IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2023, No. 4, pp. 489–510. DOI:10.46586/tosc.v2023.i4.489-510

A Framework with Improved Heuristics to Optimize Low-Latency Implementations of Linear Layers

Haotian Shi[1,2], Xiutao Feng[1(✉)] and Shengyuan Xu[3]

[1] Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] Department of Fundamental Courses, Shandong University of Science and Technology, Taian, 271019, China
shihaotian@amss.ac.cn, fengxt@amss.ac.cn, xushengyuanxsy@outlook.com

Abstract. In recent years, lightweight cryptography has been a hot field in symmetric cryptography. One of the most crucial problems is to find low-latency implementations of linear layers. The current main heuristic search methods include the Boyar-Peralta (BP) algorithm with depth limit and the backward search. In this paper we firstly propose two improved BP algorithms with depth limit mainly by minimizing the Euclidean norm of the new distance vector instead of maximizing it in the tie-breaking process of the BP algorithm. They can significantly increase the potential for finding better results. Furthermore, we give a new framework that combines forward search with backward search to expand the search space of implementations, where the forward search is one of the two improved BP algorithms. In the new framework, we make a minor adjustment of the priority of rules in the backward search process to enable the exploration of a significantly larger search space. As results, we find better results for the most of matrices studied in previous works. For example, we find an implementation of AES MixColumns of depth 3 with 99 XOR gates, which represents a substantial reduction of 3 XOR gates compared to the existing record of 102 XOR gates.

Keywords: Lightweight cryptography · Linear layer · Low-latency implementation · AES

1 Introduction

The design of cryptographic building blocks revolves around two primary criteria, namely confusion and diffusion. One of the most popular structures of block ciphers is the *Substitution-Permutation Network* (SPN) structure, wherein the attainment of confusion and diffusion predominantly relies on the nonlinear substitution boxes (Sboxes) and the linear permutations respectively.

In recent years, lightweight cryptography has been applied to provide security and privacy in many fields, such as Internet of Things (IoTs), wireless sensor networks and Radio-Frequency IDentification (RFID) tags. These devices have strict resource constraints in terms of circuit size, power consumption and latency. Given the growing concern for security, lightweight cryptography offers symmetric primitives that facilitate secure encryption with low resource costs.

This research was supported by National Key Research and Development Project under Grant No. 2018YFA0704705 and CAS Project for Young Scientists in Basic Research (Grant No. YSBR-035).

Received: 2023-09-01 Accepted: 2023-11-01 Published: 2023-12-08

There are typically two main research lines on lightweight cryptography. The first direction falls on the design of new ciphers using lightweight building blocks. Related work can be found in [BBI+15, BJK+16, LMMR21, BCG+12]. Additionally, considerable research has been conducted on constructing lightweight Maximum Distance Separable (MDS) matrices (see [SKOP15, DL18, LSL+19, YZW21] for an incomplete list). MDS matrices are widely utilized in linear layers due to their ability to achieve the maximum branch number.

The second direction focuses on optimizing the circuits of existing ciphers. Various automated tools have been proposed to search for efficient implementations of small-scale circuits, including SAT-based tools [Sto16] and the meet-in-the-middle search method [JPST17, BGLS19]. Synthesis tools such as Yosys [Wol16] and ABC [BM10] are available for rough optimization of circuits in larger domain sizes. These tools are capable of handling both linear and non-linear layers. Numerous papers in the literature have proposed novel techniques to efficiently implement middle-scale non-linear layers, particularly for the AES Sbox. One of the most popular techniques involves the use of composite field arithmetic, which leads to a small and compact implementation of AES Sbox [Can05].

There has also been a lot of research on optimizing implementations of linear layers. Among the various criteria used for evaluating implementations, one of the most commonly employed is the measurement of Gate Equivalent (GE)[1]. In the context of linear layers, the number of XOR gates required to compute outputs from inputs corresponds to the GE count, as the XOR gate is the only one utilized in their implementations. Therefore, reducing the XOR gate count is equivalent to reducing the GE cost, which formulates the Shortest Linear Program (SLP, in short) problem. However, the SLP problem has been proven to be NP-complete over a finite field [BMP08]. To address this challenge, various heuristic algorithms have been proposed to search for implementations with a reduced number of XOR gates. Paar's algorithm [Paa97] runs fast and provides some decent cancellation-free implementations by iteratively combining different columns in the matrix. The BP algorithm [BMP13] employs a greedy strategy on reducing distances to outputs, and can produce good results for many middle-scale matrices. Furthermore, several improvements upon the BP algorithm have been proposed in subsequent research [VSP18, ME19, TP20]. Based on existing implementations, Xiang *et al.* introduced new approaches to reduce the number of XOR gates by means of some reduction rules [XZL+20, LXZZ21]. All these methods use the general-XOR metric except the one in [XZL+20] with the sequential-XOR metric. In this paper we only consider the general-XOR metric.

Another criterion that has gained significant attention is latency, which relates to the total time required for running a circuit and can be characterized by the circuit depth, i.e., the length of the longest path involved in computing an output. Latency not only impacts the throughput of encryption/decryption, but also plays an important role in the low-energy consideration of ciphers [BBI+15]. As mentioned previously, numerous ciphers with low latency have been proposed. In terms of the second research line, for a given linear layer, its implementation can reach a certain minimum depth. Within the depth limit, it becomes important to reduce the XOR gate count of minimum latency implementations, which formulates the Shortest Linear Program with minimum Depth (SLPD, in short) problem. The aforementioned methods to address the SLP problem become ineffective since they do not take the depth limit into account. Currently, there are two primary types of algorithms for the SLPD problem. One is the forward search represented by Li *et al.*'s method [LSL+19]. They added a depth limit to the original BP algorithm. Banik *et al.* [BFI21] made an improvement by introducing randomness. The other is the backward search proposed by Liu *et al.* [LWF+22]. They iteratively split one

[1]The unit of gate size is Gate Equivalent (GE), where one GE equals the area of a 2-input NAND gate. The cost of other gates in terms of GE is a normalized ratio between their area and one NAND gate area.

or two nodes by several rules with different priorities. In [LZW23] they further proposed a framework for local re-optimization of a given low-latency implementation to address the SLPD problem.

1.1 Our contributions

This paper focuses on the SLPD problem. Notably, a significant difference between the SLPD and SLP problems lies in the fact that, in the SLPD problem, outputs and those with high depth have a smaller contribution to other outputs due to the depth limit, but this is not the case in the SLP problem. Consequently, prioritizing closer outputs has a smaller impact on approaching other outputs and may result in the loss of alternative, better pathways that generate the closer outputs. Therefore, in the SLPD problem, approaching all outputs at a relatively consistent pace might be a preferable choice due to the depth limit.

Based on the above idea, we propose two improved heuristics IBPD and IBPD-MD for the SLPD problem, which are based on the BP algorithm with depth limit. When faced with multiple choices for new base elements, which all result in new base sets with the same sum of distances to the outputs, we refine the tie-breaking rule by minimizing the Euclidean norm of the new distance vector instead of maximizing it. Our experiments further show that it significantly increases the potential for better search results. Both algorithms IBPD and IBPD-MD adopt this strategy, and the only difference between them is that IBPD-MD has an additional filtering operation on the choices. Additionally, the depth bound of some outputs can be set smaller.

Moreover, we propose a new framework of combining forward search with backward search. Specifically, the forward search is performed multiple times to optimize the nodes in the current working set and predecessor set after each iteration of the backward search. Finally, one implementation with the least cost is recorded. The forward search can be IBPD or IBPD-MD, and actually IBPD-MD performs better. Compared to the algorithm IBPD or IBPD-MD alone, the framework can produce better results, as backward search can expand the search space of forward search by predetermining how some outputs are generated. Moreover, we also find that leveling up the priority of the random splitting rule could explore a much larger expanded search space and lead to much better results.

Finally we apply our methods to various matrices studied in related works [LWF+22, LZW23]. For most matrices, better results can be found and are listed in Table 5[2]. Notably, we find a minimum latency implementation of AES MixColumns with only 99 XOR gates, which breaks the previous record of 102 XOR gates in [LZW23] and is given in Table 4. Additionally, we also apply them to 4254 involutory MDS matrices with minimum depth 3 proposed in [LSL+19] and obtain better results for 88.2% of them (see Table 6), which is much higher than those in previous works (54.3% in [LWF+22] and 77.6% in [LZW23]). In particular, we find implementations of several matrices with depth 3 and 84 XOR gates, which break the previous record of 85 XOR gates in [LZW23].

1.2 Organization

In Section 2, we introduce some notations, definitions and the metric in use. In Section 3, we present some existing heuristics for the SLPD problem. In Section 4, we propose our new methods, followed by some comparisons and discussions. All the experimental results are listed in Section 5. Finally, we conclude our work in Section 6.

[2]All the source codes and results of this paper are available at https://gitee.com/Haotian-Shi/A_framework_with_improved_heuristics_to_optimize_low-latency_implementations_of_linear_layers.

2 Preliminaries

2.1 Notations

Let $\mathbb{F}_2$ be the finite field with 2 elements 0 and 1, and $\mathbb{F}_2^n$ be the n-dimensional vector space over $\mathbb{F}_2$. The XOR operation, i.e., the addition in $\mathbb{F}_2$ or $\mathbb{F}_2^n$, is denoted by $\oplus$. Denote $wt(v)$ the Hamming weight of $v \in \mathbb{F}_2^n$, i.e., the number of 1's in v.

The linear layer of a cipher is actually a linear Boolean function with n inputs and m outputs and can be interpreted as a matrix $A = (a_{i,j})_{m\times n}$ over $\mathbb{F}_2$ (Usually $m = n$ and A is invertible). Denote the inputs of A as $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ and the outputs as $\boldsymbol{y} = (y_0, y_1, \ldots, y_{m-1})$, then $\boldsymbol{y}$ can be computed by the matrix multiplication $\boldsymbol{y}^T = A\boldsymbol{x}^T$. Therefore, the i-th row $(a_{i,0}, a_{i,1}, \ldots, a_{i,n-1})$, corresponds to the i-th output $y_i = a_{i,0}x_0 \oplus a_{i,1}x_1 \oplus \cdots \oplus a_{i,n-1}x_{n-1}$. In fact, each intermediate value $t = t_0x_0 \oplus t_1x_1 \oplus \cdots \oplus t_{n-1}x_{n-1}$ can be associated with a coefficient vector of length n:

$$t = (t_0, t_1, \ldots, t_{n-1}).$$

We use "node" to define such a binary vector. Then y_i is the i-th output node, and x_j is the j-th input node. Actually x_j is the j-th unit vector, i.e. the j-th component equals 1 and others equal 0, and for convenience we assume that every output node y is not a input node, i.e. $wt(y) > 1$.

For three nodes t_1, t_2, t_3, if t_1 is computed by $t_2 \oplus t_3$, we call that t_2 and t_3 generate t_1 and refer to them as the predecessors of t_1. The XOR computation of t_1 is said to be cancellation-free if there are no common input terms cancelled by XOR in t_2 and t_3. To generate an output node y, at the beginning the existing nodes only consist of input nodes. So y can be generated with $wt(y)$ input nodes using $wt(y) - 1$ XOR gates. It is easy to see that there exist different implementations which generate y if $wt(y) > 2$. The generation of multiple output nodes of a matrix $A_{m\times n}$ is done one by one, and this process is formally defined as an implementation of $A_{m\times n}$.

Definition 1. [XZL+20] An implementation $\mathcal{I}$ of a matrix $A_{m\times n}$ over $\mathbb{F}_2$ can be described as a sequence of nodes $\mathcal{I} = \{x_0, x_1, \cdots, x_{n+c-1}\}$ which contains all output nodes of $A_{m\times n}$ and satisfies $x_i = x_j \oplus x_k$ for any $i = n, n+1, \ldots, n+c-1$ with some $j, k < i$. It is also called a general implementation of A with XOR gate count c.

A trivial implementation needs $\sum_{i=0}^{m-1}(wt(y_i) - 1)$ XOR gates to generate all y_i's. However, since some intermediate nodes can be reused to save some XOR gates, different implementations might have different XOR gate counts for a given matrix $A_{m\times n}$. We take the following matrix A as an example, and give two of its implementations (a),(b) in Table 1.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

Table 1: Two implementations of A

Implementation (a)			Implementation (b)		
No.	**Operation**	**Depth**	**No.**	**Operation**	**Depth**
1	$x_4 = x_0 \oplus x_1 // y_2$	1	1	$x_4 = x_0 \oplus x_1 // y_2$	1
2	$x_5 = x_2 \oplus x_4 // y_1$	2	2	$x_5 = x_2 \oplus x_3$	1
3	$x_6 = x_3 \oplus x_5 // y_0$	3	3	$x_6 = x_4 \oplus x_2 // y_1$	2
			4	$x_7 = x_4 \oplus x_5 // y_0$	2

Implementation (a) in Table 1 gives an implementation of A with 3 XOR gates by reusing of x_4 and x_5. Implementation (b) in Table 1 is another implementation of A with 4 XOR gates. They have different XOR gate counts and different depths. Note that the trivial implementation of A needs 6 XOR gates.

An implementation of a matrix A can also be described as an implementation graph. All nodes of an implementation are the nodes of the graph, and both two predecessors of each non-input node have a directed edge pointing to it. Within the given implementation of A, the depth can be defined as follows:

Definition 2 (Depth)**.** Given an implementation $\mathcal{I}$ of A, the depth of a node t in $\mathcal{I}$ is defined as the length of the longest path from an input node to t in the implementation graph, denoted by $d(t)$. In particular, the depth of all input nodes is defined as 0. The depth of $\mathcal{I}$ is defined as the maximum depth of all output nodes denoted by $d(\mathcal{I})$, that is

$$d(\mathcal{I}) = \max_{0 \le i < m} d(y_i).$$

The depth of nodes follows the property:

Property 1. For three nodes t_1, t_2, t_3 in an implementation, if t_1 is generated by t_2 and t_3, then $d(t_1) = \max\{d(t_2), d(t_3)\} + 1$.

For a given node t, it may be at different depths since there can be different ways to generate it. We denote by $d_{min}(t)$ the minimum depth that t can reach among all its implementations. It is easy to see that

$$d_{min}(t) = \lceil \log_2 wt(t) \rceil.$$

Similarly, there exists a minimum depth that all A's implementations can achieve, denoted by $d_{min}(A)$. We have

$$d_{min}(A) = \max_{0 \le i < m} d_{min}(y_i),$$

which characterizes the lowest latency of A's implementations.

Definition 3. An implementation of A is called a minimum latency implementation if its depth is equal to $d_{min}(A)$.

It is easy to see that Implementation (b) in Table 1 has depth 2 and is a minimum latency implementation of A.

2.2 SLP problem and SLPD problem

The SLP problem mainly aims at minimizing the XOR gate counts of implementing a matrix, and is defined below:

Definition 4. The *s*hortest *l*inear *p*rogram (SLP) problem is defined as follows: given a matrix $A_{m \times n}$ over $\mathbb{F}_2$, where each row $y_i, 0 \le i < m$, represents an output node. The goal is to find an implementation of A using the least number of XOR gates.

Moreover, if one aims for a minimum latency implementation, the following problem arises:

Definition 5. [LZW23] The *s*hortest *l*inear *p*rogram with minimum *d*epth limit (SLPD) problem is defined as follows: given a matrix $A_{m \times n}$ over $\mathbb{F}_2$, where each row $y_i, 0 \le i < m$, represents an output node. The goal is to find a *minimum latency* implementation of A using the least number of XOR gates.

It is known from [BMP08] that the SLP problem over $\mathbb{F}_2$ is NP-complete. Various heuristics have been proposed to address the SLP and SLPD problems with middle-scale m and n [Paa97, BMP08, XZL+20, LXZZ21, LSL+19, LWF+22, LZW23]. This paper focuses on the SLPD problem.

2.3 Metric

In this paper, all the results of XOR gate counts correspond to the general-XOR metric, which is defined as follows:

Definition 6 (g-XOR)**.** [XZL+20] The general-XOR count of a matrix $A_{m\times n}$ over $\mathbb{F}_2$ is defined as the minimum XOR gates of all general implementations of A.

3 State-of-art heuristics

In this section, we present several existing methods to address the SLPD problem, including the BP algorithm with depth limit [LSL+19], the backward search [LWF+22], and the local re-optimization method [LZW23].

3.1 BP algorithm with depth limit

The original BP algorithm is a heuristic introduced by Boyar and Peralta to address the SLP problem [BMP13]. It involves the utilization of two key parameters: the base set $\mathcal{B}$ and the distance vector $Dist_{\mathcal{B}}$. The base set $\mathcal{B}$ consists of nodes that have already been computed, while the distance vector $Dist_{\mathcal{B}}$ describes the distances from the base set $\mathcal{B}$ to all output nodes. Formally, $Dist_{\mathcal{B}}[i]$ is defined as $Dist_{\mathcal{B}}[i] = \min\{d \mid y_i = \bigoplus_{t=1}^{d} \mathcal{B}[i_t]\}$ [3]. The subscript $\mathcal{B}$ can be omitted without ambiguity. The smaller $Dist[i]$ is, the closer $\mathcal{B}$ is to y_i.

In each iteration of the BP algorithm, one selects a new base element generated by two nodes in $\mathcal{B}$, adds it to $\mathcal{B}$, and then updates the distance vector $Dist$. BP's original strategy to choose a new base element works as follows:

- Choose the first encountered new base element that minimizes the sum of new $Dist[i]$;
- Resolve ties by maximizing the Euclidean norm of new $Dist$.

Additionally, a pre-emptive strategy is adopted:

- If there exist two base elements $\mathcal{B}[i]$ and $\mathcal{B}[j]$ such that $\mathcal{B}[i] \oplus \mathcal{B}[j]$ equals an output node, then one adds it directly to $\mathcal{B}$ and update $Dist$.

This pre-emptive strategy can often save the running time without increasing the overall cost, as the majority of time is spent computing new $Dist$s.

Note that the BP algorithm produces cancellation-allowed implementations while Paar's method [Paa97] only produces cancellation-free implementations, which is thought to be the main reason why the former performs much better than the latter. Moreover, if multiple new base elements are allowed to be selected in the BP algorithm at once, then the BP algorithm must be able to obtain the optimal solution under the condition of infinite computing resources.

Several refinements have been proposed for the BP algorithm in recent studies. Visconti *et al.* [VSP18] introduced the definition of critical path to address dense matrices. Tan *et al.* proposed the RNBP, A1 and A2 algorithms [TP20], which all are improvements of the original BP algorithm. Compared with BP's original strategy that has ruled out the possibility of subsequent choices with the same parameters, the RNBP algorithm ensures that every tie-breaking choice is equally possible. The A1 algorithm adds a filtering step to the RNBP algorithm: the choice of a new base element must reduce the distance to at least one of the nearest output nodes. Its motivation is similar to that of the tie-breaking

[3]For convenience, we have modified the original definition so that $Dist_{\mathcal{B}}[i]$ is exactly the minimum number of nodes in $\mathcal{B}$ required to generate y_i via the XOR gate. The original definition represents the number of nodes minus one, which is the required XOR gate count.

rule in the original BP algorithm. The A2 algorithm works the same as the A1 algorithm, except that the Euclidean norm of $Dist$ no longer acts as the tie-breaker.

Another improvement is proposed by Banik *et al.* [BFI21]. It involves randomly choosing two permutation matrices $P_{m\times m}, Q_{n\times n}$ over $\mathbb{F}_2$ and then applying the original BP algorithm on the matrix PAQ. This technique introduces randomness to the choice of a new base element by altering the order of input and output nodes and can find better results. However, it is worth noting that the strategy presented in the RNBP algorithm has already guaranteed an equal probability of all choices. Therefore, altering the order of input and output nodes becomes meaningless.

When it comes to low-latency implementations, the original BP algorithm as well as its several improvements no longer works, since they cannot bound the depth of chosen new base elements. To address this limitation, Li *et al.* proposed a revised BP algorithm with depth limit [LSL+19]. The main difference from the original BP algorithm is that the depth of each base element is recorded and cannot exceed the specified depth limit. Meanwhile, the distance vector $Dist$ is computed under the depth limit. Since Li *et al.*'s algorithm follows the original BP algorithm, it cannot guarantee equal probability of all possible choices, while Banik *et al.*'s strategy indeed introduces randomness to it and can yield better results [BFI21].

For simplicity, we collectively call the BP algorithm and all its variants as forward search throughout the rest of the paper.

3.2 Backward search

Liu *et al.* put forward a new greedy algorithm called backward search to address the SLPD problem [LWF+22]. Instead of starting from input nodes, the backward search initiates the search from output nodes and ensures that all nodes achieve their minimum depths [4]. Therefore it is suitable for the low-latency criteria. Specifically, they defined a working set $\mathcal{W}$ containing the nodes to be split and a predecessor set $\mathcal{P}$. The nodes p in $\mathcal{P}$ can be repeatedly utilized as predecessors to split nodes w in $\mathcal{W}$, i.e. $w = p \oplus p'$. Additionally, a parameter s, known as the current depth, represents the depth of the elements in $\mathcal{W}$. Consequently, a predecessor p of w must satisfy $d_{min}(p) < s$. The algorithm works as follows: at the beginning, $s = d_{min}(A)$, $\mathcal{P} = \varnothing$ and $\mathcal{W}$ contains all y_i's. In each iteration, randomly execute one of the choices of the rule with the highest priority. If $\mathcal{W}$ is empty, let $\mathcal{W} = \mathcal{P}, \mathcal{P} = \varnothing, s = s - 1$. The algorithm continues until $s = 0$.

For a given node w, a key operation in the backward search is to determine its predecessors according to some predefined rules. Therefore, designing splitting rules with different priorities to guide the search becomes the main task. In order to maximize the reuse of predecessor nodes, the authors developed five priority-based rules for splitting nodes within $\mathcal{W}$:

- **Rule 1** If $\exists w \in \mathcal{W}$, s.t. $d_{min}(w) < s$, move w from $\mathcal{W}$ to $\mathcal{P}$.
- **Rule 2** If $\exists p_1, p_2 \in \mathcal{P}$ and $w \in \mathcal{W}$, s.t. $w = p_1 \oplus p_2$, remove w from $\mathcal{W}$.
- **Rule 3** If $\exists p_1 \in \mathcal{P}$, $w \in \mathcal{W}$, s.t. $p_2 = w \oplus p_1$, and $d_{min}(p_2) < s$, remove w from $\mathcal{W}$ and add p_2 to $\mathcal{P}$.
- **Rule 4** If $\exists w_1, w_2 \in \mathcal{W}$, s.t. $w_1 = p_1 \oplus p_2$, $w_2 = p_2 \oplus p_3$, and $d_{min}(p_1) < s$, $d_{min}(p_2) < s$, $d_{min}(p_3) < s$, remove w_1, w_2 from $\mathcal{W}$ and add p_1, p_2, p_3 to $\mathcal{P}$.
- **Rule 5** Randomly choose $w \in \mathcal{W}$ and randomly split $w = p_1 \oplus p_2$, s.t. $d_{min}(p_1) < s, d_{min}(p_2) < s$, remove w from $\mathcal{W}$ and add p_1, p_2 to $\mathcal{P}$.

[4]This is achieved by **Rule 1**.

They further introduced the cost of a splitting rule, which is described as how many nodes are split, how many XOR gates are needed, and how many new predecessors are generated. According to the cost of 5 splitting rules illustrated in Table 2, in [LWF+22] they recommended the following priority:

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

where **Rule 4** is regarded as the combination of **Rule 3** and **Rule 5**, and has the same priority as **Rule 3**. Indeed, we noticed that **Rule 3** allows the cancellation of input terms but **Rule 4** does not, so the combination of **Rule 3** and **Rule 5** is not entirely equivalent to **Rule 4** and can produce even more possibilities than the latter, which has been utilized in our new framework to explore a larger search space.

Table 2: The costs of 5 splitting rules

Rules	Split nodes	Gates	New predecessors
Rule 1	1	0	0
Rule 2	1	1	0
Rule 3	1	1	1
Rule 4	2	2	3
Rule 5	1	1	2

3.3 Local re-optimization method

Liu *et al.* introduced a framework for local re-optimization of a given low-latency implementation [LZW23]. This framework involves enumerating an extended base set and a segment of intermediate nodes in the given low-latency implementation as inputs and outputs respectively, followed by a re-optimization process to determine if fewer XOR gates are needed. In fact, different heuristics can be used during the process of re-optimization. Though the framework seems to offer more possibilities, only a few of matrices in their experiments have improved results.

4 New methods

In this section, we first propose two improved algorithms involving refining the strategies employed in the BP algorithm with depth limit. And then we give a new framework of combining forward search with priority-modified backward search to expand the search space. Finally, we make some comparisons with known algorithms.

4.1 IBPD & IBPD-MD algorithms

Similar to the BP algorithm with depth limit, our new algorithms remain the strategy of minimizing the sum of new $Dist[i]$ unchanged when a new base element is chosen, and take consideration of depth limit in computing the $Dist$ parameter. The base set $\mathcal{B}$ is associated with the depth set $\mathcal{D}$, and for each index i, $\mathcal{B}[i]$ has depth $\mathcal{D}[i]$. Below we first provide a lemma on the depth of nodes:

Lemma 1. *[LSL+19] Let $\{v_1, v_2, \ldots, v_n\}$ be a set of nodes with $d(v_i) = d_i$. Then the lower bound of the depth of the circuit implementing $z = v_1 \oplus v_2 \oplus \cdots \oplus v_n$ is $\lceil \log_2(\sum_{i=1}^{n} 2^{d_i}) \rceil$. Moreover, there is always a circuit implementing z with depth $\lceil \log_2(\sum_{i=1}^{n} 2^{d_i}) \rceil$, i.e. the lower bound is achievable.*

In [LSL+19], a description of how to compute $Dist[i]$ with a depth limit were not explicitly provided, but the computing process can be inferred from their public C++

code. Specifically, the parameter L in their algorithm requires that if $t = v_1 \oplus v_2 \oplus \cdots \oplus v_n$ with $d(v_i) = d_i$, then $\sum_{i=1}^{n} 2^{d_i} \leq L$ is necessary. Particularly, according to Lemma 1, if $y_i = v_1 \oplus v_2 \oplus \cdots \oplus v_n$ with depth limit b_i, then $\sum_{i=1}^{n} 2^{d_i} \leq 2^{b_i}$, where $d(v_i) = d_i$. Algorithm 1, named ChooseComb, determines if a node t can be expressed by K base elements with indices larger or equal to S under the restriction of $\mathcal{D}$ and the parameter L, and returns **True** if yes, otherwise, **False**. It is executed during the computation of a new distance vector $Dist$ in order to check if some $Dist[i] > 2$ is reduced by one.

Algorithm 1 ChooseComb$(t, K, S, L, \mathcal{B}, \mathcal{D})$

```
Input: $t, K, S, L, \mathcal{B}, \mathcal{D}$
Output: True or False
  if $L < 1$ or $K = 0$ or $| \mathcal{B} | - S < K$ then
      return False
  end if
  if $K = 1$ then
      if $t \in \mathcal{B}$ then
          return True
      else
          return False
      end if
  end if
  if ChooseComb$(t, K, S + 1, L, \mathcal{B}, \mathcal{D})$ then
      return True
  end if
  if ChooseComb$(t \oplus \mathcal{B}[S], K - 1, S + 1, L - 2^{\mathcal{D}[S]}, \mathcal{B}, \mathcal{D})$ then
      return True
  end if
  return False
```

Time complexity of Algorithm 1: The time complexity of our heuristics is dominated by computing new $Dist$s under each choice of a new base element. Denote the time complexity of executing Algorithm 1 as $C(\mathcal{B}, L, K, S)$. It is easy to see that by induction and some boundary conditions,

$$C(\mathcal{B}, L, K, S) \leq C(\mathcal{B}, L - 2^{\mathcal{D}[S]}, K - 1, S + 1) + C(\mathcal{B}, L, K, S + 1).$$

The time complexity of determining whether a target node t can be computed with $(K-1)$ XOR gates is $C(\mathcal{B}, L, K, 0)$. If $\mathcal{D}[S] = 0$ for all S, a very loose estimation of $C(\mathcal{B}, L, K, 0)$ is $C(\mathcal{B}, L, K, 0) \leq \binom{|\mathcal{B}|}{\min\{L,K\}}$. For most 32×32 matrices in our experiments, $|\mathcal{B}|$ is upper-bounded by 140 and K is upper-bounded by 8. So a very loose time upper bound of computing a new $Dist[i]$ in our experiments is $\binom{140}{8} \approx 2^{41}$. In fact, things are not that bad. Since $\mathcal{D}[S]$ is 0 only for input nodes and greater than 0 for all intermediate nodes, L decreases very fast in practice. Also, as $|\mathcal{B}|$ gets larger, K gets smaller. In addition, the backtracking process has pruning operations. Thus the actual time to compute a new $Dist[i]$ is far less than the very loose time upper bound.

The primary difference between our new algorithm IBPD and Li *et al.*'s algorithm can be identified in two key aspects. Firstly, just like in RNBP, we make tie-breaking choices of new base elements with equal probability. Secondly, in order to approach all output nodes at a relatively consistent pace when the sum of new $Dist[i]$ is equal to before, our tie-breaking rule aims to *minimize* the Euclidean norm of new $Dist$ instead of maximizing it, which significantly increases the potential for searching better results.

Furthermore, similar to the idea of Additional Output Requirement in [ME19], we generalize the algorithm by assuming that each y_i has its own depth limit b_i, where $b_i \geq d_{min}(y_i)$. To obtain minimum latency implementations of a given matrix A, the condition $\max\{b_i\} = d_{min}(A)$ is necessary. Procedures of Algorithm 2, named Improved BP with Depth limit (IBPD in short), is illustrated below. The implementation of A can be inferred from the returned $\mathcal{B}$. Note that different output nodes can have different depth bounds, IBPD is easily combined with backward search.

Algorithm 2 Improved BP with Depth limit (IBPD)

```
Input: A matrix $A_{m\times n}$ over $\mathbb{F}_2$, i.e. $y_i = (a_{i,0}, a_{i,1}, \dots, a_{i,n-1}), 0 \leq i < m$, depth bound
    $b = (b_0, b_1, \dots, b_{m-1})$, $\max\{b_i\} = d_{min}(A)$.
Output: $\mathcal{B}$, such that $\mathcal{B}[i] = \mathcal{B}[j] \oplus \mathcal{B}[k], n \leq i <| \mathcal{B} |, j, k < i$, and for every $y_i$, there
    exists $j$ such that $\mathcal{B}[j] = y_i, \mathcal{D}[j] \leq b_i$.
 1: if $\exists i$, s.t. $b_i < d_{min}(y_i)$ then
 2:     return $\varnothing$.                              ▷ impossible with this given $b$
 3: end if
 4: $\mathcal{B} \leftarrow \{x_0, x_1, \dots, x_{n-1}\}$.
 5: $\mathcal{D} \leftarrow \{0, 0, \dots, 0\}$.
 6: $Dist \leftarrow \{wt(y_0), wt(y_1), \dots, wt(y_{m-1})\}$.
 7: while $\exists i$, s.t. $Dist[i] > 1$ do
 8:     if $\exists i$, s.t. $Dist[i] = 2$ then
 9:         Find $j, k$ s.t. $\mathcal{B}[j] \oplus \mathcal{B}[k] = y_i$ and $\mathcal{D}[j], \mathcal{D}[k] < b_i$, then $\mathcal{B} \leftarrow \mathcal{B} \cup \{y_i\}, \mathcal{D} \leftarrow$
    $\mathcal{D} \cup \{\max\{\mathcal{D}[j], \mathcal{D}[k]\} + 1\}$.
10:     else
11:         Randomly choose a new base element $\beta = \mathcal{B}[j] \oplus \mathcal{B}[k]$, with the following rules:
    (1) The depth of $\beta$ cannot go beyond the maximum depth limit $d_{min}(A)$; (2) Minimize
    the sum of new $Dist[i]$ with the tie breaking rule of *minimizing* the Euclidean norm of
    new $Dist$.
12:         $\mathcal{B} \leftarrow \mathcal{B} \cup \{\beta\}, \mathcal{D} \leftarrow \mathcal{D} \cup \{\max\{\mathcal{D}[j], \mathcal{D}[k]\} + 1\}$.
13:     end if
14: end while
15: return $\mathcal{B}$.
```

Just as A1 is an adjustment of RNBP [TP20], our new algorithm IBPD-MD is an adjustment of IBPD. A key difference lies in the additional filtering step: the choice of a new base element must reduce the distance to at least one of the nearest output nodes. It does not violate the principal of approaching all output nodes at a relatively consistent pace and is beneficial to find better results in many cases. IBPD-MD is illustrated below as Algorithm 3.

In practice, both algorithms can find better results in many examples where all output nodes have the same minimum depth, and IBPD-MD is better suited for cases where the minimum depth of some output nodes is smaller than others because the depth limit is a bit looser.

4.2 New framework

In this subsection we will provide a new framework of combining forward search with backward search. In our new framework, IBPD or IBPD-MD is invoked in the forward search, and a modified priority of rules is used in the backward search. In practice, the IBPD-MD version performs better than the IBPD version, and they both can find better results than single IBPD or IBPD-MD in many cases. Figure 1 illustrates the IBPD-MD version.

Algorithm 3 Improved BP with Depth limit and reduction of Minimum Distance (IBPD-MD)

```
Input: A matrix $A_{m\times n}$ over $\mathbb{F}_2$, i.e. $y_i = (a_{i,0}, a_{i,1}, \ldots, a_{i,n-1}), 0 \le i < m$, depth bound $b = (b_0, b_1, \ldots, b_{m-1})$, $\max\{b_i\} = d_{min}(A)$.
Output: $\mathcal{B}$, such that $\mathcal{B}[i] = \mathcal{B}[j] \oplus \mathcal{B}[k], n \le i <| \mathcal{B} |, j, k < i$, and for every $y_i$, there exists $j$ such that $\mathcal{B}[j] = y_i, \mathcal{D}[j] \le b_i$.
1: if $\exists i$, s.t. $b_i < d_{min}(y_i)$ then
2:     return $\varnothing$.                    ▷ impossible with this given $b$
3: end if
4: $\mathcal{B} \leftarrow \{x_0, x_1, \ldots, x_{n-1}\}$.
5: $\mathcal{D} \leftarrow \{0, 0, \ldots, 0\}$.
6: $Dist \leftarrow \{wt(y_0), wt(y_1), \ldots, wt(y_{m-1})\}$.
7: while $\exists i$, s.t. $Dist[i] > 1$ do
8:     if $\exists i$, s.t. $Dist[i] = 2$ then
9:         Find $j, k$ s.t. $\mathcal{B}[j] \oplus \mathcal{B}[k] = y_i$ and $\mathcal{D}[j], \mathcal{D}[k] < b_i$, then $\mathcal{B} \leftarrow \mathcal{B} \cup \{y_i\}, \mathcal{D} \leftarrow \mathcal{D} \cup \{\max\{\mathcal{D}[j], \mathcal{D}[k]\} + 1\}$.
10:    else
11:        Randomly choose a new base element $\beta = \mathcal{B}[j] \oplus \mathcal{B}[k]$, with the following rules: (1) The depth of $\beta$ cannot go beyond the maximum depth limit $d_{min}(A)$; (2) The choice $\beta$ must reduce the distance to at least one of the nearest output nodes; (3) Minimize the sum of new $Dist[i]$ with the tie breaking rule of minimizing the Euclidean norm of new $Dist$.
12:        $\mathcal{B} \leftarrow \mathcal{B} \cup \{\beta\}, \mathcal{D} \leftarrow \mathcal{D} \cup \{\max\{\mathcal{D}[j], \mathcal{D}[k]\} + 1\}$.
13:    end if
14: end while
15: return $\mathcal{B}$.
```

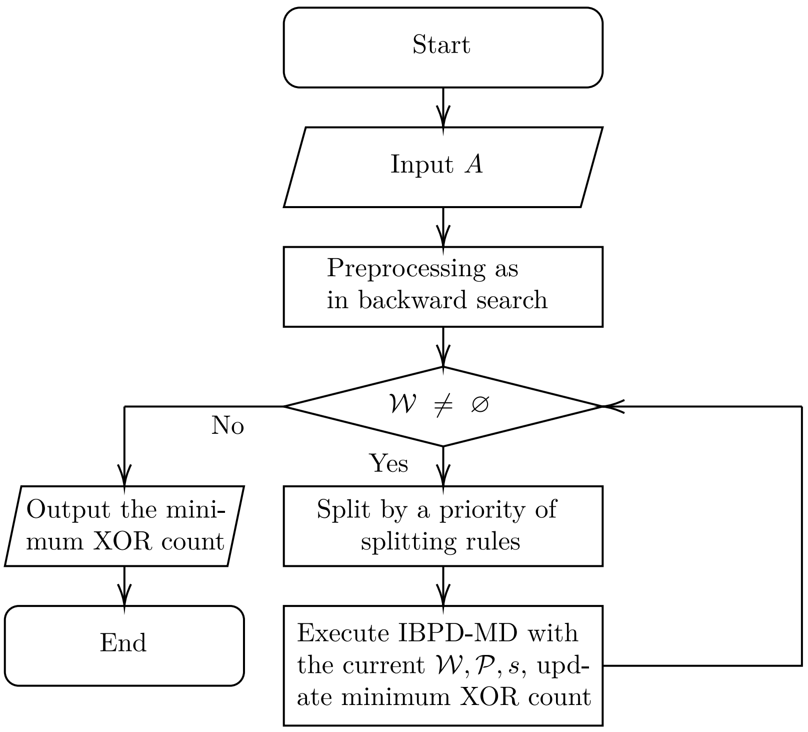
Start
Input A
Preprocessing as in backward search
$\mathcal{W} \neq \varnothing$
No
Yes
Output the minimum XOR count
Split by a priority of splitting rules
End
Execute IBPD-MD with the current $\mathcal{W}, \mathcal{P}, s$, update minimum XOR count

Figure 1: Framework of integrating IBPD-MD with backward search.

In the above IBPD-MD version, the nodes in $\mathcal{W}, \mathcal{P}$ are actually viewed as output nodes in IBPD-MD in every iteration of the backward search. Among them, the nodes in $\mathcal{W}$ have depth bound s, and the predecessor and non-predecessor nodes in $\mathcal{P}$ have depth bound $s-1$ and s, respectively. Since we only split the nodes in the initial $\mathcal{W}$, IBPD-MD will dominate the search and the backward search only adjusts the search space by predetermining how to generate some y_i's. The adjustments of the search space is helpful for IBPD-MD to jump out of local minima.

As for the splitting rules of the backward search in our framework, their priority can be adjusted to explore how to generate output nodes in different ways. A few examples of priority are listed as follows. Among them, the priority (1) is introduced in [LWF+22].

- (1) **Rule 1** > **Rule 2** > **Rule 3**, **Rule 4** > **Rule 5**;
- (2) **Rule 1** > **Rule 2** > **Rule 4**, **Rule 5**;
- (3) **Rule 1** > **Rule 2** > **Rule 3**, **Rule 5**;
- (4) **Rule 1** > **Rule 2** > **Rule 3**, **Rule 4**, **Rule 5**;
- (5) **Rule 1** > **Rule 2** > **Rule 5**.

Interestingly, the priority (3) yields significantly better results than the others according to our experiments. We think there are three main reasons for this. Firstly, IBPD-MD itself can find better results than the original backward search in many cases. Secondly, compared to **Rule 3** and **Rule 4**, higher priority of **Rule 5** can result in a larger search space. Finally, **Rule 4** does not allow cancellation of input terms, and all its cases can be included by the combination of **Rule 3** and **Rule 5**. Therefore we use the priority (3) in our framework. Here it should be pointed out that alternative priorities may also yield satisfactory results in specific cases.

In conclusion, combining IBPD-MD with priority-modified backward search can enlarge the search space of IBPD-MD by constraining how the original output nodes are generated. Below we provide a precise formulation of the framework through Algorithm 4 named BPBS. The implementation can be inferred from $\mathcal{B}$ and S.

Algorithm 4 Improved BP combined with Backward Search(BPBS)

```
Input: A matrix $A_{m\times n}$ over $\mathbb{F}_2$, i.e. $y_i = (a_{i,0}, a_{i,1}, \ldots, a_{i,n-1}), 0 \le i < m$.
Output: Minimum XOR gate count of all searched implementations.
1: $\mathcal{W} \leftarrow \{y_i\}, 0 \le i < m$
2: $\mathcal{P} \leftarrow \varnothing$.
3: $s \leftarrow \max(d_{min}(y_i))$.
4: $S \leftarrow \varnothing$
5: $minm \leftarrow \infty$
6: while $\mathcal{W} \neq \varnothing$ do
7:     Split by a random choice of the rule with the highest priority under the priority (3). Add the split node to $S$.
8:     The algorithm IBPD-MD is executed multiple times, where the output nodes consist of two parts: (1) the nodes in $\mathcal{W}$, whose depth bounds are set to $s$; (2) the predecessor and non-predecessor nodes in $\mathcal{P}$, whose depth bounds are set to $s-1$ and $s$ respectively. Record one returned $\mathcal{B}$ with minimum number of elements.
9:     $minm = \min(minm, |\mathcal{B}| + |S| - n)$
10: end while
11: return $minm$.
```

4.3 Comparisons and discussions

In this subsection we compare our methods with the existing ones and explain the advantages of our methods.

Advantage of the new tie-breaking rule. Our new algorithms IBPD and IBPD-MD both outperform Li *et al.*'s algorithm [LSL+19] and its improvement in [BFI21], which we believe is mainly due to the improved strategy: when the selection of a new base element results in a sum of new $Dist[i]$ equal to before, our tie-breaking resolution will minimize the Euclidean norm of new $Dist$. In the original BP algorithm [BMP08], the authors put forward a novel tie-breaking resolution of maximizing the Euclidean norm of new $Dist$. They believed that a distance vector like $(1, 1, 4, 2)$ is superior to one like $(2, 2, 2, 2)$ because the latter needs 4 XOR gates while 3 XOR gates may be possible for the former. This strategy works well for the SLP problem and is adopted in [LSL+19, BFI21]. However, things are different in the case of the SLPD problem. Due to the depth limit, output nodes and those with high depth have a smaller contribution to other output nodes. Take the distance vector $(1, 1, 4, 2)$ as an example. y_3 cannot contribute to y_2 if it reaches the depth bound. Thus the original tie-breaking strategy may not work well in the new scenario, since prioritizing closer outputs has a smaller impact on approaching other outputs and may result in the loss of alternative, better pathways that generate the closer outputs. Instead, we tend to reduce all $Dist[i]$'s at a relatively consistent pace, which is achieved by the tie-breaking rule of minimizing the Euclidean norm of new $Dist$ and is proved to be effective in the SLPD problem by our experiments.

Difference between IBPD and IBPD-MD. Since both IBPD and IBPD-MD work well in different examples according to our experiments, they have their own suitable scenarios. For matrices with the same minimum depth of all output nodes, some of the best results can only be searched by one of the two algorithms, which might be due to the different characteristics of different matrices. However, for matrices with different minimum depths of output nodes, IBPD-MD performs better than IBPD due to the relatively loose depth limit. In fact, output nodes with a smaller minimum depth may contribute to others, and prioritizing them may be beneficial for finding low-latency implementations that save more gates. This is exactly the case in our new framework, so the IBPD-MD version performs better than the IBPD version.

Improvements in the new framework. In order to further improve the search results of IBPD and IBPD-MD, we formulate our framework that combines forward search with backward search. A significant advantage of the backward search is the ability to explore more possibilities for generating output nodes, which is not easy with forward search alone. Conversely, the backward search cannot predetermine the initial steps of implementations. Therefore, their combination seems likely to yield better results. Liu *et al.* introduced a method for local re-optimization of a given implementation [LZW23]. Compared to their method, our framework does not depend on any existing implementations and has a much larger expanded search space.

Each node does not have to be at a minimum depth in our framework, which provides more possibilities than the original backward search. In each iteration of our framework, the nodes in $\mathcal{P}$ which are not predecessors of any other nodes can have a relaxed depth bound in IBPD-MD. Note that in IBPD-MD, different output nodes are allowed to have different depth bounds. As mentioned earlier, the original backward search ensures that all nodes achieve their minimum depths. If some $d_{min}(y_i)$ is smaller than $d_{min}(A)$, the original backward search would eliminate the possibility of $d(y_i) > d_{min}(y_i)$, which may result in the omission of better implementations. Actually, the only matrix with minimum depth 4 in Table 5 proposed in [SS16] and many examples of MDS matrices proposed in [LSL+19] demonstrate this drawback of the original backward search. An important property of these matrices is that the minimum depth of only a few output nodes is equal to that of the matrix.

The priority (3) in our framework effectively explores a wide range of ways to generate outputs. Since IBPD-MD dominates the search of implementations in our framework and we aim to explore alternative ways to generate output nodes, the combination of **Rule 3** and **Rule 5** works well by taking randomness into account while allowing for the cancellation of input terms. So the priority (3) performs better than (1) and (2) in our framework. Actually, all the best results can be found by (3) in our experiments, while it does not mean that other priorities cannot produce some of the same results. It is important to note that alternative choices of priority rules may also yield satisfactory results, such as higher priority of **Rule 4** may be more efficient in some cases. Moreover, some priorities involving much larger randomness such as priority (4) and (5) also work well under the assumption that the computation resource is sufficient.

5 Results and applications

In this section we apply our algorithms IBPD, IBPD-MD and framework BPBS to various 16×16 and 32×32 matrices studied in related works. The priority (3) is used in our framework. All methods are repeated tens of thousands of times for each matrix and the best results are recorded. With 360 threads running in parallel, it takes less than a minute and less than two hours to find the best result for each 16×16 and 32×32 matrix, respectively. For the most of matrices studied in previous works, improvements of their minimum latency implementations can be found, which are listed in Table 5.

5.1 Application to AES MixColumns

We first apply our methods to AES MixColumns, whose minimum depth is 3. The paper [LSL+19] reported an implementation with 105 XOR gates, and [BFI21] found an implementation with 103 XOR gates. In [LWF+22], the backward search can also find an implementation with 103 XOR gates, which is optimized to 102 XOR gates in [LZW23] by means of the local re-optimization method.

As for our methods, IBPD discovers a minimum latency implementation with 101 XOR gates, while IBPD-MD can find one with 100 XOR gates. Remarkably, the framework BPBS is capable of finding an implementation of depth 3 with only 99 XOR gates. The implementation is presented in Table 4. A comparison with previous findings is outlined in Table 3.

Table 3: XOR/depth costs of AES MixColumns

Source	[KLSW17]	[TP20]	[XZL+20]	[Max19]
XORs/depth	97/8	94/6	92/6	92/6
Source	[LXZZ21]	[LSL+19]	[BFI21]	[LWF+22]
XORs/depth	91/7	105/3	103/3	103/3
Source	[LZW23]	**IBPD**	**IBPD-MD**	**BPBS**
XORs/depth	102/3	**101/3**	**100/3**	**99/3**

In a personal computer, applying IBPD-MD to AES MixColumns for one time takes no more than ten minutes.

Table 4: An implementation of AES MixColoumns of depth 3 with 99 XOR gates

No.	Operation	Depth
1	$x_{32} = x_2 \oplus x_{10}$	1
2	$x_{33} = x_1 \oplus x_{25}$	1
3	$x_{34} = x_{18} \oplus x_{33}$	2
4	$x_{35} = x_9 \oplus x_{25}$	1
5	$x_{36} = x_{10} \oplus x_{26}$	1
6	$x_{37} = x_{35} \oplus x_{36}$	2
7	$x_{38} = x_9 \oplus x_{17}$	1
8	$x_{39} = x_2 \oplus x_{38}$	2
9	$x_{40} = x_{37} \oplus x_{39} // y_{18}$	3
10	$x_{41} = x_{18} \oplus x_{26}$	1
11	$x_{42} = x_{39} \oplus x_{41} // y_{10}$	3
12	$x_{43} = x_7 \oplus x_{31}$	1
13	$x_{44} = x_8 \oplus x_{16}$	1
14	$x_{45} = x_0 \oplus x_{43}$	2
15	$x_{46} = x_{44} \oplus x_{45} // y_{24}$	3
16	$x_{47} = x_7 \oplus x_{15}$	1
17	$x_{48} = x_{24} \oplus x_{44}$	2
18	$x_{49} = x_{47} \oplus x_{48} // y_0$	3
19	$x_{50} = x_{14} \oplus x_{22}$	1
20	$x_{51} = x_{23} \oplus x_{43}$	2
21	$x_{52} = x_{50} \oplus x_{51} // y_{15}$	3
22	$x_{53} = x_6 \oplus x_{30}$	1
23	$x_{54} = x_{47} \oplus x_{53}$	2
24	$x_{55} = x_{23} \oplus x_{54} // y_{31}$	3
25	$x_{56} = x_{21} \oplus x_{29}$	1
26	$x_{57} = x_{14} \oplus x_{56}$	2
27	$x_{58} = x_{53} \oplus x_{57} // y_{22}$	3
28	$x_{59} = x_4 \oplus x_{12}$	1
29	$x_{60} = x_{13} \oplus x_{59}$	2
30	$x_{61} = x_{56} \oplus x_{60} // y_5$	3
31	$x_{62} = x_5 \oplus x_{13}$	1
32	$x_{63} = x_{30} \oplus x_{62}$	2
33	$x_{64} = x_{50} \oplus x_{63} // y_6$	3
34	$x_{65} = x_{20} \oplus x_{28}$	1
35	$x_{66} = x_{29} \oplus x_{65}$	2
36	$x_{67} = x_{62} \oplus x_{66} // y_{21}$	3
37	$x_{68} = x_1 \oplus x_{24}$	1
38	$x_{69} = x_{38} \oplus x_{68}$	2
39	$x_{70} = x_{45} \oplus x_{69} // y_{25}$	3
40	$x_{71} = x_6 \oplus x_{22}$	1
41	$x_{72} = x_{31} \oplus x_{71}$	2
42	$x_{73} = x_{54} \oplus x_{72} // y_{23}$	3
43	$x_{74} = x_{15} \oplus x_{23}$	1
44	$x_{75} = x_{50} \oplus x_{74}$	2
45	$x_{76} = x_{72} \oplus x_{75} // y_7$	3
46	$x_{77} = x_0 \oplus x_8$	1
47	$x_{78} = x_{74} \oplus x_{77}$	2
48	$x_{79} = x_{48} \oplus x_{78} // y_8$	3
49	$x_{80} = x_{23} \oplus x_{31}$	1
50	$x_{81} = x_{24} \oplus x_{80}$	2
51	$x_{82} = x_{77} \oplus x_{81} // y_{16}$	3
52	$x_{83} = x_5 \oplus x_{21}$	1
53	$x_{84} = x_{71} \oplus x_{83}$	2
54	$x_{85} = x_{57} \oplus x_{84} // y_{30}$	3
55	$x_{86} = x_{63} \oplus x_{84} // y_{14}$	3
56	$x_{87} = x_{12} \oplus x_{28}$	1
57	$x_{88} = x_{83} \oplus x_{87}$	2
58	$x_{89} = x_{60} \oplus x_{88} // y_{29}$	3
59	$x_{90} = x_{66} \oplus x_{88} // y_{13}$	3
60	$x_{91} = x_{68} \oplus x_{80}$	2
61	$x_{92} = x_{16} \oplus x_{35}$	2
62	$x_{93} = x_{91} \oplus x_{92} // y_{17}$	3
63	$x_{94} = x_{17} \oplus x_{35}$	2
64	$x_{95} = x_{47} \oplus x_{77}$	2
65	$x_{96} = x_{94} \oplus x_{95} // y_1$	3
66	$x_{97} = x_{17} \oplus x_{44}$	2
67	$x_{98} = x_{33} \oplus x_{74}$	2
68	$x_{99} = x_{97} \oplus x_{98} // y_9$	3
69	$x_{100} = x_{19} \oplus x_{27}$	1
70	$x_{101} = x_{11} \oplus x_{47}$	2
71	$x_{102} = x_{32} \oplus x_{100}$	2
72	$x_{103} = x_{101} \oplus x_{102} // y_3$	3
73	$x_{104} = x_3 \oplus x_{20}$	1
74	$x_{105} = x_{87} \oplus x_{104}$	2
75	$x_{106} = x_{101} \oplus x_{105} // y_4$	3
76	$x_{107} = x_{80} \oplus x_{100}$	2
77	$x_{108} = x_4 \oplus x_{87}$	2
78	$x_{109} = x_{107} \oplus x_{108} // y_{20}$	3
79	$x_{110} = x_{27} \oplus x_{104}$	2
80	$x_{111} = x_{43} \oplus x_{59}$	2
81	$x_{112} = x_{110} \oplus x_{111} // y_{28}$	3
82	$x_{113} = x_{11} \oplus x_{19}$	1
83	$x_{114} = x_{65} \oplus x_{113}$	2
84	$x_{115} = x_4 \oplus x_{74}$	2
85	$x_{116} = x_{114} \oplus x_{115} // y_{12}$	3
86	$x_{117} = x_2 \oplus x_{26}$	1
87	$x_{118} = x_3 \oplus x_{43}$	2
88	$x_{119} = x_{113} \oplus x_{117}$	2
89	$x_{120} = x_{118} \oplus x_{119} // y_{27}$	3
90	$x_{121} = x_{10} \oplus x_{74}$	2
91	$x_{122} = x_3 \oplus x_{18}$	1
92	$x_{123} = x_{100} \oplus x_{122}$	2
93	$x_{124} = x_{121} \oplus x_{123} // y_{11}$	3
94	$x_{125} = x_{41} \oplus x_{80}$	2
95	$x_{126} = x_{11} \oplus x_{27}$	1
96	$x_{127} = x_3 \oplus x_{126}$	2
97	$x_{128} = x_{125} \oplus x_{127} // y_{19}$	3
98	$x_{129} = x_{34} \oplus x_{37} // y_2$	3
99	$x_{130} = x_{32} \oplus x_{34} // y_{26}$	3

5.2 Application to many proposed matrices

Following the work of [LWF+22, LZW23], we apply our methods to various matrices in the literature including:

- some MDS matrices which are independently constructed in [SKOP15, SS16, JPST17, LS16, LW16, BKL16, KLSW17];
- some matrices which are used in block ciphers [DR02, CMR05, JNP15, Ava17, BBI+15, BCG+12, ADK+14, BJK+16, AIK+01].

For the 11 matrices used in block ciphers, we have identified 8 matrices whose minimum latency implementations have the same number of XOR gates as previous works, while successfully optimizing the remaining 3 matrices. Here it should be noted that 7 out of the 8 matrices have minimum depth 2, and it is most likely that they have no room for optimization due to their relatively simple form. For the constructed MDS matrices, we optimize 19 of the 21 matrices and get equally good results on the rest 2 of them. Note that more than half of the best results can be searched by IBPD or IBPD-MD, and some results can only be searched by the framework BPBS. All the better results and their comparison with previous works are listed in Table 5.

Table 5: The XOR/depth costs for minimum latency implementations of matrices

Matrix	Size	[LSL+19]	[BFI21]	[LWF+22]	[LZW23]	**This paper**
[DR02]AES	32	105/3	103/3	103/3	102/3	**99/3**[a]
[CMR05]SMALLSCALE AES	16	49/3	49/3	47/3	47/3	**46/3**[a]
[JNP15]Joltik	16	51/3	50/3	48/3	48/3	**47/3**[a]
[SKOP15](Hadamard)	16	51/3	50/3	49/3	48/3	**47/3**[a]
[LS16](Circulant)	16	47/3	44/3	44/3	44/3	**43/3**[a]
[LW16](Circulant)	16	47/3	44/3	44/3	44/3	**43/3**[c]
[SS16](Toeplitz)	16	44/3	43/3	45/3	43/3	**42/3**[c]
[JPST17]	16	45/3	45/3	45/3	44/3	**43/3**[a]
[SKOP15](Involutory)	16	51/3	49/3	48/3	48/3	**47/3**[a]
[LW16](Involutory)	16	51/3	49/3	48/3	48/3	**47/3**[a]
[SS16](Involutory)	16	48/3	46/3	45/3	43/3	**42/3**[b]
[JPST17](Involutory)	16	47/3	47/3	47/3	47/3	**46/3**[b]
[SKOP15](Hadamard)	32	102/3	99/3	100/3	99/3	**96/3**[b]
[LS16](Circulant)	32	113/3	113/3	113/3	112/3	**110/3**[c]
[LW16]	32	102/3	103/3	102/3	102/3	**101/3**[c]
[BKL16](Circulant)	32	112/3	110/3	111/3	110/3	**107/3**[b]
[SS16](Toeplitz)	32	107/3	107/3	107/3	107/3	**105/3**[bc]
[JPST17](Subfield)	32	90/3	90/3	93/3	90/3	**89/3**[c]
[SKOP15](Involutory)	32	102/3	100/3	100/3	99/3	**98/3**[b]
[LW16](Involutory)	32	99/3	95/3	94/3	93/3	**89/3**[c]
[SS16](Involutory)	32	104/4	102/4	109/4	102/4	**98/4**[c]
[KLSW17]	32	96/3	-	92/3	89/3	**86/3**[a]
[LSL+19]	32	88/3	-	86/3	85/3	**84/3**[a]

[a] The result can only be searched by BPBS.
[b] The result can be searched by IBPD.
[c] The result can be searched by IBPD-MD.

5.3 Application to matrices from [LSL+19]

We also apply our methods to the 4254 involutory MDS matrices with minimum depth 3 proposed in [LSL+19]. All matrices are of size 32×32, and their Hamming weights range from 148 to 172.

As results, we have an overall improvement about 3752 (88.2%) matrices in terms of their minimum latency implementations (see all the results in Table 6), which is much higher than those in previous works (2310 (54.3%) in [LWF+22] and 3300 (77.6%) in [LZW23]). Compared to the results in [LZW23], our proposed methods yield much better overall performance indices, such as minimum XOR gates (see Figure 2) and reduced XOR gates.

Table 6: Experiments for matrices in [LSL+19]

Hamming weight	Number	Opt.[a]	Percentage[b]	Max.[c]	MinXOR[d]
148	18	18	100.0%	3	87
149	48	48	100.0%	5	86
150	72	72	100.0%	6	86
151	48	48	100.0%	7	88
152	60	60	100.0%	8	89
153	72	72	100.0%	7	87
154	84	84	100.0%	8	87
155	24	24	100.0%	8	91
156	48	48	100.0%	6	93
157	72	72	100.0%	7	90
158	84	84	100.0%	10	88
160	162	146	90.1%	12	84
161	96	96	100.0%	12	86
162	132	132	100.0%	11	85
163	120	96	80.0%	10	86
164	144	132	91.7%	15	86
165	240	240	100.0%	11	85
166	228	228	100.0%	15	85
167	216	168	77.8%	13	84
168	528	493	93.4%	14	84
169	360	322	89.4%	11	85
170	432	388	89.8%	11	84
171	432	362	83.8%	11	85
172	534	319	59.7%	17	86
All	4254	3752	88.2%	17	84

[a] The number of matrices that our algorithms can optimize.
[b] The percentage of matrices that our algorithms can optimize.
[c] The maximum number of reduced XOR gates from our algorithms.
[d] The minimum number of XOR gates.

In [LSL+19], the authors modified the BP algorithm to make it suitable for the low-latency constraint. The algorithm was applied to the matrices they constructed and gave minimum latency implementations for each matrix. The minimum cost is 88 XOR gates. After that, the backward search method [LWF+22] can find an implementation of depth 3 with 86 XOR gates. The result is improved to 85 XOR gates in [LZW23]. Applying our BPBS framework, we find several matrices that have implementations of depth 3 with 84 XOR gates, which represents a new record within the range of 4×4 involutory MDS matrices in $\mathrm{GL}(2, 8)$. One lightest matrix M is the 130th matrix with Hamming weight 168. It has the parameter [4, 4, 10, -4, -6, 0] as referenced in [LSL+19]. Table 7 in the appendix illustrates its implementation.

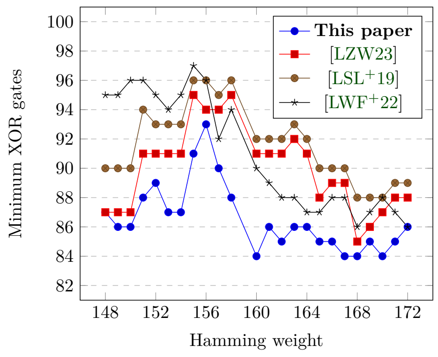

Figure 2: Comparison of the minimum XOR gates with different Hamming weights.

6 Conclusion

In this work we mainly studied the SLPD problem. Two new improved heuristics with a new tie-breaking rule suitable for tight depth bound scenarios are given by improving the BP algorithm with depth bound. Moreover, a new framework of combining an improved forward search with priority-modified backward search is proposed. As applications, we find better results of minimum latency implementations for most of the 16×16 and 32×32 matrices studied in related works. For example, we find a minimum latency implementation of AES MixColumns with 99 XOR gates. As a future work, it would be interesting to study the results underlying the trade-off between area and latency. The design of linear layers in lightweight symmetric primitives can be potentially guided by this work.

References

[ADK+14] Martin R Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçın. Block ciphers–focus on the linear layer (feat. pride). In *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34*, pages 57–76. Springer, 2014.

[AIK+01] 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 andanalysis. In *Selected Areas in Cryptography: 7th Annual International Workshop, SAC 2000 Waterloo, Ontario, Canada, August 14–15, 2000 Proceedings 7*, pages 39–56. Springer, 2001.

[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 Transactions on Symmetric Cryptology*, pages 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 *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 21*, 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, et al. Prince–a low-latency block cipher for pervasive computing applications. In *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 18*, pages 208–225. Springer, 2012.

[BFI21] Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. Further results on efficient implementations of block cipher linear layers. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 104(1):213–225, 2021.

[BGLS19] Zhenzhen Bao, Jian Guo, San Ling, and Yu Sasaki. Peigen–a platform for evaluation, implementation, and generation of s-boxes. *IACR Transactions on Symmetric Cryptology*, pages 330–394, 2019.

[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 *Advances in Cryptology–CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II 36*, pages 123–153. Springer, 2016.

[BKL16] Christof Beierle, Thorsten Kranz, and Gregor Leander. Lightweight multiplication in with applications to mds matrices. In *Annual International Cryptology Conference*, pages 625–653. Springer, 2016.

[BM10] Robert Brayton and Alan Mishchenko. ABC: An academic industrial-strength verification tool. In *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*, pages 24–40. Springer, 2010.

[BMP08] Joan Boyar, Philip Matthews, and René Peralta. On the shortest linear straight-line program for computing linear forms. In *Mathematical Foundations of Computer Science 2008: 33rd International Symposium, MFCS 2008, Toru ́ n, Poland, August 25-29, 2008. Proceedings 33*, 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.

[Can05] David Canright. A very compact s-box for AES. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 441–455. Springer, 2005.

[CMR05] Carlos Cid, Sean Murphy, and Matthew JB Robshaw. Small scale variants of the AES. In *FSE*, volume 3557, pages 145–162. Springer, 2005.

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

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael*, volume 2. Springer, 2002.

[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 Transactions on Symmetric Cryptology*, pages 130–168, 2017.

[KLSW17] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Shorter linear straight-line programs for MDS matrices. *IACR Transactions on Symmetric Cryptology*, pages 188–211, 2017.

[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 Transactions on Cryptographic Hardware and Embedded Systems*, pages 510–545, 2021.

[LS16] Meicheng Liu and Siang Meng Sim. Lightweight MDS generalized circulant matrices. In *Fast Software Encryption: 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, 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 Transactions on Symmetric Cryptology*, pages 84–117, 2019.

[LW16] Yongqiang Li and Mingsheng Wang. On the construction of lightweight circulant involutory MDS matrices. In *Fast Software Encryption: 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 121–139. Springer, 2016.

[LWF$^+$22] Qun Liu, Weijia Wang, Yanhong Fan, Lixuan Wu, Ling Sun, and Meiqin Wang. Towards low-latency implementation of linear layers. *IACR Transactions on Symmetric Cryptology*, pages 158–182, 2022.

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

[LZW23] Qun Liu, Zheng Zhao, and Meiqin Wang. Improved heuristics for low-latency implementations of linear layers. In *Topics in Cryptology–CT-RSA 2023: Cryptographers' Track at the RSA Conference 2023, San Francisco, CA, USA, April 24–27, 2023, Proceedings*, pages 524–550. Springer, 2023.

[Max19] Alexander Maximov. AES Mixcolumn with 92 XOR gates. *Cryptology ePrint Archive*, 2019.

[ME19] Alexander Maximov and Patrik Ekdahl. New circuit minimization techniques for smaller and faster AES sboxes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 91–125, 2019.

[Paa97] Christof Paar. Optimized arithmetic for reed-solomon encoders. In *Proceedings of IEEE international symposium on information theory*, page 250. IEEE, 1997.

[SKOP15] Siang Meng Sim, Khoongming Khoo, Frédérique Oggier, and Thomas Peyrin. Lightweight MDS involution matrices. In *Fast Software Encryption: 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers 22*, pages 471–493. Springer, 2015.

[SS16] Sumanta Sarkar and Habeeb Syed. Lightweight diffusion layer: Importance of toeplitz matrices. *IACR Transactions on Symmetric Cryptology*, pages 95–113, 2016.

[Sto16] Ko Stoffelen. Optimizing s-box implementations for several criteria using sat solvers. In *Fast Software Encryption: 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers 23*, pages 140–160. Springer, 2016.

[TP20] Quan Quan Tan and Thomas Peyrin. Improved heuristics for short linear programs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 203–230, 2020.

[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). *Information processing letters*, 137:1–5, 2018.

[Wol16] Clifford Wolf. Yosys open synthesis suite, 2016.

[XZL$^+$20] Zejun Xiang, Xiangyoung Zeng, Da Lin, Zhenzhen Bao, and Shasha Zhang. Optimizing implementations of linear layers. *IACR Transactions on Symmetric Cryptology*, pages 120–145, 2020.

[YZW21] Yumeng Yang, Xiangyong Zeng, and Shi Wang. Construction of lightweight involutory MDS matrices. *Designs, Codes and Cryptography*, 89(7):1453–1483, 2021.

A The low-latency implementation

Table 7: An implementation of matrix M with 84 XOR gates

No.	Operation	Depth	No.	Operation	Depth
1	$x_{32} = x_{15} \oplus x_{23}$	1	43	$x_{74} = x_{18} \oplus x_{68}$	2
2	$x_{33} = x_{13} \oplus x_{25}$	1	44	$x_{75} = x_{73} \oplus x_{74} // y_6$	3
3	$x_{34} = x_{32} \oplus x_{33}$	2	45	$x_{76} = x_{14} \oplus x_{22}$	1
4	$x_{35} = x_{12} \oplus x_{28}$	1	46	$x_{77} = x_8 \oplus x_{76}$	2
5	$x_{36} = x_4 \oplus x_{22}$	1	47	$x_{78} = x_{73} \oplus x_{77} // y_{22}$	3
6	$x_{37} = x_{35} \oplus x_{36} // y_{28}$	2	48	$x_{79} = x_{12} \oplus x_{24}$	1
7	$x_{38} = x_{11} \oplus x_{29}$	1	49	$x_{80} = x_{76} \oplus x_{79}$	2
8	$x_{39} = x_1 \oplus x_{17}$	1	50	$x_{81} = x_{43} \oplus x_{80} // y_{12}$	3
9	$x_{40} = x_{38} \oplus x_{39} // y_{17}$	2	51	$x_{82} = x_{28} \oplus x_{48}$	2
10	$x_{41} = x_{14} \oplus x_{30}$	1	52	$x_{83} = x_{80} \oplus x_{82} // y_2$	3
11	$x_{42} = x_6 \oplus x_{16}$	1	53	$x_{84} = x_0 \oplus x_8$	1
12	$x_{43} = x_{41} \oplus x_{42} // y_{30}$	2	54	$x_{85} = x_{82} \oplus x_{84} // y_8$	3
13	$x_{44} = x_5 \oplus x_{31}$	1	55	$x_{86} = x_{20} \oplus x_{30}$	1
14	$x_{45} = x_{21} \oplus x_{23}$	1	56	$x_{87} = x_{36} \oplus x_{86}$	2
15	$x_{46} = x_{44} \oplus x_{45}$	2	57	$x_{88} = x_{80} \oplus x_{87} // y_{20}$	3
16	$x_{47} = x_{38} \oplus x_{46} // y_{11}$	3	58	$x_{89} = x_{68} \oplus x_{87} // y_{10}$	3
17	$x_{48} = x_2 \oplus x_{20}$	1	59	$x_{90} = x_0 \oplus x_{35}$	2
18	$x_{49} = x_{10} \oplus x_{26}$	1	60	$x_{91} = x_{86} \oplus x_{90} // y_0$	3
19	$x_{50} = x_{48} \oplus x_{49} // y_{26}$	2	61	$x_{92} = x_{16} \oplus x_{26}$	1
20	$x_{51} = x_{21} \oplus x_{29}$	1	62	$x_{93} = x_{14} \oplus x_{92}$	2
21	$x_{52} = x_3 \oplus x_{51}$	2	63	$x_{94} = x_{71} \oplus x_{93} // y_{14}$	3
22	$x_{53} = x_{34} \oplus x_{52} // y_3$	3	64	$x_{95} = x_{19} \oplus x_{56}$	2
23	$x_{54} = x_{27} \oplus x_{38}$	2	65	$x_{96} = x_{17} \oplus x_{25}$	1
24	$x_{55} = x_{52} \oplus x_{54} // y_{27}$	3	66	$x_{97} = x_{95} \oplus x_{96} // y_{25}$	3
25	$x_{56} = x_1 \oplus x_9$	1	67	$x_{98} = x_{15} \oplus x_{64}$	2
26	$x_{57} = x_{52} \oplus x_{56} // y_9$	3	68	$x_{99} = x_{95} \oplus x_{98} // y_{15}$	3
27	$x_{58} = x_7 \oplus x_{19}$	1	69	$x_{100} = x_{44} \oplus x_{96}$	2
28	$x_{59} = x_{54} \oplus x_{58} // y_7$	3	70	$x_{101} = x_{65} \oplus x_{100} // y_5$	3
29	$x_{60} = x_{13} \oplus x_{31}$	1	71	$x_{102} = x_{19} \oplus x_{38}$	2
30	$x_{61} = x_{51} \oplus x_{60}$	2	72	$x_{103} = x_3 \oplus x_{60}$	2
31	$x_{62} = x_1 \oplus x_{61} // y_1$	3	73	$x_{104} = x_{102} \oplus x_{103} // y_{19}$	3
32	$x_{63} = x_{46} \oplus x_{61} // y_{29}$	3	74	$x_{105} = x_7 \oplus x_{31}$	1
33	$x_{64} = x_9 \oplus x_{27}$	1	75	$x_{106} = x_{15} \oplus x_{17}$	1
34	$x_{65} = x_{32} \oplus x_{64}$	2	76	$x_{107} = x_{105} \oplus x_{106} // y_{31}$	2
35	$x_{66} = x_7 \oplus x_{65} // y_{23}$	3	77	$x_{108} = x_{36} \oplus x_{70}$	2
36	$x_{67} = x_0 \oplus x_{16}$	1	78	$x_{109} = x_{41} \oplus x_{92}$	2
37	$x_{68} = x_{10} \oplus x_{28}$	1	79	$x_{110} = x_{108} \oplus x_{109} // y_4$	3
38	$x_{69} = x_{67} \oplus x_{68} // y_{16}$	2	80	$x_{111} = x_{12} \oplus x_{30}$	1
39	$x_{70} = x_8 \oplus x_{24}$	1	81	$x_{112} = x_2 \oplus x_{111}$	2
40	$x_{71} = x_{18} \oplus x_{67}$	2	82	$x_{113} = x_{74} \oplus x_{112} // y_{18}$	3
41	$x_{72} = x_{70} \oplus x_{71} // y_{24}$	3	83	$x_{114} = x_{34} \oplus x_{46} // y_{21}$	3
42	$x_{73} = x_6 \oplus x_{26}$	1	84	$x_{115} = x_{34} \oplus x_{107} // y_{13}$	3