cryptography ensures secure encryption and expands cryptography applications to devices with limited resources in circuit size, power consumption, and latency.

There are many criteria for designing lightweight cryptographic primitives, and the most popular one should be the gate equivalents (GE) required to implement a cryptographic algorithm. As it nicely approximates the complexity of digital electronic circuits, there is a growing body of work solely concentrating on decreasing the GE (see [BP10, BMP13, KLSW17, DL18, BFI19, TP19, XZL+20, LXZZ21] for an incomplete list). Meanwhile,

Licensed under Creative Commons License CC-BY 4.0.
Received: 2021-11-23 Accepted: 2022-01-23 Published: 2022-03-11

another criterion called latency is also crucial and has been attracting more and more attention, since it not only impacts the throughput of encryption/decryption, but plays an important role in the low-energy consideration of ciphers [BBI+15].

Generally speaking, research on lightweight cryptography falls in two directions. The first direction focuses on designing new ciphers that suppose to be efficient in either hardware (i.e., by logical gates) or software (i.e., on microprocessors) implementations. The community has devoted a lot to this direction and proposes plenty of structures [BKL+07, BSS+13, BBI+15, ZBL+15, BJK+16, BPP+17, DL18, LMMR21].

The second direction tries to optimize the implementation of given ciphers, which has drawn a lot of attention as well. On the one hand, it is somewhat of more practical significance. For example, the Advanced Encryption Standard (AES) [DR20] has been widely used in practice, and its round function has been frequently used in the design of other cryptographic primitives (e.g., AEGIS [WP13] and ForkAES [ARVV18]); thus, an efficient implementation will directly reduce the cost of deploying AES and the primitives that employ its round function. On the other hand, the optimizing approach can aid the designing of lightweight ciphers. For example, Li et al. applied a heuristic optimization tool to the cost evaluation of the proposed lightweight Maximal Distance Separable (MDS) matrices for linear layers [LSL+19]. This paper follows the second line of work and focuses on the hardware implementation of linear layers that provide diffusion for many cryptographic primitives.

The linear layer of a cryptography cipher can be represented as the multiplication (over $\mathbb{F}_2$) between a matrix and a vector. For an $m \times n$ binary matrix A, given inputs $\vec{x} = (x_0, x_1, ..., x_{n-1})^T$, the outputs $\vec{y} = (y_0, y_1, ..., y_{m-1})^T$ can be calculated by $\vec{y} = A\vec{x}$. We give an example with a matrix A:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix},$$

inputs $\vec{x} = (x_0, x_1, x_2, x_3, x_4)^T$, and the outputs $\vec{y} = (y_0, y_1, y_2)^T$. For the worst case w.r.t. the cost, the implementation can be performed by the procedure described by Figure 1-left, requiring 8 XOR gates. An optimized implementation is given in Figure 1-right, saving half of the number of XOR gates. The above optimization can be formulated as a problem of finding the smallest number of linear operations necessary to compute a set of linear forms, which is called the **Shortest Linear Program (SLP)** problem. Although it has been shown that the SLP problem is NP-hard [BMP08], in practice, we can build circuit implementations of linear layers using a variety of heuristics [Paa97, BP10, BMP13, KLSW17, BFI19, LSL+19, TP19, XZL+20, BFI21, LXZZ21].

$$\begin{aligned} y_0 &= x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 & \qquad y_2 &= x_3 \oplus x_4 \\ y_1 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 & \qquad y_1 &= x_1 \oplus x_2 \oplus y_2 \\ y_2 &= x_3 \oplus x_4 & \qquad y_0 &= x_0 \oplus y_1 \end{aligned}$$

Figure 1: Two implementations of $(y_0, y_1, y_2)^T = A \times (x_0, x_1, x_2, x_3, x_4)^T$

The first heuristic approach employs the strategy originated in Paar's work [Paa97] and can be regarded as the *forward* search. Paar's method encodes input bits by one-hot encoding.[1] Paar optimized the matrix by iteratively combining different columns in the matrix. Paar's method has been improved in a number of follow-up works. Boyar et al. proposed a new algorithm called BP algorithm in [BP10]. It uses a set *Base* to save all the

[1]One-hot encoding encodes each bit into a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0).

values which have been generated by the algorithm. It repeatedly picks two values from $Base$ according to some rules, adds them together as a new value, and puts this new value into $Base$. It further optimizes the procedure of route searching by dedicatedly choosing values to combine. BP algorithm has a series of variants (see [VSP18, RTA18, TP19]). Then, Xiang et al. decomposed the matrix A into the product of several elementary matrices, based on which the search can be significantly improved [XZL+20].

However, the above algorithms cannot solve another problem: how to take circuit depth into account and optimize the matrices with respect to achieving the minimum depth. Li et al. provided a solution by adding a depth constraint in BP algorithm called LSL algorithm [LSL+19], which is further improved by Banik et al. called BFI algorithm [BFI21]. When executing BP algorithm, they give a bound of depth and only pick the choices that are not beyond the bound. In the selection, some choices which have the low priority in their heuristics may lead to a better implementation. We illustrate the case with the following example. Suppose that the objective matrix is M_P:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

As shown in Table 1-left, Li et al. find the implementation by 11 XOR gates with depth 3. Based on their heuristics, y_4 is always generated with depth 3. This disables y_4 from being used in subsequent calculations. Otherwise, it will be beyond the bound of minimum depth. However, we find that y_4 can be generated with depth 2 and be used in subsequent implementation to reduce the number of XOR gates (see Table 1-right). This shows that there is a gap between existing algorithms and the lowest number of XOR gates with respect to the minimum depth.

Table 1: The implementations of M_P (left: *forward* algorithm; right: *backward* algorithm).

No.	Operation	Depth	No.	Operation	Depth
0	$t_1 = x_0 \oplus x_1 // y_6$	1	0	$y_0 = \boldsymbol{y_4} \oplus t_7$	3
1	$t_2 = x_2 \oplus t_1 // y_5$	2	1	$y_1 = \boldsymbol{y_4} \oplus x_5$	3
2	$\boldsymbol{t_3 = x_3 \oplus t_2 // y_4}$	**3**	2	$y_2 = \boldsymbol{y_4} \oplus x_6$	3
3	$t_4 = x_3 \oplus x_5$	1	3	$y_3 = y_5 \oplus t_7$	3
4	$t_5 = t_2 \oplus t_4 // y_1$	3	4	$\boldsymbol{y_4 = t_8 \oplus y_6}$	**2**
5	$t_6 = x_3 \oplus x_6$	1	5	$y_5 = x_2 \oplus y_6$	2
6	$t_7 = t_2 \oplus t_6 // y_2$	3	6	$y_6 = x_0 \oplus x_1$	1
7	$t_8 = x_4 \oplus x_5$	1	7	$t_7 = x_4 \oplus x_5$	1
8	$t_9 = t_2 \oplus t_8 // y_3$	3	8	$t_8 = x_2 \oplus x_3$	1
9	$t_{10} = x_3 \oplus t_8$	2	-	-	-
10	$t_{11} = t_2 \oplus t_{10} // y_0$	3	-	-	-

Therefore, we propose the *backward* framework. It has completely different heuristics. Actually, our method is inspired by the Constant Matrix Multiplication (CMM) problem. The CMM problem is defined as finding a solution using additions, subtractions, and shifts to compute the multiplication of an $m \times n$ constant matrix M over $\mathbb{Z}$. Kumm et al. proposed an algorithm called RPAG-CMM in [KHZ17] to solve the CMM problem with the minimum depth. It catches our attention. However, in our experiment for binary matrix, the XOR gates generated by RPAG-CMM are more than LSL algorithm. Therefore, we study the special version of RPAG-CMM for binary matrix and propose a new heuristic algorithm based on the *backward* framework for binary matrices. The main algorithm will be introduced in Algorithm 2, which is more relevant to the SLP problem.

1.1 Our Contributions

In this paper, we investigate a new strategy of *backward* search. Concretely, rather than combining the inputs to reach the outputs, our new method attempts to iteratively split the outputs until all the input values appear. As shown in Table 1-right, for the matrix M_P, the minimum depth of y_0, y_1, y_2, y_3 is 3, but others are less than 3. We give the method to compute the minimum depth in Subsection 2.1. Thus, we first consider the four values. We have $y_0 = y_4 \oplus t_7$, $y_1 = y_4 \oplus x_5$, $y_2 = y_4 \oplus x_6$, and $y_3 = y_5 \oplus t_7$. Because y_4, y_5, t_7 are not input values, we split them into t_8, y_6, x_2, x_4, x_5. Finally, we split t_8, y_6 by $t_8 = x_2 \oplus x_3$ and $y_6 = x_0 \oplus x_1$. The complete processes can be seen in Subsection 3.1. The above process leads to a new implementation with 9 XOR gates and the depth (that is defined by the longest path from the input to the output) 3. It should be noted that the new method can bound the depth in the search, which contributes to the solutions that take both latency and GE into consideration. For some matrices, the framework can cover more implementations than previous algorithms with minimum depth in a limited time, as illustrated in details in Subsection 3.4.

Then, we apply the strategy to linear layers of block ciphers and find many low-latency candidates for implementations. The results can be seen in Table 2 and Table 3. For the 11 linear layers that we analyzed, we find 8 matrices that have the same XOR gates with minimum depth and optimize 3 matrices in XOR gates. For the 26 lightweight matrices proposed, we find 9 matrices that have the same XOR gates with minimum depth and optimize 8 matrices in XOR gates. Notably, for AES Mixcolumns, we achieve an implementation with 103 XOR gates and depth 3. This is the same as the best low-latency result recently reported in [BFI21]. We also apply the algorithm to 4256 MDS matrices proposed by Li et al. in [LSL$^+$19], and achieve better implementations in XOR gates for 54.3% of them (see Table 8). The smallest matrix among them can be implemented with 86 XOR gates and depth 3, while the previous result is 88 XOR gates in [LSL$^+$19].

Last but not least, we synthesize the above results using two different ASIC libraries, namely TSMC 90 nm and NanGate 45 nm. Our implementation of the AES MixColumns has lower power and latency in the two libraries than those in [LSL$^+$19, XZL$^+$20, LXZZ21, BFI21]. All the source codes and results of this paper are available at

https://github.com/QunLiu-sdu/Towards-Low-Latency-Implementation.

1.2 Organization

In Section 2, we give some basic notations and metrics. Moreover, in Section 3, we formally propose our algorithm and give some examples using our algorithm. All the results and implementations in hardware are given in Section 4. Finally, we conclude and propose future research directions in Section 5.

2 Preliminaries

2.1 Notations

Let $\mathbb{F}_2$ be a field with two elements and its additive and multiplicative identities are respectively denoted as 0 and 1. Let $\mathbb{F}_2^n$ be the vector space of all n-dimensional vectors over $\mathbb{F}_2$ and $\mathbb{F}_2^{n\ell}$ be the vector space of all ℓ-dimensional vectors over $\mathbb{F}_2^n$. We use $\mathbf{M}_\ell(\mathbb{F}_2^n)$ to denote the set of all $\ell \times \ell$ matrices over $\mathbb{F}_2^n$, and use $\mathbf{M}_\ell(\mathrm{GL}(n, \mathbb{F}_2))$ to denote the set

Table 2: The XOR number/depth of implementation cost of matrices. Except for the last row, every matrix is consistent with the choice in [KLSW17].

Matrix[a]	[KLSW17][b]	[XZL+20][b]	[LXZZ21][b]	[LSL+19][c]	[BFI21][c]	**Ours**[c]
AES [DR20]	97/8	92/6	**91/7**	105/3	**103/3**	**103/3**
SMALLSCALE AES [CMR05]	47/7	**43/5**	**43/5**	49/3	49/3	**47/3**
JOLTIK [JNP15]	48/4	44/7	**43/8**	51/3	50/3	**48/3**
QARMA128 [Ava17]	48/3	48/3	48/3	**48/2**	**48/2**	**48/2**
MIDORI [BBI+15]	24/4	24/3	24/3	**24/2**	**24/2**	**24/2**
PRINCE M_0,M_1 [BCG+12]	24/4	24/6	24/6	**24/2**	**24/2**	**24/2**
PRIDE L_0,L_3 [ADK+14]	24/4	24/3	24/3	**24/2**	**24/2**	**24/2**
PRIDE L_1,L_2 [ADK+14]	24/3	24/3	24/3	**24/2**	**24/2**	**24/2**
QARMA64 [Ava17]	24/3	24/5	24/5	**24/2**	**24/2**	**24/2**
SKINNY64 [BJK+16]	**12/2**	**12/2**	**12/2**	**12/2**	**12/2**	**12/2**
CAMELLIA [AIK+00]	**16/4**	**16/4**	**16/4**	20/3	-	**19/3**
[SKOP15](Hadamard)	48/3	**44/7**	**44/7**	51/3	50/3	**49/3**
[LS16](Circulant)	44/3	44/6	**43/4**	47/3	**44/3**	**44/3**
[LW16](Circulant)	44/5	44/8	**43/4**	47/3	**44/3**	**44/3**
[BKL16](Circulant)	42/5	41/6	**40/5**	47/3	**43/3**	45/3
[SS16](Toeplitz)	43/5	41/7	**40/7**	44/3	**43/3**	45/3
[JPST17]	43/5	41/6	**40/6**	**45/3**	**45/3**	**45/3**
[SKOP15](Involutory)	48/4	44/8	**43/8**	51/3	49/3	**48/3**
[LW16](Involutory)	48/4	44/6	**43/8**	51/3	49/3	**48/3**
[SS16](Involutory)	42/4	38/8	**37/7**	48/3	46/3	**45/3**
[JPST17](Involutory)	47/7	**41/6**	41/10	**47/3**	**47/3**	**47/3**
[SKOP15](Hadamard)	100/5	**90/6**	91/7	102/3	**99/3**	100/3
[LS16](Circulant)	112/5	121/11	**107/6**	114/3	**113/3**	**113/3**
[LW16]	102/3	104/6	**99/4**	**102/3**	103/3	**102/3**
[BKL16](Circulant)	110/5	114/10	**105/7**	112/3	**110/3**	111/3
[SS16](Toeplitz)	107/5	114/12	**100/9**	**107/3**	**107/3**	**107/3**
[JPST17](Subfield)	86/5	82/7	**80/6**	**90/3**	**90/3**	93/3
[SKOP15](Involutory)	100/6	91/6	**89/8**	102/3	**100/3**	**100/3**
[LW16](Involutory)	91/6	87/6	**86/9**	99/3	95/3	**94/3**
[SS16](Involutory)	100/6	93/8	**92/8**	104/4	**102/4**	109/4
[JPST17](Involutory)	91/7	**83/6**	84/6	**94/3**	**94/3**	97/3
[KLSW17]	**84/4**	-	-	96/3	-	**92/3**
[LSL+19](Involutory)	-	-	-	88/3	-	**86/3**[d]

[a] For the block cipher, we optimize the matrix used in the linear layer.
[b] The results only take the number of XOR gates into account.
[c] The results take the number of XOR gates into account with respect to the minimum depth.
[d] We show the lowest one from all the results.

Table 3: The XOR number/depth of implementation cost of matrices with depth 3 in [DL18].

Matrix	Instantiation	Const [DL18]	BP [BP10]	Paar2 [Paa97]	RSDF [RTA18]	RNBP [TP19]	A1 [TP19]	A2 [TP19]	**Ours**
$M_{4,3}^{9,5}$	A_4	41/3	40/4	43/4	43/7	40/5	41/7	40/5	**40/3**
$M_{4,3}^{9,5}$	A_4^{-1}	**41/3**	43/5	44/3	44/10	41/6	41/7	40/6	**41/3**
$M_{4,3}^{9,5}$	A_8	**77/3**	76/7	86/4	87/10	76/6	77/7	76/6	82/3
$M_{4,3}^{9,5}$	A_8^{-1}	**77/3**	79/5	86/4	91/14	77/6	77/7	77/5	81/3

of all $\ell \times \ell$ matrices whose elements are taken from the general linear group $\mathrm{GL}(n, \mathbb{F}_2)$ formed by all invertible $n \times n$ matrices over $\mathbb{F}_2$. Note that we abuse $\mathbf{M}_\ell(\mathbb{F}_2^n)$ or $\mathbf{M}_{n\ell}(\mathbb{F}_2)$ to denote the set of all $n\ell \times n\ell$ binary matrices.

For a vector $x \in \mathbb{F}_2^{n\ell}$, let $\omega_n(x)$ be the number of non-zero n-bit chunks. Particularly, if $n = 1$, $\omega_1(x)$ is the Hamming weight of x. For a matrix $A \in \mathbf{M}_{n\ell}(\mathbb{F}_2)$, the branch number $\mathcal{B}_n(A)$ is defined as $\min_{x \in \mathbb{F}_2^{n\ell} \setminus \{0\}} \{\omega_n(x) + \omega_n(Ax)\}$. Then, an invertible matrix A is an MDS matrix if and only if $\mathcal{B}_n(A) = \ell + 1$. Moreover, A is an involutory MDS matrix, if A is MDS and $A = A^{-1}$.

For any linear layer of a cipher associated to an $m \times n$ binary matrix A, given inputs $\vec{x} = (x_0, x_1, ..., x_{n-1})^T$, the outputs $\vec{y} = (y_0, y_1, ..., y_{m-1})^T$ of the linear layer can be calculated by $\vec{y} = A\vec{x}$, and y_i can be computed by

$$y_i = a_{i0}x_0 \oplus a_{i1}x_1 \oplus \ldots \oplus a_{i(n-1)}x_{n-1},$$

where each coefficient a_{ij} is the entry of matrix A at i-th row and j-th column. We can then associate y_i with a binary vector:

$$[a_{i0}, a_{i1}, \ldots, a_{i(n-1)}].$$

We use "node" to define such binary vector. That is, the node $\mathcal{N}_{y_j} \overset{\text{def}}{=} [a_{i0}, a_{i1}, ..., a_{i(n-1)}]$. $\mathcal{N}_{x_i}$ is the unit node and $\mathcal{N}_{y_j}$ is the target node. For three nodes $\mathcal{N}_{y_{i_1}}$, $\mathcal{N}_{y_{i_2}}$ and $\mathcal{N}_{y_i'}$, we say $\mathcal{N}_{y_{i_1}}$ and $\mathcal{N}_{y_{i_2}}$ generate $\mathcal{N}_{y_i'}$, if $\mathcal{N}_{y_i'} = \mathcal{N}_{y_{i_1}} \oplus \mathcal{N}_{y_{i_2}}$ with $\oplus$ element-wise plus. For a node $\mathcal{N}_{y_i}$, we define its depth $\mathcal{D}(\mathcal{N}_{y_i})$ as the maximum number of XOR operations of a path from unit nodes to $\mathcal{N}_{y_i}$. Obviously, the depth of $\mathcal{D}(\mathcal{N}_{y_i'}) \geq \mathcal{D}(\mathcal{N}_{y_{i_1}}) + 1$ and $\mathcal{D}(\mathcal{N}_{y_i'}) \geq \mathcal{D}(\mathcal{N}_{y_{i_2}}) + 1$. We define the minimum depth of a node $\mathcal{N}_{y_i}$ as:

$$\lceil \log_2(\omega_1(\mathcal{N}_{y_i})) \rceil. \tag{1}$$

Given an $m \times n$ binary matrix A over $\mathbb{F}_2$, the minimum depth of A is defined as:

$$\max_{y_i \in A} \{\lceil \log_2(\omega_1(\mathcal{N}_{y_i})) \rceil\}, \text{ where } \mathcal{N}_{y_i} = [a_{i0}, a_{i1}, \ldots, a_{i(n-1)}]. \tag{2}$$

Note that we can treat the implementation of the matrix A as a graph. Thus, the depth of the implementation is the critical path length of the graph.

2.2 Metrics

In this sub-section, we recall some metrics for a matrix over $\mathbb{F}_2$, which are helpful in the proposed solver of minimum depth SLP problem.

The direct-XOR (d-XOR) [KPPY14]. Given an $m \times n$ binary matrix A over $\mathbb{F}_2$, the d-XOR count is defined as $\omega_1(A) - m$, where we define $\omega_1(A)$ as the Hamming weight of A, i.e., the number of 1's in A.

The sequential-XOR (s-XOR) [JPST17]. Let $A \in GL(n, \mathbb{F}_2)$ be an invertible matrix. Assume $x_0, x_1, ..., x_{n-1}$ are the n input values of A. It is always possible to perform a sequence of XOR instructions $x_i = x_i \oplus x_j$ with $0 \leq i, j \leq n-1$, such that the n input values are updated to the n output values. The s-XOR count of A is defined as the minimal number of XOR instructions to update the inputs to the outputs.

The general-XOR (g-XOR) [XZL+20]. Given an $m \times n$ binary matrix A over $\mathbb{F}_2$, the implementation of A can be viewed as a sequence of XOR operations $x_i = x_{j_1} \oplus x_{j_2}$ where $0 < x_{j_1}, x_{j_2} < i$ and $i = n, n+1, \ldots, t-1$. The g-XOR count is defined as the minimal number of operations $x_i = x_{j_1} \oplus x_{j_2}$ that compute the m outputs completely.

Since the d-XOR is intuitive and easy to compute, it has been adopted in the design of new lightweight diffusion layers. For further optimization, the s-XOR and the g-XOR are used in evaluating matrices. The difference is that the g-XOR can generate new values while the s-XOR always renews original values. For example, for computing $x_0 \oplus x_1$, the s-XOR performs $x_0 = x_0 \oplus x_1$ or $x_1 = x_0 \oplus x_1$. While the g-XOR can generate t_2 with $t_2 = x_0 \oplus x_1$.

Global optimization. For an $m \times n$ binary matrix A over $\mathbb{F}_2$, we can obtain an estimation of its hardware cost by finding a good linear straight-line program corresponding to A with state-of-the-art automatic tools based on certain SLP heuristics [BMP13]. Using different heuristics, global optimization leads to better results than local optimization [KLSW17]. And in this paper, the global optimization corresponds to optimizing with respect to the g-XOR metric.

3 Backward Search

In this section, we formally present the new search framework and algorithm. The new framework provides another intuitive perspective to solve the SLP problem with respect to achieving the minimum depth. We find that other methods always ignore some choices because of their rules. Our new framework can keep such choices (ignored by other methods) and potentially can achieve better solutions. We only use the g-XOR metric in our framework.

First, we introduce the *backward* search framework and provide an example in Subsection 3.1. The example using our strategy can help readers to understand our framework. We begin with the example and then formalize the process. Then, we propose five heuristics rules of splitting nodes for the low-area *backward* strategy in Subsection 3.2, and discuss their priorities (i.e., to determine which rule will be used if multiple ones can be matched) in Subsection 3.3. Finally, we compare LSL algorithm with our framework in Subsection 3.4, to explain the advantages of the *backward* search for low-latency implementation compared to the forward one.

The notations used in this section are as follows. For convenience, we use node y instead of $\mathcal{N}_y$.

- A: An $m \times n$ binary matrix to be implemented.
- x_i: The unit node with the i-th bit set.
- y_i: The target node of A.
- $\mathcal{X}$: A set of unit nodes.
- d_{y_i}: The minimum depth of y_i.
- d_A: The maximum value of the depth of each row in A.
- L_d: A set of all the nodes with minimum depth d.
- $\mathcal{W}$: The working set containing target nodes.
- $\mathcal{P}$: The predecessor set containing the predecessor nodes.
- $\mathcal{E}$: The edge set containing the edges used to generate the graph.
- s: The current depth of $\mathcal{W}$ used to determine which nodes will be considered.

3.1 The Backward Strategy

The *backward* strategy can be regarded as the search from outputs to inputs by iteratively *splitting* the nodes. Given an implementation of a matrix $A : \{x_i = x_{j_1} \oplus x_{j_2}\}$ with $0 < j_1, j_2 < i$ and $i = n, n+1, \ldots$, every non-unit node $x_i (i \geq n)$ can be split into two nodes x_{j_1} and x_{j_2}. If we use new nodes to split the target nodes into unit nodes, we can also find an implementation of A. For convenience, we call p and q the predecessors of w, if and only if $w = p \oplus q$ holds. The *backward* strategy is given in Algorithm 2. We bring the example M_P in Section 1 to cast some light on our framework:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Initialization. In M_P, y_i represents the i-th row of the matrix, and x_i is the unit node with the i-th bit set. We put each row in M_P into the working set $\mathcal{W}$. The unit set $\mathcal{X}$ contains all the unit nodes. Thus, we have the predecessor set $\mathcal{P} = \phi$, $\mathcal{W} = \{y_0, y_1, y_2, y_3, y_4, y_5, y_6\}$, and $\mathcal{X} = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6\}$. The nodes in $\mathcal{W}$ need to be split. Then, we use Equation (1) to give the minimum depth of each node:

$$d_{y_0} = 3, d_{y_1} = 3, d_{y_2} = 3, d_{y_3} = 3, d_{y_4} = 2, d_{y_5} = 2, d_{y_6} = 1.$$

And we calculate the minimum depth of M_P is $d_{M_P} = 3$ by Equation (2).

Step 1. The current depth s is $d_{M_P} = 3$. For $y_i \in \mathcal{W}$, if the minimum depth d_{y_i} of y_i meets $d_{y_i} < s$, we put it from $\mathcal{W}$ to $\mathcal{P}$. Therefore, $\mathcal{W} = \{y_0, y_1, y_2, y_3\}$. $\mathcal{P} = \{y_4, y_5, y_6\}$.

Step 2. $s = d_{M_P} = 3$. We split $y_0, y_1, y_2, y_3 \in \mathcal{W}$. If the nodes in $\mathcal{P}$ can split one node in $\mathcal{W}$, we will not generate new nodes. We first use y_4 to split y_0, y_1, y_2. We have to generate new nodes $t_7 = [0, 0, 0, 0, 1, 1, 0]$, x_5, and x_6, as we have $y_0 = y_4 \oplus t_7$, $y_1 = y_4 \oplus x_5$, and $y_2 = y_4 \oplus x_6$. Then, we find that y_3 can be split into y_5 and t_7 without any new nodes. Now, $\mathcal{W} = \phi$, and $\mathcal{P} = \{y_4, y_5, y_6, t_7, x_5, x_6\}$.

Step 3. Because of $\mathcal{W} = \phi$, we set $\mathcal{W} \leftarrow \mathcal{P}$, $\mathcal{P} \leftarrow \phi$, and $s = s - 1 = 2$. Like Step 1, we put y_6, t_7, x_5, x_6 to $\mathcal{P}$.

Step 4. $s = 2$. $\mathcal{W} = \{y_4, y_5\}$. $\mathcal{P} = \{y_6, t_7, x_5, x_6\}$. We generate $t_8 = [0, 0, 1, 1, 0, 0, 0]$ and x_2. y_4 and y_5 can be split into t_8, y_6 and x_2, y_6 respectively. Thus, $\mathcal{W} = \phi$, and $\mathcal{P} = \{y_6, t_7, t_8, x_2, x_5, x_6\}$.

Step 5. Because of $\mathcal{W} = \phi$, we set $\mathcal{W} \leftarrow \mathcal{P}$, $\mathcal{P} \leftarrow \phi$, and $s = s - 1 = 1$. Like Step 1, we put x_2, x_5, x_6 to $\mathcal{P}$.

Step 6. $s = 1$. $\mathcal{W} = \{y_6, t_7, t_8\}$. $\mathcal{P} = \{x_2, x_5, x_6\}$. New nodes x_0, x_1, x_3, and x_4 are generated to split y_6, t_7, and t_8. Therefore, $\mathcal{W} = \phi$, and $\mathcal{P} = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6\}$.

Step 7. Because of $\mathcal{W} = \phi$, we set $\mathcal{W} \leftarrow \mathcal{P}$, $\mathcal{P} \leftarrow \phi$, and $s = s - 1 = 0$. Now, $\mathcal{W} = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6\}$. All the target nodes are split into unit nodes. We finish the search.

Now, we formalize our framework. First, we state the problem that we focus on. The SLP problem is defined as follows.

Definition 1 ([BMP08]). The Shortest Linear Program (SLP) problem is defined as finding a solution with the minimum number of XORs to compute the multiplication of an $m \times n$ constant matrix A over $\mathbb{F}_2$.

Then, we extend the SLP problem by considering the depth of the solution. The *backward* framework aims at solving the SLP problem as well as achieving the minimum depth. In other words, we give the solution of the SLP problem, where the depth of each node is not greater than the minimum depth of A.

Definition 2. The *backward* framework is an approach to search for a solution for the SLP problem that starts from target nodes, chooses a node iteratively, and splits it into two ones until all the nodes are unit ones.

Our framework returns a directed graph formed by nodes and by edges connecting pairs of nodes. In the case of a directed graph, each edge has an orientation, from one node to another. If there exists an edge from p to q, we say that q has a predecessor node p. And the in-degree of a node is defined as the number of edges whose origins are the node. As our framework always splits one node into two predecessor nodes, the in-degree of every node is either 0 or 2. This gives rise to the following property.

Property 1. The *backward* framework returns a directed graph. In the graph, the in-degree of each node is 0 or 2. Every unit node has the in-degree 0. And every non-unit node has the in-degree 2 and can represent an XOR gate.

We use the set $\mathcal{E}$ to save the graph. The implementation is encoded in $\mathcal{E}$ in the form (p, u), which means that there exists an edge from p to u. Normally, a graph is defined by a tuple of sets, one for edges and one for vertices (i.e., the nodes). However, for the sake of brevity, we omit the set of vertices since every edge (p, u) explicitly implies that the graph contains two nodes p and u. For each non-unit node, there exist two nodes p and q such that (p, u) and (q, u) are saved in $\mathcal{E}$. In Step 2 of the example, we have $y_0 = y_4 \oplus t_7$. Thus, we put (y_4, y_0) and (t_7, y_0) into $\mathcal{E}$.

For a non-unit node u, the splitting method uses two predecessor nodes a and b to split u by $u = a \oplus b$. The depth of a graph is defined as the number of non-unit nodes involved in its critical path. If we choose the appropriate predecessors, we can ensure that the SLP problem is solved with respect to achieving the minimum depth.

Proposition 1. *For any $y \in L_d$ $(d \geq 1)$, there exist y_1 and y_2 with $y_1 \in L_{d-1}$ or $y_2 \in L_{d-1}$ such that $y_1 \oplus y_2 = y$.*

Proof. Based on Equation (1) and the definition of L_d, we have

$$\lceil \log_2(\omega_1(y)) \rceil = d.$$

Therefore, we obtain that

$$2^{d-1} < \omega_1(y) \leq 2^d.$$

For $d = 1$, Proposition 1 holds obviously. We consider the case of $d \geq 2$. If the minimum depth $d_{(y_i)}$ $(i \in \{0, 1\})$ of y_1 or y_2 meets $d_{(y_i)} \geq d$, $y \in L_d$ does not hold. To ensure that each node reaches the minimum depth, we only consider the case of $d_{y_1}, d_{y_2} < d$. Assume that $y_1 \notin L_{d-1}$ and $y_2 \notin L_{d-1}$. Without loss of generality, let $y_1, y_2 \in L_k$ $(0 \leq k < d-1)$, we have

$$0 < \omega_1(y_1), \omega_1(y_2) \leq 2^k \leq 2^{d-2}.$$

Since $y_1 \notin L_{d-1}$ and $y_2 \notin L_{d-1}$, we have

$$0 < \omega_1(y) \leq \omega_1(y_1) + \omega_1(y_2) \leq 2^k + 2^k \leq 2^{d-1}.$$

This contradicts to $2^{d-1} < \omega_1(y)$. Therefore, we must have $y_1 \in L_{d-1}$ or $y_2 \in L_{d-1}$. □

Proposition 1 can help us to execute the splitting process. We use an example to illustrate it. Suppose that $y \in L_3$. The Hamming weight $\omega_1(y)$ of y is 5, 6, 7, or 8. If $y_1, y_2 \in L_1$, $\omega_1(y_1)$ and $\omega_1(y_2)$ are not greater than 2. $y = y_1 \oplus y_2$ is impossible. Thus, we must have $y_1 \in L_2$ or $y_2 \in L_2$. We use $\mathcal{W}$ to save the nodes which need to be split. Note that the splitting method may generate new nodes. We put them into $\mathcal{P}$. We recommend reusing the nodes in $\mathcal{P}$ for reducing the number of XOR gates. Our heuristics in Subsection 3.2 aim to reuse non-unit nodes based on the current states of $\mathcal{W}$ and $\mathcal{P}$. And if we cannot match any heuristics, a function DefaultSplit() is used to split the nodes by a default method (see Algorithm 1).

Algorithm 1 DefaultSplit()

```
Input: A target set $\mathcal{W}$
Output: The target node $w$ and suitable predecessor nodes $p$ and $q$
  $w \overset{select}{\longleftarrow} \mathcal{W}$
  $s \leftarrow \lceil \log_2(\omega_1(w)) \rceil$
  $p \overset{select}{\longleftarrow} L_{s-1}$
  $q \leftarrow w \oplus p$
  return $w, p, q$
```

In the function DefaultSplit(), we use an operation "$\overset{select}{\longleftarrow}$". The operation selects an element from the set randomly. Algorithm 1 shows the selection based on Proposition 1. The selection of p is a random process. Suppose that we need to split $y = [1, 1, 1, 1, 1]$. Because of $y \in L_3$, we first select predecessor node y_1. $\omega_1(y_1)$ can only be one of $\{2, 3, 4\}$. We randomly choose $\omega_1(y_1) = 3$. Then, we randomly set the three bits of y_1 to 1 and others to 0. Thus, in this selection, we select $y_1 = [1, 1, 1, 0, 0]$ in L_2. And y_2 can be generated by $y_2 = y \oplus y_1 = [0, 0, 0, 1, 1]$.

Proposition 2. *Given an $m \times n$ binary matrix A and its minimum depth d_A, we can always find a graph based on the backward framework where each target node y_i can be split into unit nodes and the depth of each node is not greater than d_A.*

Proof. The target nodes are $y_0, y_1, ..., y_{m-1}$. We put them into the working set $\mathcal{W}$ and initialize the predecessor set $\mathcal{P} \leftarrow \phi$. Based on Proposition 1, for each node y_i in $\mathcal{W}$, if y_i is not unit node, we can always find two predecessors t_p and t_q where d_{t_p} and d_{t_q} are less than d_{y_i}. Through repeating the process, we split all the nodes in $\mathcal{W}$ into $\mathcal{P}$. Next, we treat $\mathcal{P}$ as $\mathcal{W}$ and continue to split the nodes in $\mathcal{W}$. The above process stops only when no nodes need to be split. Finally, every target node will be split into unit nodes. □

Proposition 2 ensures that our framework can always work. Note that the proof of the proposition implies the processes of the *backward* framework. It is simple to see that this method is suitable for low-latency implementation. For an $m \times n$ binary matrix A, the minimum depth of A is d_A. If using the *backward* strategy, we can use d_{y_i} to represent the minimum depth of each target node y_i ($0 \le i \le m-1$) and have $y_i \in L_{d_{y_i}}$. When selecting predecessor nodes, we always choose them from L_k ($k < d_{y_i}$), which can reach the bound of minimum depth easily. We can give the following steps and the complete process is in Algorithm 2.

1. Initialize the target set $\mathcal{W}$ and the predecessor set $\mathcal{P}$.
2. If $\mathcal{X} \cup \mathcal{W} = \mathcal{X}$, return the implementation $\mathcal{E}$.
3. If $\mathcal{W} = \phi$, treat $\mathcal{P}$ as the target set to split, $s \leftarrow s - 1$, go to Step 2.
4. Use heuristics to split. If it is successful, go to Step 3.
5. Use the default method to split. Go to Step 3.

Algorithm 2 *backward* search framework

```
Input: An $m \times n$ binary matrix $A$
Output: The implementation $\mathcal{E}$ of $A$
  $\mathcal{W} \leftarrow \{y_i\}(0 \le i \le m-1)$                          ▷ The target set
  $\mathcal{X} \leftarrow \{x_j\}(0 \le j \le n-1)$
  $\mathcal{P} \leftarrow \phi$
  $\mathcal{E} \leftarrow \phi$
  $d_{y_i} \leftarrow \lceil \log_2(\omega_1(y_i)) \rceil$
  $d_A \leftarrow \max\{d_{y_i}\}$
  $s \leftarrow d_A$
  while $\mathcal{X} \cup \mathcal{W} \neq \mathcal{X}$ do
      while $\mathcal{W} \neq \phi$ do
          if Search($\mathcal{W}, \mathcal{P}, s$) = True then     ▷ Using heuristics to reduce circuit area
              continue
          end if
          $w, p_1, p_2$ = DefaultSplit($\mathcal{W}$)                 ▷ Default method
          $\mathcal{W} \leftarrow \mathcal{W} \backslash \{w\}$, $\mathcal{P} \leftarrow \mathcal{P} \cup \{p_1, p_2\}$
          $\mathcal{E} \leftarrow \mathcal{E} \cup \{(p_1, w), (p_2, w)\}$
          continue
      end while
      $s \leftarrow s - 1$                                             ▷ Deal with the depth
      $\mathcal{W} \leftarrow \mathcal{P}$
      $\mathcal{P} \leftarrow \phi$
  end while
  return $\mathcal{E}$
```

As the processes of splitting and generating nodes are randomized, recalling the computations at different times would lead to different results. Hence, it is difficult to determine how long we wait to achieve the best solution. We execute the algorithm in a limited and reasonable time (several days) to collect many implementations and select the best one among them. This strategy is quite similar to many previous search approaches in, e.g., [KLSW17, TP19, XZL$^+$20, BFI21].

For a matrix, the initial information includes the target nodes and unit nodes. All the target nodes will be put into $\mathcal{W}$. Next, we generate new predecessor nodes or use existing nodes to split the nodes in $\mathcal{W}$. Which predecessor nodes will be used depends on different strategies. Since the selection of the predecessors is randomized, it is possible that we cannot find a good implementation taking a long time. Thus, we provide five rules of heuristics (see Subsection 3.2). The rules can help us to find the sub-optimal result. It is easy to transform our implementation to a circuit. The depiction "a is split into b and c" means $a = b \oplus c$.

Given the current depth s, we only search the predecessor nodes from $L_k(0 \le k < s)$. L_k is a set of all the nodes with minimum depth k. The loop condition in Algorithm 2 $\mathcal{X} \cup \mathcal{W} \neq \mathcal{X}$ indicates that there is at least one non-unit node in $\mathcal{W}$, which will be split according to our strategy. In Step 7 of the example, the loop condition does not hold, and it returns the result. In addition, for splitting the nodes with fewer XOR gates, we give a function Search() to describe how the algorithm uses the heuristics to split nodes and update parameters. We can view more details in Algorithm 3. The purpose of the heuristics is to reduce the XOR gates in a reasonable time.

3.2 Heuristics of Splitting Nodes

In this section, we present our heuristic algorithm that takes the working set $\mathcal{W}$, the predecessor set $\mathcal{P}$, and the current depth s in Algorithm 2 and output the best candidate node to be split and the splitting scheme. We present several splitting rules of $\mathcal{W}$, $\mathcal{P}$, and s and the corresponding action of splitting. The heuristic search is performed by matching one of the rules and conducting the corresponding actions. Most examples are from M_P.

Algorithm 3 Search()

```
Input: $\mathcal{W}$, $\mathcal{P}$, and $s$ in the framework
Output: True or False
  if $\exists w_1 \in \mathcal{W}$ meets $d_{w_1} < s$ then
    $\mathcal{W} \leftarrow \mathcal{W} \backslash \{w_1\}$, $\mathcal{P} \leftarrow \mathcal{P} \cup \{w_1\}$        ▷ Rule 1
    return True
  end if
  if $\exists p, q \in \mathcal{P}$, $\exists w \in \mathcal{W}$ s.t. $w = p \oplus q$ then
    $\mathcal{W} \leftarrow \mathcal{W} \backslash \{w\}$        ▷ Rule 2
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(p, w), (q, w)\}$
    return True
  end if
  if $\exists p \in \mathcal{P}$, $\exists w \in \mathcal{W}$, $\exists q \in L_k (0 \leq k < s)$ s.t. $w = p \oplus q$ then
    $\mathcal{W} \leftarrow \mathcal{W} \backslash \{w\}$        ▷ Rule 3
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{q\}$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(p, w), (q, w)\}$
    return True
  end if
  if $\exists p_1, p_2, p_3 \in L_k (0 \leq k < s)$, $\exists w_1, w_2 \in \mathcal{W}$ s.t. $w_1 = p_1 \oplus p_2, w_2 = p_2 \oplus p_3$ then
    $\mathcal{W} \leftarrow \mathcal{W} \backslash \{w_1, w_2\}$        ▷ Rule 4
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{p_1, p_2, p_3\}$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(p_1, w_1), (p_2, w_1), (p_2, w_2), (p_3, w_2)\}$
    return True
  end if
  return False
```

Rule 1:
Match: There is $w_1 \in \mathcal{W}$ which meets $d_{w_1} < s$.
Operation: Put w_1 from $\mathcal{W}$ to $\mathcal{P}$. (see Figure 2(a)).
Example: In Step 1 of the example in Subsection 3.1, $\mathcal{W} = \{y_0, y_1, y_2, y_3, y_4, y_5, y_6\}$ and $s = 3$. The minimum depth of y_4, y_5, y_6 is less than s. Thus, we set $\mathcal{W} \leftarrow \mathcal{W} \backslash \{y_4, y_5, y_6\}$ and $\mathcal{P} \leftarrow \mathcal{P} \cup \{y_4, y_5, y_6\}$.

Rule 2:
Match: $\exists p_1, p_2 \in \mathcal{P}$, $\exists w_1 \in \mathcal{W}$ s.t. $w_1 = p_1 \oplus p_2$.
Operation: Delete w_1. (see Figure 2(b)).
Example: In Step 2 of the example in Subsection 3.1, after splitting y_0, y_1, y_2, $\mathcal{W} = \{y_3\}$, $\mathcal{P} = \{y_4, y_5, y_6, t_7, x_5, x_6\}$. We find that y_3 can be split directly into y_5 and t_7. Thus, we set $\mathcal{W} \leftarrow \mathcal{W} \backslash \{y_3\}$.

Rule 3:
Match: $\exists p_1 \in \mathcal{P}$, $\exists w_1 \in \mathcal{W}$, $\exists p_2 \in L_k (0 \leq k < s)$ s.t. $w_1 = p_1 \oplus p_2$.
Operation: $\mathcal{W} \leftarrow \mathcal{W} \backslash \{w_1\}, \mathcal{P} \leftarrow \mathcal{P} \cup \{p_2\}$ (see Figure 2(c)).
Example: In Step 4 of the example in Subsection 3.1, $\mathcal{W} = \{y_4, y_5\}$, $\mathcal{P} = \{y_6, t_7, x_5, x_6\}$ and $s = 2$. We use y_6 to split y_5. We need to generate a new node x_2 to meet $y_5 = y_6 \oplus x_2$, in which $y_5 \in L_2$, $y_6 \in L_1$, and $x_2 \in L_0$. Thus, we set $\mathcal{W} \leftarrow \mathcal{W} \backslash \{y_5\}$ and $\mathcal{P} \leftarrow \mathcal{P} \cup \{x_2\}$.

Rule 4:
Match: $\exists p_1, p_2, p_3 \in L_k (0 \le k < s)$, $\exists w_1, w_2 \in \mathcal{W}$ s.t. $w_1 = p_1 \oplus p_2, w_2 = p_2 \oplus p_3$.
Operation: $\mathcal{W} \leftarrow \mathcal{W} \backslash \{w_1, w_2\}, \mathcal{P} \leftarrow \mathcal{P} \cup \{p_1, p_2, p_3\}$ (see Figure 2(d)).
Example: Suppose that $s = 2$ and $\mathcal{W}$ has two nodes y_0 and y_1. We can split them to three new predecessors $p_1 = [0,0,0,0,1,1,0]$, $p_2 = [1,1,1,1,0,0,0]$, and $p_3 = [0,0,0,0,0,1,0]$ by $y_0 = p_1 \oplus p_2$ and $y_1 = p_2 \oplus p_3$, in which $y_0 \in L_3$, $y_1 \in L_3$, $p_1 \in L_1$, $p_2 \in L_2$, and $p_3 \in L_0$. We set $\mathcal{W} \leftarrow \mathcal{W} \backslash \{y_0, y_1\}, \mathcal{P} \leftarrow \mathcal{P} \cup \{p_1, p_2, p_3\}$.

Rule 5:
Match: This is the default splitting rule.
Operation: Using the default method to split. (see Figure 2(e)).
Example: In Step 6 of the example in Subsection 3.1, $\mathcal{W} = \{y_6, t_7, t_8\}$. $\mathcal{P} = \{x_2, x_5, x_6\}$. After splitting t_7, t_8, we use DefaultSplit() to split y_6. We generate x_0 and x_1, then, $y_6 = x_0 \oplus x_1$. We set $\mathcal{W} \leftarrow \mathcal{W} \backslash \{y_6\}, \mathcal{P} \leftarrow \mathcal{P} \cup \{x_0, x_1\}$.

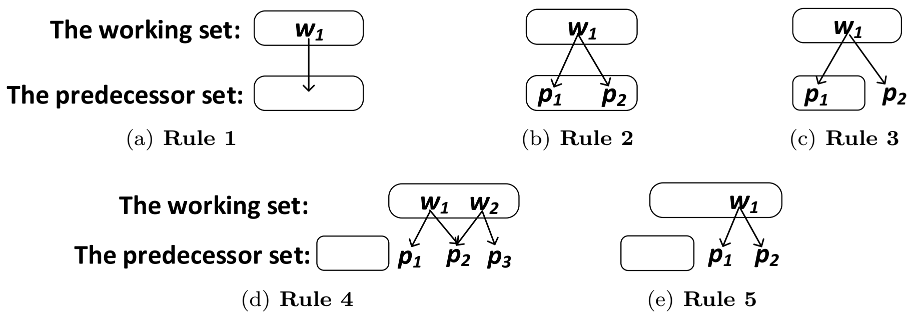

Figure 2: The different splitting rules

3.3 Discussion on the Priority

Another problem is to decide which rule takes precedence. If we make bad choices, the number of nodes in $\mathcal{P}$ will increase, which may lead to a worse implementation. Our strategy is a greedy one to choose the best candidate in the current state rather than the global optimal choice. The method helps us return a feasible solution in a reasonable time.

Thus, we need to investigate the priority between different rules. For this problem, we make a series of experiments. We try to modify the order of different rules, give them a hierarchical priority, and compare their results. Meanwhile, we give the theoretical costs of each rule in Table 4. Ideally, the output nodes can be generated without any cost. However, this case cannot happen. Each non-unit node has two predecessors with one XOR gate.

Rule 1 will be matched first because it reduces one node in $\mathcal{W}$ and adds one node in $\mathcal{P}$ without additional XOR operations. In Rule 2, we need one XOR gate. Besides, we don't generate new predecessors in Rule 2, while Rule 3 generates a new predecessor. Thus, we prefer Rule 2 to Rule 3 if both of them can be matched.

In the worst case, i.e., Rule 5, a node will be split into two predecessor nodes. Rule 4 is preferred than Rule 5 because two nodes in $\mathcal{W}$ are only split into 3 new predecessors. Actually, Rule 4 can be regarded as the combination of Rule 3 and Rule 5. Thus, in some cases, Rule 3 and Rule 4 can be regarded as the same. Therefore, the relation of rules' priorities is

$$\textbf{Rule 1} > \textbf{Rule 2} > \textbf{Rule 3} \ge \textbf{Rule 4} > \textbf{Rule 5}.$$

Candidates with the same priority. In the running of our strategy, it is inevitable that several candidates with the same priority may be chosen. For the case of a tie, one

Table 4: The costs of predecessors

Rule	Output nodes[1]	Gates[2]	predecessors[3]
Rule 1	1	0	1
Rule 2	1	1	0
Rule 3	1	1	1
Rule 4	2	2	3
Rule 5	1	1	2

[1] The number of nodes we deal with.
[2] The number of XOR gates we use.
[3] The number of new predecessors we generate.

possible solution is to record all the candidates and try them sequentially. However, this may lead to a large memory requirement. Sometimes, we even cannot exhaust search all possible candidates in a reasonable time. We use an alternative method. It takes a random selection and randomly selects a candidate to speed up the search process.

3.4 Comparison of Backward and Forward Search for Low-latency Implementation

In this section, we explain the advantage of the *backward* searching for low-latency implementation beyond the forward one. The advantage is that the *backward* framework ensures that each node reaches the minimum depth, which holds for all matrices. However the *forward* algorithm cannot (see Section 1). This feature will affect whether the node can be used to generate new nodes. Thus, for some matrices, the framework can cover more implementations than previous algorithms with minimum depth in a limited time. Then, we give a further explanation as follows.

First, we review the very effective *forward* search algorithm proposed by Boyar et al. in [BP10]. Given a set of unit nodes and target nodes as a binary matrix, for generating every row y_i $(0 \leq i \leq m-1)$ in matrix, BP algorithm places all unit nodes $\{x_0, x_1, ..., x_{n-1}\}$ into the *Base* set B and initialize an m-integer vector *Dist* which keeps track of the distances of each target node from B. The *Dist* is $[\delta(B, y_0), \delta(B, y_1), ..., \delta(B, y_{m-1})]$, where $\delta(B, y_i)$ indicates the minimum number of XOR gates required that can obtain y_i from B. Then, Boyar repeatedly picks two nodes from B according to some rules, adds them together as a new node, and puts this new node into B. The rules are described as follows:

1. Perform XOR on every unique pair of nodes in B to generate a new node. The node is used to re-evaluate the *Dist* vector, and calculate the new distance $\sum_{i=0}^{m-1} Dist[i]$.
2. Select the smallest distance and put the corresponding node into B. Meanwhile, if a pair can generate the target node, then choose it first.
3. In the case of a tie, use the Euclidean norm of *Dist* to determine which candidate is better.

Intuitively, after each iteration, the *Base* set becomes closer to the target nodes according to the reduction of *Dist*. The algorithm stops executing if and only if the *Dist* is a zero vector. That is, the *Base* set can compute all the target nodes. In the following, LSL algorithm enhances BP algorithm with circuit depth awareness in [LSL+19]. It proposes a new set that keeps track of the circuit depth of *Base*. At each iteration, LSL algorithm only picks two different elements from *Base* and generates a new node that will never be out of depth bound.

However, there are some noteworthy issues in the above algorithms with circuit depth awareness. Based on the LSL algorithm, we always choose the pair of nodes in *Base* which can reduce the maximum distance. In other words, if one pair cannot reduce the distance or it only reduces fewer distance, it is the *bad* choice, and may never be chosen. We surprisingly find that the known *forward* search approaches for low-latency always omit some good implementations. While *backward* approach can cover them. Of course, our approach also abandons the choice that appeared to be not *good* in the current state. It might miss good implementations since it cannot achieve the exhaustive search. Nevertheless, from the experimental results, this new strategy enables us to find better implementations in multiple cases.

We take a more comprehensive example to illustrate it. For the matrix M_C used in Camellia [AIK+00],

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix},$$

we provide two implementations. The first is generated by LSL algorithm with 20 XOR gates produced. Table 5 shows the depth, new nodes and *dist*. It can be seen that the distance reduces faster in the first half of the execution, which is related to the heuristic rules. Next, we use our *backward* framework to generate the implementation, which is shown in Table 6 and only needs 19 XOR gates, thanks to our new strategy.

For LSL algorithm, the *Base* of M_C is $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$, and the $Dist = [5,5,5,5,4,4,4,4]$. The initial distance is $\sum_{i=0}^{7} Dist[i] = 36$. We have four choices $\{x_0 \oplus x_5, x_1 \oplus x_7, x_2 \oplus x_4, x_3 \oplus x_6\}$ which can reduce the distance 4 continuously (see Table 5). Now, we consider a new choice:

$$x_0 \oplus x_3.$$

Only y_0, y_1, and y_7 are included this combination. Meanwhile, $x_3 \oplus x_6$ is also included in y_0, y_1, y_7 and y_3. In this case, the priority of $x_0 \oplus x_3$ is always lower than $x_3 \oplus x_6$. We will explain that $x_0 \oplus x_3$ will never be chosen in M_C by LSL algorithm. After choosing $x_3 \oplus x_6$, the distance is 32. The distance of y_0, y_1, y_3, y_7 is reduced. When performing the next XOR, $x_0 \oplus x_3$ will not reduce any distance. $x_3 \oplus x_6$ limits the effect of $x_0 \oplus x_3$. Actually, no heuristics based on BP algorithm will choose this choice even though it leads to fewer XOR gates.

Our framework can avoid such an issue. For a target node y, we calculate its minimum depth d_y, and choose predecessors from $L_{d_y - 1}$. Thus, the choice $x_0 \oplus x_3$ can be used.

Moreover, there is another good feature of our algorithm from that of LSL. A reusable node means that we need not generate it again. This kind of node can be found by *backward* framework easily based on our splitting Rule 2 and Rule 3. It hopes that fewer nodes are used to split nodes in the working set $\mathcal{W}$. While LSL algorithm does not pay attention to the above feature. It tries to find the nodes which make the *Base* closer to the target. It is not to say that LSL algorithm will not reuse nodes. The difference may explain why for some matrices, our algorithm performs well, for others, we do not. For example, in M_C, $t_8 = [0,1,0,0,1,0,1,1]$ is more suitable for the implementation. In Table 6, t_8 is used four times. Therefore, we use only four nodes in L_2 to split all the target nodes. There are six nodes in L_2 in Table 5.

Table 5: The implementation of M_C using LSL algorithm

No.	Operation	Depth	New Node	New Dist
0	$t_1 = x_0 \oplus x_5$	1	$t_1 = [1,0,0,0,0,1,0,0]$	$[\mathbf{4},5,\mathbf{4},5,\mathbf{3},4,4,\mathbf{3}] = 32$
1	$t_2 = x_1 \oplus x_7$	1	$t_2 = [0,1,0,0,0,0,0,1]$	$[4,\mathbf{4},\mathbf{3},5,\mathbf{2},\mathbf{3},4,3] = 28$
2	$t_3 = x_2 \oplus x_4$	1	$t_3 = [0,0,1,0,1,0,0,0]$	$[4,4,\mathbf{2},\mathbf{4},2,\mathbf{2},\mathbf{3},3] = 24$
3	$t_4 = x_3 \oplus x_6$	1	$t_4 = [0,0,0,1,0,0,1,0]$	$[\mathbf{3},\mathbf{3},2,\mathbf{3},2,2,3,\mathbf{2}] = 20$
4	$t_5 = x_6 \oplus t_2$	2	$t_5 = [0,1,0,0,0,0,1,1]$	$[3,3,2,3,\mathbf{1},\mathbf{1},3,2] = 18$
5	$t_6 = t_1 \oplus t_5 // y_4$	3	$t_6 = [1,1,0,0,0,1,1,1]$	$[3,3,2,3,\mathbf{0},1,3,2] = 17$
6	$t_7 = t_3 \oplus t_5 // y_5$	3	$t_7 = [0,1,1,0,1,0,1,1]$	$[3,3,2,3,0,\mathbf{0},3,2] = 16$
7	$t_8 = x_4 \oplus t_4$	2	$t_8 = [0,0,0,1,1,0,1,0]$	$[3,\mathbf{2},2,3,0,0,3,\mathbf{1}] = 14$
8	$t_9 = t_1 \oplus t_8 // y_7$	3	$t_9 = [1,0,0,1,1,1,1,0]$	$[3,2,2,3,0,0,3,\mathbf{0}] = 13$
9	$t_{10} = x_5 \oplus t_3$	2	$t_{10} = [0,0,1,0,1,1,0,0]$	$[3,2,2,\mathbf{2},0,0,\mathbf{2},0] = 11$
10	$t_{11} = x_0 \oplus t_2$	2	$t_{11} = [1,1,0,0,0,0,0,1]$	$[3,\mathbf{1},\mathbf{1},2,0,0,2,0] = 9$
11	$t_{12} = t_8 \oplus t_{11} // y_1$	3	$t_{12} = [1,1,0,1,1,0,1,1]$	$[3,\mathbf{0},1,2,0,0,2,0] = 8$
12	$t_{13} = t_{10} \oplus t_{11} // y_2$	3	$t_{13} = [1,1,1,0,1,1,0,1]$	$[3,0,\mathbf{0},2,0,0,2,0] = 7$
13	$t_{14} = x_1 \oplus t_4$	2	$t_{14} = [0,1,0,1,0,0,1,0]$	$[3,0,0,\mathbf{1},0,0,2,0] = 6$
14	$t_{15} = t_{10} \oplus t_{14} // y_3$	3	$t_{15} = [0,1,1,1,1,1,1,0]$	$[3,0,0,\mathbf{0},0,0,2,0] = 5$
15	$t_{16} = x_3 \oplus x_7$	1	$t_{16} = [0,0,0,1,0,0,0,1]$	$[3,0,0,0,0,0,\mathbf{1},0] = 4$
16	$t_{17} = t_{10} \oplus t_{16} // y_6$	3	$t_{17} = [0,0,1,1,1,1,0,1]$	$[3,0,0,0,0,0,\mathbf{0},0] = 3$
17	$t_{18} = x_2 \oplus x_6$	1	$t_{18} = [0,0,1,0,0,0,1,0]$	$[\mathbf{2},0,0,0,0,0,0,0] = 2$
18	$t_{19} = t_1 \oplus t_{16}$	2	$t_{19} = [1,0,0,1,0,1,0,1]$	$[\mathbf{1},0,0,0,0,0,0,0] = 1$
19	$t_{20} = t_{18} \oplus t_{19} // y_0$	3	$t_{20} = [1,0,1,1,0,1,1,1]$	$[\mathbf{0},0,0,0,0,0,0,0] = 0$

Table 6: The implementation of M_C using our framework

No.	Operation	Depth	New Node	New Dist
0	$t_{12} = x_3 \oplus x_6$	1	$t_{12} = [0,0,0,1,0,0,1,0]$	$[\mathbf{4},\mathbf{4},5,\mathbf{4},4,4,4,\mathbf{3}] = 32$
1	$t_9 = x_0 \oplus x_3$	1	$t_9 = [1,0,0,1,0,0,0,0]$	$[4,4,5,4,4,4,4,3] = 32$
2	$t_{18} = x_3 \oplus x_5$	1	$t_{18} = [0,0,0,1,0,1,0,0]$	$[4,4,5,4,4,4,\mathbf{3},3] = 31$
3	$t_{17} = x_2 \oplus x_7$	1	$t_{17} = [0,0,1,0,0,0,0,1]$	$[\mathbf{3},4,\mathbf{4},4,4,\mathbf{3},\mathbf{2},3] = 27$
4	$t_{14} = x_1 \oplus x_4$	1	$t_{14} = [0,1,0,0,1,0,0,0]$	$[3,\mathbf{3},\mathbf{3},\mathbf{3},4,\mathbf{2},2,3] = 23$
5	$t_{16} = x_0 \oplus x_5$	1	$t_{16} = [1,0,0,0,0,1,0,0]$	$[\mathbf{2},3,\mathbf{2},3,\mathbf{3},2,2,\mathbf{2}] = 19$
6	$t_{15} = x_6 \oplus x_7$	1	$t_{15} = [0,0,0,0,0,0,1,1]$	$[2,\mathbf{2},2,3,\mathbf{2},2,2,2] = 17$
7	$t_{10} = t_{16} \oplus x_4$	2	$t_{10} = [1,0,0,0,1,1,0,0]$	$[2,2,2,3,2,2,2,\mathbf{1}] = 16$
8	$t_{13} = t_{16} \oplus t_{17}$	2	$t_{13} = [1,0,1,0,0,1,0,1]$	$[\mathbf{1},2,\mathbf{1},3,2,2,2,1] = 14$
9	$t_{11} = t_{17} \oplus t_{18}$	2	$t_{11} = [0,0,1,1,0,1,0,1]$	$[1,2,1,\mathbf{2},2,2,\mathbf{1},1] = 12$
10	$t_8 = t_{14} \oplus t_{15}$	2	$t_8 = [0,1,0,0,1,0,1,1]$	$[1,\mathbf{1},1,\mathbf{1},\mathbf{1},\mathbf{1},1,1] = 8$
11	$y_4 = t_8 \oplus t_{10}$	3	$y_4 = [1,1,0,0,0,1,1,1]$	$[1,1,1,1,\mathbf{0},1,1,1] = 7$
12	$y_7 = t_{10} \oplus t_{12}$	3	$y_7 = [1,0,0,1,1,1,1,0]$	$[1,1,1,1,0,1,1,\mathbf{0}] = 6$
13	$y_0 = t_{13} \oplus t_{12}$	3	$y_0 = [1,0,1,1,0,1,1,1]$	$[\mathbf{0},1,1,1,0,1,1,0] = 5$
14	$y_2 = t_{13} \oplus t_{14}$	3	$y_2 = [1,1,1,0,1,1,0,1]$	$[0,1,\mathbf{0},1,0,1,1,0] = 4$
15	$y_3 = t_8 \oplus t_{11}$	3	$y_3 = [0,1,1,1,1,1,1,0]$	$[0,1,0,\mathbf{0},0,1,1,0] = 3$
16	$y_6 = x_4 \oplus t_{11}$	3	$y_6 = [0,0,1,1,1,1,0,1]$	$[0,1,0,0,0,1,\mathbf{0},0] = 2$
17	$y_5 = t_8 \oplus x_2$	3	$y_5 = [0,1,1,0,1,0,1,1]$	$[0,1,0,0,0,\mathbf{0},0,0] = 1$
18	$y_1 = t_8 \oplus t_9$	3	$y_1 = [1,1,0,1,1,0,1,1]$	$[0,\mathbf{0},0,0,0,0,0,0] = 0$

4 Applications

4.1 XOR Gates of Many Proposed Matrices

In this subsection, we first apply our algorithm to several linear layers from the literature including

- matrices that have been already used in many ciphers [DR20, CMR05, JNP15, Ava17, BBI+15, BCG+12, ADK+14, Ava17, BJK+16, AIK+00], and
- matrices that are independently proposed in many previous works [SKOP15, LS16, LW16, BKL16, SS16, JPST17, KLSW17].

All the results are listed in Table 2. For comparison, we also list the results from [KLSW17], LSL algorithm [LSL+19], and BFI algorithm [BFI21]. The last two are based on BP algorithm and thus use the *forward* strategy. Experimental results reveal that our algorithm can always find the minimum depth implementations and generally outperforms LSL and BFI algorithm in many cases. For the results provided by Kranz et al., we can get some implementations with lower latency. We takes 12 hours and 5 days to run our algorithm for each 16×16 and 32×32 binary matrix respectively. The time is for one matrix. Notably, on 16×16 binary matrices, we find many better results, indicating that even in small-scale cases, there is still a room for improvements with low latency. Particularly, for AES Mixcolumns, we achieve the implementation with 103 XOR gates with a depth of 3, which is equal to the result from [BFI21], while the other best previous result is 105 from [LSL+19]. This shows the effectiveness of our strategy. We show the 103-XOR implementation in Table 7, where $x_0, x_1, x_2, ..., x_{31}$ are the 32 inputs and $y_0, y_1, y_2, ..., y_{31}$ are the 32 outputs. The above implementation has a lower power consumption than those in [LSL+19] and [XZL+20]. We will introduce the details in Subsection 4.3.

Then, we apply our algorithm to other matrices with low latency. Duval and Leurent proposed many matrices with different depths in [DL18]. As [LSL+19] emphasized, for the matrices with branch number 5 in $\mathbf{M}_4(\mathrm{GL}(8, \mathbb{F}_2))$, the minimum depth is 3. Actually, the minimum depth of matrices in $\mathbf{M}_4(\mathrm{GL}(4, \mathbb{F}_2))$ with branch number 5 is also 3. Therefore, we test all of the matrices with depth 3 (see Table 3). The column "Const" is from [DL18]. Duval and Leurent constructed the matrices by choosing optimal paths in order to minimize the number of XOR required. BP, Paar2, RSDP, RNBP, A1, A2 are different algorithms to find optimistic implementations of matrices. Notably, we get a better implementation for matrix $M_{4,3}^{9,5}$ with A_4. It only needs 40 XOR gates with depth 3.

4.2 XOR Gates of More MDS Matrices

As Li et al. proposed a new algorithm and many MDS matrices [LSL+19], we also apply our algorithm to these matrices for comparison. There are 4256 matrices with depth 3, which reach the minimal depth, and we only consider them in our experiments. We ran the algorithm 60 minutes for each matrix and compare the obtained implementation with the results from [LSL+19]. It is listed in Table 8.

From the results, we can see that the results from LSL algorithm has a significant room for improvement. We find that about 54.3% matrices have fewer XOR gates with the minimum depth. The maximum number of XOR gates we can reduce is 12 (from 98 to 86 XOR gates). Meanwhile, we check all the implementations and find some interesting results. In [LSL+19], the minimal implementation with depth 3 needs 88 XOR gates. Using our algorithm, some matrices can be implemented by 86 XOR gates with the same depth. We show a representative matrix that we call matrix R. It is the 483-*rd* matrix with Hamming weight 168, generated by the parameter $[6, 10, 4, -4, -2, -6]$ from [LSL+19] using 98 XOR gates. We implement matrix R with only 86 XOR gates with depth 3. The

Table 7: An implementation of AES MixColumns with 103 XOR operations

No.	Operation	Depth	No.	Operation	Depth
0	$t_{32} = x_5 \oplus x_{13}$	1	52	$t_{49} = t_{85} \oplus t_{86}$	2
1	$t_{43} = x_{21} \oplus x_{29}$	1	53	$t_{87} = x_2 \oplus x_{10}$	1
2	$t_{44} = x_{15} \oplus x_{30}$	1	54	$t_{55} = t_{87} \oplus x_{25}$	2
3	$t_{46} = x_7 \oplus x_{16}$	1	55	$t_{18} = t_{55} \oplus t_{57} // y_{18}$	3
4	$t_{47} = x_{23} \oplus x_{24}$	1	56	$t_{26} = t_{55} \oplus t_{56} // y_{26}$	3
5	$t_{56} = x_1 \oplus x_{18}$	1	57	$t_{88} = x_3 \oplus x_{26}$	1
6	$t_{57} = x_{17} \oplus x_{26}$	1	58	$t_{66} = t_{88} \oplus x_{31}$	2
7	$t_{70} = x_6 \oplus x_{22}$	1	59	$t_{19} = t_{69} \oplus t_{66} // y_{19}$	3
8	$t_{35} = t_{70} \oplus t_{43}$	2	60	$t_{89} = x_{12} \oplus x_{27}$	1
9	$t_{71} = x_{14} \oplus x_{31}$	1	61	$t_{61} = t_{89} \oplus x_{31}$	2
10	$t_{42} = t_{71} \oplus t_{70}$	2	62	$t_{28} = t_{64} \oplus t_{61} // y_{28}$	3
11	$t_{72} = x_7 \oplus x_{15}$	1	63	$t_{90} = x_8 \oplus x_{15}$	1
12	$t_{37} = t_{72} \oplus t_{71}$	2	64	$t_{48} = x_{24} \oplus t_{90}$	2
13	$t_{73} = x_0 \oplus x_{17}$	1	65	$t_0 = t_{48} \oplus t_{46} // y_0$	3
14	$t_{51} = x_7 \oplus t_{73}$	2	66	$t_8 = t_{49} \oplus t_{48} // y_8$	3
15	$t_{74} = x_6 \oplus x_{23}$	1	67	$t_{91} = x_9 \oplus x_{25}$	1
16	$t_{39} = x_7 \oplus t_{74}$	2	68	$t_{53} = t_{90} \oplus t_{91}$	2
17	$t_7 = t_{39} \oplus t_{37} // y_7$	3	69	$t_1 = t_{53} \oplus t_{51} // y_1$	3
18	$t_{15} = t_{42} \oplus t_{39} // y_{15}$	3	70	$t_{92} = x_1 \oplus x_{17}$	1
19	$t_{31} = t_{44} \oplus t_{39} // y_{31}$	3	71	$t_{54} = t_{90} \oplus t_{92}$	2
20	$t_{75} = x_{12} \oplus x_{28}$	1	72	$t_9 = t_{54} \oplus t_{52} // y_9$	3
21	$t_{76} = x_3 \oplus x_7$	1	73	$t_{93} = x_4 \oplus x_{28}$	1
22	$t_{63} = t_{75} \oplus t_{76}$	2	74	$t_{33} = t_{93} \oplus x_{21}$	2
23	$t_{77} = x_{13} \oplus x_{29}$	1	75	$t_5 = t_{36} \oplus t_{33} // y_5$	3
24	$t_{36} = t_{75} \oplus t_{77}$	2	76	$t_{29} = t_{32} \oplus t_{33} // y_{29}$	3
25	$t_{38} = x_{30} \oplus t_{77}$	2	77	$t_{94} = x_{19} \oplus x_{23}$	1
26	$t_{14} = t_{38} \oplus t_{35} // y_{14}$	3	78	$t_{60} = t_{93} \oplus t_{94}$	2
27	$t_{78} = x_{14} \oplus x_{22}$	1	79	$t_{20} = t_{60} \oplus t_{61} // y_{20}$	3
28	$t_{34} = t_{78} \oplus x_{30}$	2	80	$t_{95} = x_{10} \oplus x_{26}$	1
29	$t_6 = t_{32} \oplus t_{34} // y_6$	3	81	$t_{59} = x_9 \oplus t_{95}$	2
30	$t_{22} = t_{35} \oplus t_{34} // y_{22}$	3	82	$t_2 = t_{59} \oplus t_{56} // y_2$	3
31	$t_{23} = t_{37} \oplus t_{34} // y_{23}$	3	83	$t_{96} = x_2 \oplus x_{18}$	1
32	$t_{79} = x_4 \oplus x_{20}$	1	84	$t_{58} = x_9 \oplus t_{96}$	2
33	$t_{64} = t_{79} \oplus t_{76}$	2	85	$t_{10} = t_{58} \oplus t_{57} // y_{10}$	3
34	$t_{80} = x_{12} \oplus x_{20}$	1	86	$t_{97} = x_{10} \oplus x_{27}$	1
35	$t_{41} = x_5 \oplus t_{80}$	2	87	$t_{67} = x_{15} \oplus t_{97}$	2
36	$t_{13} = t_{43} \oplus t_{41} // y_{13}$	3	88	$t_{11} = t_{68} \oplus t_{67} // y_{11}$	3
37	$t_{21} = t_{41} \oplus t_{36} // y_{21}$	3	89	$t_{98} = x_{11} \oplus x_{20}$	1
38	$t_{81} = x_{14} \oplus x_{21}$	1	90	$t_{62} = x_{15} \oplus t_{98}$	2
39	$t_{40} = x_5 \oplus t_{81}$	2	91	$t_4 = t_{63} \oplus t_{62} // y_4$	3
40	$t_{30} = t_{40} \oplus t_{35} // y_{30}$	3	92	$t_{12} = t_{60} \oplus t_{62} // y_{12}$	3
41	$t_{82} = x_{18} \oplus x_{23}$	1	93	$t_{99} = x_{11} \oplus x_{19}$	1
42	$t_{83} = x_{11} \oplus x_{27}$	1	94	$t_{100} = x_2 \oplus x_7$	1
43	$t_{69} = t_{82} \oplus t_{83}$	2	95	$t_{65} = t_{99} \oplus t_{100}$	2
44	$t_{84} = x_3 \oplus x_{19}$	1	96	$t_3 = t_{65} \oplus t_{67} // y_3$	3
45	$t_{68} = t_{82} \oplus t_{84}$	2	97	$t_{27} = t_{65} \oplus t_{66} // y_{27}$	3
46	$t_{85} = x_{16} \oplus x_{23}$	1	98	$t_{101} = x_9 \oplus x_{31}$	1
47	$t_{52} = t_{85} \oplus x_{25}$	2	99	$t_{102} = x_1 \oplus x_{24}$	1
48	$t_{86} = x_0 \oplus x_8$	1	100	$t_{50} = t_{101} \oplus t_{102}$	2
49	$t_{45} = x_{31} \oplus t_{86}$	2	101	$t_{17} = t_{50} \oplus t_{52} // y_{17}$	3
50	$t_{16} = t_{45} \oplus t_{47} // y_{16}$	3	102	$t_{25} = t_{50} \oplus t_{51} // y_{25}$	3
51	$t_{24} = t_{45} \oplus t_{46} // y_{24}$	3			

implementation is shown in Table 10. R has three advantages: it is involutory; its depth is 3; and its area footprint and power consumption are lower than AES Mixcolumns.

Table 8: The optimized results of matrices with depth limitation from [LSL+19]

HW[a]	**Size**	**Depth**	**Number of matrices**	**Optimizations**[b]	**Maximum**[c]
148	32	3	18	0	0
149	32	3	48	0	0
150	32	3	72	0	0
151	32	3	48	0	0
152	32	3	60	**3**	1
153	32	3	72	0	0
154	32	3	84	0	0
155	32	3	24	**6**	2
156	32	3	48	**6**	1
157	32	3	72	**16**	2
158	32	3	84	**34**	3
160	32	3	162	**114**	6
161	32	3	96	**88**	6
162	32	3	132	**108**	6
163	32	3	120	**95**	8
164	32	3	144	**97**	7
165	32	3	240	**207**	9
166	32	3	228	**190**	9
167	32	3	218	**153**	8
168	32	3	528	**355**	12
169	32	3	360	**233**	9
170	32	3	432	**246**	7
171	32	3	432	**182**	10
172	32	3	534	**177**	12
Total	-	-	**4256**	**2310**	-

[a] The Hamming weight of matrices.
[b] The number of matrices that have fewer XOR gates than the results from [LSL+19].
[c] It represents the maximum number of reduced XOR gates.

4.3 Hardware Implementation

Our algorithm aims at finding optimized implementation in circuit size, power consumption, and latency. In this subsection, we synthesize existing implementations and show their performance in hardware. We first provide the implementations of AES Mixcolumns. Through three metrics (area, power, and latency), we can discuss which is the better implementation. The AES results are synthesized using two different ASIC libraries, namely TSMC 90 nm and NanGate 45 nm (in Table 9). The logic synthesis is performed with Synopsys Design Compiler version D-2010.03-SP1(using the compile_ultra -no_autoungroup command), and simulation is done in Mentor Graphics ModelSim SE v10.2c. From the above tables, we can find our AES Mixcolumns result has more advantages than the results from LSL and BFI algorithms in hardware, and ours has less power and latency than the results from [XZL+20] and [LXZZ21], which is crucial in devices with limited resources. Besides, we synthesize the result about R using the ASIC library NanGate 45 nm. Results are also shown in Table 9. Notably, the results are better than the results from [LSL+19] and [BFI21]. As the best matrix found by [LSL+19] with 88 XOR gates and depth 3 needs 176 GE while R we find has 172 GE.

Table 9: Synthesized results using two different ASIC libraries

Results of AES MixColumns for TSMC 90 nm library					
Result	**Cost**	**Depth**	**Area**(GE)	**Power**(uw)	**Latency**(ns)
[LXZZ21]	91	7	182	135.3	0.45
[XZL+20]	92	6	184	130.0	0.38
[LSL+19]	105	3	210	124.4	0.22
[BFI21]	103	3	206	122.6	0.23
Ours	**103**	**3**	**206**	**116.3**	**0.21**
Results of AES MixColumns for NanGate 45 nm library					
Result	**Cost**	**Depth**	**Area**(GE)	**Power**(uw)	**Latency**(ns)
[LXZZ21]	91	7	182	142.5	0.42
[XZL+20]	92	6	184	139.3	0.37
[LSL+19]	105	3	210	127.0	0.22
[BFI21]	103	3	206	125.5	0.22
Ours	**103**	**3**	**206**	**122.1**	**0.20**
Result of R for NanGate 45 nm library					
Result	**Cost**	**Depth**	**Area**(GE)	**Power**(uw)	**Latency**(ns)
Ours	**86**	**3**	**172**	**101.6**	**0.20**

5 Conclusion

In this paper, we investigate a new framework of heuristic search for the implementation of a given linear layer. Our new approach takes the strategy that iteratively splits the output bits until all the input bits appear, which is very suitable to the low-latency criteria. Our new framework contributes to

- an implementation of AES Mixcolumns with 103 XOR gates with a depth of 3, which is among the best hardware implementations of AES linear layer with minimum depth;
- better implementations for 54.3% of matrices proposed in [LSL+19], in which we find an involutory MDS with fewer XOR gates (i.e., 86, saving 2 from the state-of-the-art result) of XORs with minimum depth.

Though *backward* framework can solve the low-latency problem easily, it is still important to further reduce the number of XOR gates without any constraints (i.e. no depth limitation). In addition, our research provides a new tool for the construction of lightweight MDS matrices. There should exist some matrices more compatible with our algorithm and thus have better implementations with minimum depth, which we leave as a promising future work.

Acknowledgments

The authors would like to thank the anonymous reviewers and the shepherd for their valuable comments and suggestions to improve the quality of the paper. The research leading to these results has received funding from the National Natural Science Foundation of China (Grant No. 62032014, Grant No. 62002201, Grant No. 62002204), the National Key Research and Development Program of China (Grant No. 2018YFA0704702, Grant No. 2021YFA1000600), the Major Basic Research Project of Natural Science Foundation of Shandong Province, China (Grant No. ZR202010220025), and the Program of Qilu Young Scholars (Grant No. 61580082063088) of Shandong University.

References

[ADK$^+$14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. Block ciphers - focus on the linear layer (feat. PRIDE). In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2014.

[AIK$^+$00] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-bit block cipher suitable for multiple platforms - design and analysis. In Douglas R. Stinson and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000, Proceedings*, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2000.

[ARVV18] Elena Andreeva, Reza Reyhanitabar, Kerem Varici, and Damian Vizár. Forking a blockcipher for authenticated encryption of very short messages. *IACR Cryptol. ePrint Arch.*, 2018:916, 2018.

[Ava17] Roberto Avanzi. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.

[BBI$^+$15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.

[BCG$^+$12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BFI19] Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. More results on shortest linear programs. In Nuttapong Attrapadung and Takeshi Yagi, editors, *Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28-30, 2019, Proceedings*, volume 11689 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2019.

[BFI21] Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. Further results on efficient implementations of block cipher linear layers. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 104-A(1):213–225, 2021.

[BJK$^+$16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[BKL$^+$07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[BKL16] Christof Beierle, Thorsten Kranz, and Gregor Leander. Lightweight multiplication in gf(2^n) with applications to MDS matrices. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 625–653. Springer, 2016.

[BMP08] Joan Boyar, Philip Matthews, and René Peralta. On the shortest linear straight-line program for computing linear forms. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 168–179. Springer, 2008.

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

[BP10] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In Paola Festa, editor, *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings*, volume 6049 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2010.

[BPP$^+$17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 321–345. Springer, 2017.

[BSS$^+$13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. *IACR Cryptol. ePrint Arch.*, 2013:404, 2013.

[CMR05] Carlos Cid, Sean Murphy, and Matthew J. B. Robshaw. Small scale variants of the AES. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 145–162. Springer, 2005.

[DGB19] Indira Kalyan Dutta, Bhaskar Ghosh, and Magdy A. Bayoumi. Lightweight cryptography for internet of insecure things: A survey. In *IEEE 9th Annual Computing and Communication Workshop and Conference, CCWC 2019, Las Vegas, NV, USA, January 7-9, 2019*, pages 475–481. IEEE, 2019.

[DL18] Sébastien Duval and Gaëtan Leurent. MDS matrices with lightweight circuits. *IACR Trans. Symmetric Cryptol.*, 2018(2):48–78, 2018.

[DR20] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition.* Information Security and Cryptography. Springer, 2020.

[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.

[KHZ17] Martin Kumm, Martin Hardieck, and Peter Zipf. Optimization of constant matrix multiplication with low power and high throughput. *IEEE Trans. Computers*, 66(12):2072–2080, 2017.

[KLSW17] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Shorter linear straight-line programs for MDS matrices. *IACR Trans. Symmetric Cryptol.*, 2017(4):188–211, 2017.

[KPPY14] Khoongming Khoo, Thomas Peyrin, Axel York Poschmann, and Huihui Yap. FOAM: searching for hardware-optimal SPN structures and components with a fair comparison. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 433–450. Springer, 2014.

[LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):510–545, 2021.

[LS16] Meicheng Liu and Siang Meng Sim. Lightweight MDS generalized circulant matrices. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 101–120. Springer, 2016.

[LSL$^+$19] Shun Li, Siwei Sun, Chaoyun Li, Zihao Wei, and Lei Hu. Constructing low-latency involutory MDS matrices with lightweight circuits. *IACR Trans. Symmetric Cryptol.*, 2019(1):84–117, 2019.

[LW16] Yongqiang Li and Mingsheng Wang. On the construction of lightweight circulant involutory MDS matrices. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2016.

[LXZZ21] Da Lin, Zejun Xiang, Xiangyong Zeng, and Shasha Zhang. A framework to optimize implementations of matrices. In Kenneth G. Paterson, editor, *Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings*, volume 12704 of *Lecture Notes in Computer Science*, pages 609–632. Springer, 2021.

[Paa97] Christof Paar. Optimized arithmetic for reed-solomon encoders. In *Proceedings of IEEE International Symposium on Information Theory*, page 250. IEEE, 1997.

[RTA18] Arash Reyhani-Masoleh, Mostafa M. I. Taha, and Doaa Ashmawy. Smashing the implementation records of AES s-box. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):298–336, 2018.

[SKOP15] Siang Meng Sim, Khoongming Khoo, Frédérique E. Oggier, and Thomas Peyrin. Lightweight MDS involution matrices. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 471–493. Springer, 2015.

[SS16] Sumanta Sarkar and Habeeb Syed. Lightweight diffusion layer: Importance of toeplitz matrices. *IACR Trans. Symmetric Cryptol.*, 2016(1):95–113, 2016.

[TP19] Quan Quan Tan and Thomas Peyrin. Improved heuristics for short linear programs. *IACR Cryptol. ePrint Arch.*, page 847, 2019.

[VSP18] Andrea Visconti, Chiara Valentina Schiavo, and René Peralta. Improved upper bounds for the expected circuit complexity of dense systems of linear equations over GF(2). *Inf. Process. Lett.*, 137:1–5, 2018.

[WP13] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 185–201. Springer, 2013.

[XZL$^+$20] Zejun Xiang, Xiangyong Zeng, Da Lin, Zhenzhen Bao, and Shasha Zhang. Optimizing implementations of linear layers. *IACR Trans. Symmetric Cryptol.*, 2020(2):120–145, 2020.

[ZBL$^+$15] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. *Sci. China Inf. Sci.*, 58(12):1–15, 2015.

A The Low-latency Implementation

Table 10: The implementation of R with 86 XOR operations and depth of 3

No.	Operation	Depth	No.	Operation	Depth
0	$t_{42} = x_1 \oplus x_{25}$	1	43	$t_{25} = t_{42} \oplus t_{39} // y_{25}$	3
1	$t_{43} = x_0 \oplus x_{24}$	1	44	$t_{69} = x_{19} \oplus x_{27}$	1
2	$t_{45} = x_{20} \oplus x_{26}$	1	45	$t_{19} = t_{68} \oplus t_{69} // y_{19}$	2
3	$t_{48} = x_{21} \oplus x_{27}$	1	46	$t_{70} = x_7 \oplus x_{31}$	1
4	$t_{52} = x_1 \oplus x_{15}$	1	47	$t_{35} = t_{70} \oplus x_{15}$	2
5	$t_{15} = t_{52} \oplus t_{48} // y_{15}$	2	48	$t_{71} = x_{15} \oplus x_{23}$	1
6	$t_{53} = x_9 \oplus x_{17}$	1	49	$t_{23} = t_{71} \oplus t_{70} // y_{23}$	2
7	$t_{17} = t_{53} \oplus t_{42} // y_{17}$	2	50	$t_{29} = t_{33} \oplus t_{23} // y_{29}$	3
8	$t_{54} = x_8 \oplus x_{16}$	1	51	$t_{72} = x_0 \oplus x_8$	1
9	$t_{16} = t_{54} \oplus t_{43} // y_{16}$	2	52	$t_{73} = x_6 \oplus x_{18}$	1
10	$t_{55} = x_0 \oplus x_{14}$	1	53	$t_{32} = t_{72} \oplus t_{73}$	2
11	$t_{14} = t_{55} \oplus t_{45} // y_{14}$	2	54	$t_6 = t_{32} \oplus t_{14} // y_6$	3
12	$t_{56} = x_5 \oplus x_{23}$	1	55	$t_{74} = x_{15} \oplus x_{19}$	1
13	$t_{57} = x_{17} \oplus x_{31}$	1	56	$t_{75} = x_{13} \oplus x_{25}$	1
14	$t_{40} = t_{56} \oplus t_{57}$	2	57	$t_{37} = t_{74} \oplus t_{75}$	2
15	$t_{58} = x_{11} \oplus x_{13}$	1	58	$t_5 = t_{40} \oplus t_{37} // y_5$	3
16	$t_{36} = t_{56} \oplus t_{58}$	2	59	$t_{13} = t_{37} \oplus t_{35} // y_{13}$	3
17	$t_{27} = t_{48} \oplus t_{36} // y_{27}$	3	60	$t_{76} = x_{22} \oplus x_{28}$	1
18	$t_{59} = x_{13} \oplus x_{29}$	1	61	$t_{77} = x_0 \oplus x_2$	1
19	$t_{33} = t_{59} \oplus t_{57}$	2	62	$t_{47} = t_{76} \oplus t_{77}$	2
20	$t_{11} = t_{36} \oplus t_{33} // y_{11}$	3	63	$t_0 = t_{47} \oplus t_{41} // y_0$	3
21	$t_{60} = x_5 \oplus x_{21}$	1	64	$t_8 = x_8 \oplus t_{47} // y_8$	3
22	$t_{21} = t_{60} \oplus t_{59} // y_{21}$	2	65	$t_{78} = x_{22} \oplus x_{30}$	1
23	$t_{61} = x_{12} \oplus x_{28}$	1	66	$t_{79} = x_6 \oplus x_{14}$	1
24	$t_{62} = x_4 \oplus x_{20}$	1	67	$t_{22} = t_{78} \oplus t_{79} // y_{22}$	2
25	$t_{20} = t_{61} \oplus t_{62} // y_{20}$	2	68	$t_{28} = t_{38} \oplus t_{22} // y_{28}$	3
26	$t_{63} = x_{16} \oplus x_{30}$	1	69	$t_{46} = x_{30} \oplus t_{79}$	2
27	$t_{38} = t_{61} \oplus t_{63}$	2	70	$t_{30} = t_{46} \oplus t_{32} // y_{30}$	3
28	$t_{64} = x_4 \oplus x_{22}$	1	71	$t_{80} = x_3 \oplus x_{23}$	1
29	$t_{51} = t_{64} \oplus t_{63}$	2	72	$t_{81} = x_1 \oplus x_{29}$	1
30	$t_{65} = x_{10} \oplus x_{12}$	1	73	$t_{49} = t_{80} \oplus t_{81}$	2
31	$t_{44} = t_{65} \oplus t_{64}$	2	74	$t_1 = t_{49} \oplus t_{39} // y_1$	3
32	$t_{10} = t_{44} \oplus t_{38} // y_{10}$	3	75	$t_9 = x_9 \oplus t_{49} // y_9$	3
33	$t_{26} = t_{45} \oplus t_{44} // y_{26}$	3	76	$t_{82} = x_{18} \oplus x_{24}$	1
34	$t_{66} = x_2 \oplus x_{10}$	1	77	$t_{83} = x_{12} \oplus x_{14}$	1
35	$t_{41} = t_{66} \oplus x_{20}$	2	78	$t_{50} = t_{82} \oplus t_{83}$	2
36	$t_2 = t_{41} \oplus t_{38} // y_2$	3	79	$t_4 = t_{51} \oplus t_{50} // y_4$	3
37	$t_{24} = t_{43} \oplus t_{41} // y_{24}$	3	80	$t_{12} = t_{50} \oplus t_{46} // y_{12}$	3
38	$t_{67} = x_{18} \oplus x_{26}$	1	81	$t_{84} = x_1 \oplus x_{19}$	1
39	$t_{18} = t_{66} \oplus t_{67} // y_{18}$	2	82	$t_{85} = x_7 \oplus x_9$	1
40	$t_{68} = x_3 \oplus x_{11}$	1	83	$t_{34} = t_{84} \oplus t_{85}$	2
41	$t_{39} = t_{68} \oplus x_{21}$	2	84	$t_7 = t_{34} \oplus t_{15} // y_7$	3
42	$t_3 = t_{39} \oplus t_{33} // y_3$	3	85	$t_{31} = t_{35} \oplus t_{34} // y_{31}$	3