

Addition-Efficient MDS Matrices from Superconcentrators (Full Version)

Jooyoung Lee^{1*}, Seungmin Park¹, and Mincheol Son^{2**}

¹ KAIST, Daejeon, Korea,
{hicalf, smpak}@kaist.ac.kr

² DGIST, Daegu, Korea, encrypted.def@dgist.ac.kr

Abstract. MDS matrices are a key structure for providing optimal diffusion in symmetric primitives. However, the theoretical analysis of their cost remains limited. This issue is particularly relevant to arithmetization-oriented permutations, where designs often either use costly MDS matrices or sacrifice the MDS property to reduce the number of constraints. This paper studies the number of fan-in-two additions needed to implement MDS matrices. We represent fan-in-two addition constraints by a directed acyclic graph and derive lower bounds on the number of additions using the established result that any such computation graph implementing an MDS matrix must be a superconcentrator. Building on size-reduction lemmas for superconcentrators, we present a recursive algorithm that improves both lower and upper bounds for $t \times t$ matrices with $t \leq 8$. As a result, we obtain explicit MDS matrices over large primes for $t = 3, 4, 5, 6, 7, 8$, requiring 5, 8, 12, 16, 21, 26 additions, respectively. These bounds are tight for $t \leq 6$. We also use the same superconcentrator graphs as templates for MDS matrices with k -bit words. For $t = 5, 6, 7$, our matrices require fewer XORs than the state of the art for most considered parameter choices in this line of work.

Keywords: MDS matrix · Superconcentrator · ZK-friendly hash functions · Plonk · Linear layer

1 Introduction

Symmetric primitives such as block ciphers and cryptographic permutations are typically based on a substitution-permutation network (SPN), where nonlinear operations are alternated with linear diffusion layers. The role of the linear layer is to spread local changes across the state, and maximum distance separable (MDS) matrices are a standard choice since they provide optimal branch number. The affine part of such a round can be written as

$$\text{LinLayer}(\vec{x}) = M\vec{x} + \vec{c}$$

* Jooyoung Lee was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00399491, Development of Privacy-Preserving Multiparty Computation Techniques for Secure Multiparty Data Integration).

** This work was done while Mincheol Son was a PhD student at KAIST.

where $M \in \mathbb{F}^{t \times t}$ and $\vec{c} \in \mathbb{F}^t$. The constant vector $\vec{c}$ is used as a round constant, whereas the matrix M determines the diffusion of the state. Choosing M to be MDS prevents sparse input differences from remaining sparse after the linear layer.

Most arithmetization-oriented permutations are also based on a substitution-permutation network. In early ZK-friendly designs such as Poseidon [9] and Rescue [2], additions were often treated as negligible compared to multiplications. However, in a Plonkish arithmetization [6] of an acyclic fan-in-two circuit, additions also contribute to the total gate count. A straightforward $t \times t$ matrix-vector multiplication requires $t(t-1)$ additions in total. This computational overhead becomes significant as the state size increases, leading several ZK-friendly hash functions to employ non-MDS matrices for larger states to enhance efficiency. For instance, Arion [22], Griffin [7], and Poseidon2 [10] adopt non-MDS matrices when $t \geq 5$ to prioritize computational efficiency. While this is a reasonable engineering trade-off, this approach introduces potential vulnerabilities, as non-MDS matrices may possess underlying algebraic structures that are otherwise avoided by using generic MDS matrices. A recent attack on Poseidon2 and Poseidon2b exploits the non-MDS matrices used in their linear layers to improve algebraic attacks [20].

On the other hand, the number of additions of an MDS matrix is not necessarily as high as the naive implementation suggests. Depending on the matrix, fewer additions may be sufficient. For instance, for

$$M = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}, \quad (1)$$

the naive computation uses 6 additions. On the other hand, by first computing

$$s = x_1 + x_2 + x_3,$$

one obtains $M\vec{x} = (s + x_1, s + x_2, s + x_3)$ using only 5 additions. From this perspective, heuristic methods for minimizing the addition count of MDS matrices have been explored, primarily over $\mathbb{F}_{2^k}$ in the context of lightweight cryptography [5, 11, 13, 15, 18, 23, 25, 27, 28]. Over large prime fields, the designers of Polocolo also found addition-efficient MDS matrices through heuristic search as part of their ZK-friendly design [12].

1.1 Our Contribution

In this paper, we first formulate the construction of addition-efficient MDS matrices as a graph-theoretic problem using the established connection between MDS matrices and superconcentrators. The computation graph of any fan-in-two implementation of a $t \times t$ MDS matrix must be a *superconcentrator* with t input and t output nodes, where any set of r input nodes and any set of r output nodes can be connected by r pairwise node-disjoint directed paths for every $r \leq t$. Its number of non-input nodes equals the number of additions. This connection

allows us to study addition complexity through the sizes of superconcentrators. Building on the classical size-reduction arguments of Lev [16], we then develop a recursive algorithm for superconcentrators, prove its completeness, and use it to improve both upper and lower bounds for $t \times t$ matrices. By assigning concrete coefficients to the resulting graphs, we obtain addition-efficient MDS matrices over a finite field. The corresponding addition bounds are tight for $t \leq 6$. We also instantiate the same graphs for MDS matrices with k -bit words for $t = 5, 6, 7$ by heuristically assigning coefficient blocks.

Our primary cost metric is the number of fan-in-two additions. Only additions are counted, while scalar multiplications by fixed field elements are treated as free. For MDS matrices with k -bit words, we instead compare the number of bit-level XOR gates. We provide explicit small-integer edge weightings that yield MDS matrices over every prime field $\mathbb{F}_p$ with $p \geq 2^{19}$. The resulting matrices require 5, 8, 12, 16, 21, 26 additions for $t = 3, 4, 5, 6, 7, 8$, respectively. The corresponding addition bounds are tight for $t \leq 6$. In particular, our $t = 7$ construction establishes the new upper bound $s(7, 7) \leq 21$, improving on previous results. Table 1 compares the addition counts. We note a structural difference in this comparison. The constructions in this work, in Poloccolo [12], and in Tian et al. [24] allow auxiliary variables for storing intermediate sums. Constructions based on in-place row-XOR operations instead update one of the existing state words at each step without introducing auxiliary variables. Thus, the in-place results are obtained under a more restrictive computational model.

Table 1. Number of additions for MDS matrices. The Auxiliary variables column indicates whether additional intermediate variables are allowed. The Setting column indicates the domain considered in each reference. For references that consider MDS matrices with k -bit words, the entries count word-level XOR operations, excluding the cost of coefficient multiplications. The values in bold match the lower bounds in Table 2.

Ref.	Number of additions						Auxiliary variables	Setting
	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = 7$	$t = 8$		
Ours	5	8	12	16	21	26	Allowed	$\mathbb{F}_p$ and k -bit words
[12]	5	8	13	17	24	31	Allowed	$\mathbb{F}_p$
[28]	–	12	20	30	42	56	Not allowed	
[15]	–	8	15	18	28	–	Not allowed	
[11]	–	8	15	18	28	–	Not allowed	k -bit words
[23]	–	8	–	18	–	28	Not allowed	
[13]	–	8	13	17	23	26	Not allowed	
[24]	–	8	12	16	22	27	Allowed	

Our approach also improves theoretical lower bounds on the number of additions (see Table 2). The resulting lower bounds are tight for $t \leq 6$ and remain close to the upper bounds for $t \leq 8$.

Table 2. Theoretical lower bounds for MDS matrices.

Ref.	Theoretical lower bound					
	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = 7$	$t = 8$
Ours	5	8	12	16	20	23
[16]	5	8	11*	14	17	20

*Lev obtained the conditional bound 12 for $t = 5$ but the supporting proof was incomplete because the underlying classification was not exhaustive.

Our superconcentrator graph-based approach can be applied to search for XOR-efficient MDS matrices with k -bit words. Table 3 summarizes the resulting XOR counts. For $t = 5, 6, 7$, our matrices require fewer XORs than the state of the art for all considered parameter choices except $(t, k) = (6, 4)$ and $(7, 4)$.

The source code and computational data underlying our results are available at <https://github.com/KAIST-CryptLab/AC26-MDS-matrix-artifacts>.

1.2 Related Work

Addition- and XOR-efficient MDS matrices have been widely studied for symmetric primitives, especially in lightweight cryptography [11, 14, 15, 18, 23, 25, 27, 28]. In this line of work, Duval and Leurent studied MDS matrices with lightweight circuits and optimized the number of XOR operations needed to implement an MDS matrix [5]. They also found addition-efficient MDS matrices for $t \leq 4$. This search is exhaustive under the constraints they imposed. Most ZK-friendly hash functions use these MDS matrices when $t \leq 4$ [2, 7–10, 22].

Recently, He et al. [13] studied XOR-efficient MDS matrices with k -bit words. Their method is to represent a $t \times t$ MDS matrix as a sequence of elementary row operations. The main operations are performed on k -bit words in the form of $x_i \leftarrow x_i \oplus Cx_j$, where a source row is multiplied by a coefficient and then XORed into a target row. To minimize these row-XOR operations, they first search for MDS matrix patterns and then optimize the coefficients C . This approach yields efficient MDS matrices across various sizes and word lengths.

Tian, Feng, and Li [24] developed a search based on implementation trees, allowing intermediate values to be reused in operations of the form $z = Ax \oplus By$. They constructed MDS matrices with 8-bit words requiring 12, 16, and 22 word-XOR operations for $t = 5, 6, 7$, respectively. For these 8-bit implementations, the total costs including coefficient multiplications are 114, 148, and 218 bit-level XOR gates, respectively. They also report an exhaustive exclusion of implementations with fewer than 12 word-XOR operations for $t = 5$ within their block implementation model.

Over large prime fields, Poloccolo’s designers were the first to propose addition-efficient MDS matrices for $5 \leq t \leq 8$ by heuristic search in the context of ZK-friendly hash functions [12]. Compared to the naive implementation, their matrices require about 45% fewer additions.

Table 3. Comparison of the number of XORs for $t \times t$ MDS matrices with k -bit words. Parenthetical names distinguish multiple constructions from the same reference. A dash indicates no reported entry for that reference and parameter. For the Ours rows, the parenthesized values indicate the difference from the previous best reported XOR count.

t	Ref.	Number of XORs				
		$k = 4$	$k = 8$	$k = 16$	$k = 32$	$k = 64$
5	Ours	54 ($\downarrow 11$)	103 ($\downarrow 11$)	197 ($\downarrow 18$)	389 ($\downarrow 34$)	773 ($\downarrow 66$)
	[28]	95	175	335	655	1295
	[25]	—	155	—	—	—
	[18]	—	150	—	—	—
	[11]	130	140	—	—	—
	[13]	65	117	215	423	839
	[24]	—	114	—	—	—
6	Ours	—	144 ($\downarrow 4$)	265 ($\downarrow 25$)	521 ($\downarrow 32$)	1033 ($\downarrow 64$)
	[28] (LFSR)	150	270	510	990	1950
	[28] (GFS)	—	216	360	612	1188
	[25]	—	186	—	—	—
	[11]	—	180	—	—	—
	[23]	90	156	294*	582*	1158*
	[13]	—	154	290	553	1097
	[24]	—	148	—	—	—
7	Ours	—	212 ($\downarrow 6$)	370 ($\downarrow 44$)	689 ($\downarrow 70$)	1361 ($\downarrow 134$)
	[28]	210	378	714	1386	2730
	[25]	—	378	—	—	—
	[15]	—	329	—	—	—
	[11]	—	315	—	—	—
	[13]	—	227	414	759	1495
	[24]	—	218	—	—	—

* Entries marked with an asterisk follow the calculations reported in [13].

2 Preliminaries

2.1 Notation

For a prime power q , a finite field with q elements is denoted $\mathbb{F}_q$. Vectors are written with arrows, as in $\vec{x} = (x_1, \dots, x_t)$. The parameter t denotes the size of the matrix, so an MDS matrix in this paper has size $t \times t$.

2.2 MDS Matrices

We use the following standard definition of an MDS matrix over a finite field.

Definition 1 (MDS matrix over a finite field). *Let $\mathbb{F}$ be a finite field. A matrix $M \in \mathbb{F}^{t \times t}$ is called maximum distance separable (MDS) if every square submatrix of M is invertible over $\mathbb{F}$. Equivalently, the linear map $\vec{x} \mapsto M\vec{x}$ has an optimal branch number $t + 1$.*

In addition to MDS matrices over a finite field, we also consider linear layers with k -bit words. A linear layer over $\mathbb{F}_{2^k}$ can be viewed as a linear layer in the k -bit word setting: with a fixed basis of $\mathbb{F}_{2^k}$ over $\mathbb{F}_2$, each field element is identified with a vector in $\mathbb{F}_2^k$, and then multiplication by a fixed nonzero element of $\mathbb{F}_{2^k}$ is represented by an invertible $k \times k$ matrix over $\mathbb{F}_2$. Therefore, a $t \times t$ matrix over $\mathbb{F}_{2^k}$ can be represented as a $t \times t$ block matrix over $\mathbb{F}_2$, where each block is a $k \times k$ binary matrix. This motivates the following block-level definition of the MDS property.

Definition 2 (MDS matrix with k -bit words). *Let M be a $t \times t$ block matrix whose blocks are $k \times k$ matrices over $\mathbb{F}_2$. The matrix M is called maximum distance separable with k -bit words if every $r \times r$ block submatrix of M is invertible as an $rk \times rk$ matrix over $\mathbb{F}_2$.*

For MDS matrices with k -bit words considered in this paper, the blocks are treated as $k \times k$ binary linear maps over $\mathbb{F}_2$ and need not be induced by field multiplication in $\mathbb{F}_{2^k}$.

2.3 Addition Constraints and Computation Graphs

We model the computation of $M\vec{x}$ by addition constraints. Each constraint has a linear relation of the form

$$w_c = c_a w_a + c_b w_b$$

where $c_a, c_b \in \mathbb{F}$, and w_a, w_b are either input variables or previously computed witnesses. From these constraints, we construct a directed acyclic graph that records which variables are used to compute each new variable. For each constraint, we place two directed edges into w_c . These are the edge (w_a, w_c) with weight c_a and the edge (w_b, w_c) with weight c_b . The resulting graph is called the *computation graph* of the constraints.

As a simple illustrative example, Figure 1 shows this construction for the matrix M in Equation (1). The two intermediate witnesses first compute $w_2 = x_1 + x_2 + x_3$, and the three outputs are then obtained as $y_i = w_2 + x_i$. Hence the resulting linear map is $M\vec{x}$.

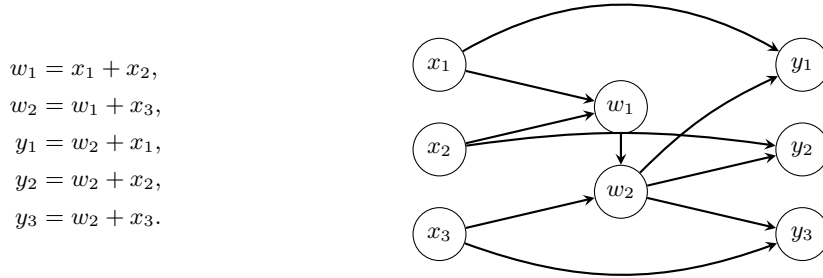


Fig. 1. Addition constraints and their computation graph, where x_i are input variables and y_i are output variables. All edge weights are equal to 1.

3 Superconcentrators and MDS Matrices

Section 2.3 associates addition constraints with computation graphs. A related graph-theoretic concept is a *superconcentrator*, which captures strong connectivity between input and output nodes. Superconcentrators have been extensively studied in connection with network size and computational complexity [17, 21, 26]. In particular, Lev [16] investigated small superconcentrators under the same fan-in-two node-count model used here. Lev and Valiant [17] studied lower bounds on additive complexity in this model and demonstrated how enumerating small superconcentrators and then assigning scalar coefficients to their edges can yield addition-efficient algorithms for cyclic convolution and the discrete Fourier transform.

In this section, we review superconcentrators, their connection to MDS matrices [4], and the classical size-reduction arguments underlying our recursive enumeration algorithm.

3.1 Superconcentrators

Informally, a superconcentrator ensures that any number of selected inputs can be connected to the same number of selected outputs by pairwise node-disjoint paths. Superconcentrators were introduced in the study of sparse networks with robust routing properties and have been used in several areas of theoretical computer science, including lower bounds for network size, parallel computation, and circuit complexity [1, 16, 21].

Definition 3 (Superconcentrator). Let $G = (V, E)$ be a directed acyclic graph with n input nodes $X = \{x_1, \dots, x_n\}$ and m output nodes $Y = \{y_1, \dots, y_m\}$. The graph G is called an (n, m) -superconcentrator if, for every pair of subsets $A \subseteq X$ and $B \subseteq Y$ satisfying $|A| = |B|$, there exist $|A|$ pairwise node-disjoint directed paths from A to B , each starting at a distinct node of A and ending at a distinct node of B .

Remark. According to the definition above, no restriction is placed on the indegrees of the nodes in a superconcentrator. However, in this paper, unless stated otherwise, to satisfy the fan-in-two addition constraints, we consider superconcentrators where every input node has indegree 0, every non-input node has indegree 2, and output nodes are permitted to have outgoing edges. We also assume that the input and output node sets are disjoint.

3.2 MDS Matrices to Superconcentrators

We use the following result of Drucker and Li [4] to relate MDS matrices to superconcentrators.

Lemma 1 ([4, Lemma 5.3]). Suppose that a matrix $M \in \mathbb{F}^{t \times t}$ is represented by addition constraints whose computation graph is G . If M is MDS, then G is a (t, t) -superconcentrator.

Lemma 1 allows us to derive lower bounds on the number of fan-in-two additions from lower bounds on the number of non-input nodes in superconcentrators. The same result holds for MDS matrices with k -bit words, including noncommuting binary blocks, by applying the same vertex-cut argument to the word-level computation graph. Drucker and Li also show that every superconcentrator admits suitable edge weights over sufficiently large finite fields [4, Lemma 5.4]. In Section 6, we provide explicit coefficient assignments and verify the MDS property of the resulting matrices over large prime fields ($p \geq 2^{19}$) and with k -bit words.

3.3 Size-Reduction Lemmas

In this section, we review classical reductions of Lev [16] that relate an (n, m) -superconcentrator to smaller superconcentrators. The input and output reductions are stated as Lemmas 4 and 5, corresponding to [16, Lemma 3.1] and [16, Lemma 5.1], respectively. For readability, two claims utilized in the proof of [16, Lemma 3.1] are presented as independent lemmas, namely Lemmas 2 and 3.

Lemmas 4 and 5 yield the inequalities

$$s(n, m) \geq s(n - 1, m) + 2 \quad \text{and} \quad s(n, m) \geq s(n, m - 1) + 1,$$

respectively. In Section 4, we reverse the graph reductions underlying these inequalities to generate candidate graphs from smaller superconcentrators, yielding a recursive enumeration algorithm. We present the proofs here because the constructions they provide are used directly in this algorithm.

Notation. For convenience in the statements and proofs, we fix the following notation for superconcentrators. A node that is neither an input node nor an output node is called an internal node. The size of a superconcentrator is the number of its internal and output nodes; this is the number of addition gates in the associated constraints. The node set of G is denoted by $V(G)$. An edge from u to v is denoted by (u, v) . The predecessor and successor sets of u are denoted by $\text{Pre}(u)$ and $\text{Succ}(u)$, and its indegree and outdegree in G are denoted by $\text{indeg}_G(u)$ and $\text{outdeg}_G(u)$. The subscript G is omitted when the graph is clear from context. Let $SC(n, m, c)$ denote the set of superconcentrators with n input nodes, m output nodes, and size c . Define

$$s(n, m) := \min\{c : SC(n, m, c) \neq \emptyset\}.$$

Then $s(n, m)$ denotes the minimum size of such a superconcentrator. It is emphasized once again that in this notation, the superconcentrators under consideration are required to have indegree 2 at every non-input node.

Before stating the reduction lemmas, we note that Lemma 2 temporarily relaxes the indegree-2 convention. Although the final superconcentrators considered in this paper have indegree 2 at every non-input node, this lemma and its reverse step **AddNode** allow non-input nodes of indegree 1. Such graphs appear only as intermediate objects in the reduction and expansion procedures.

Lemma 2. *Let $G = (V, E)$ be an (n, m) -superconcentrator of size c such that $n, m \geq 2$ and every non-input node has indegree 1 or 2. If there exists a non-input node u of indegree 1 in G , then there exists an (n, m) -superconcentrator $G' = (V \setminus \{u\}, E')$ of size $c - 1$ such that for every node $v \in V \setminus \{u\}$,*

$$\text{indeg}_{G'}(v) \leq \text{indeg}_G(v).$$

Proof. Let p be the unique predecessor of u , and let $s_1, \dots, s_k$ be the successors of u . Since every input node has indegree 0, the node u is either an internal node or an output node. We first show that if u is an output node, then p is an internal node.

Assume that both p and u are output nodes. Since $n, m \geq 2$, take any two distinct input nodes and consider the two output nodes p and u . Since G is a superconcentrator, there must exist two pairwise node-disjoint directed paths connecting these two inputs to $\{p, u\}$. However, every directed path ending at u must pass through p . Hence no two such paths can be node-disjoint, a contradiction. Moreover, if u is an output node, then p cannot be an input node. Since $n \geq 2$, there is another input node distinct from p , and no directed path from that input node to u can pass through p . Therefore, if u is an output node, then p is an internal node.

We now construct G' as follows:

1. Start with $G' := G$.
2. Remove the node u and all edges incident to u .
3. For each successor s_i of u , add the edge (p, s_i) unless this edge is already present in G .

4. If u was an output node, replace it by p in the output node set.

By construction, one node is removed, so the size decreases from c to $c - 1$. Moreover, the only new edges added are outgoing edges from p , so the indegree of every remaining node in G' is at most its indegree in G . Also, G' is still acyclic: if (p, s_i) is added, then in G there is already a directed path $p \rightarrow u \rightarrow s_i$, so p precedes s_i in every topological ordering of G .

It remains to prove that G' is still an (n, m) -superconcentrator. Let A be any subset of the input nodes of G' and let B be any subset of the output nodes of G' such that $|A| = |B|$. It must be shown that there exists a family of $|A|$ pairwise node-disjoint directed paths from A to B in G' .

Define a subset B^* of the output nodes of G by

$$B^* := \begin{cases} B, & \text{if } u \text{ is not an output node or if } p \notin B, \\ (B \setminus \{p\}) \cup \{u\}, & \text{if } u \text{ is an output node and } p \in B. \end{cases}$$

Then $|A| = |B^*|$, and since G is an (n, m) -superconcentrator, there exists a family $\mathcal{P}$ of pairwise node-disjoint directed paths from A to B^* in G .

We now convert $\mathcal{P}$ into a path family in G' by modifying any path that uses u . If a path contains a segment $p \rightarrow u \rightarrow s_i$, then this segment is replaced by the edge $p \rightarrow s_i$, which belongs to G' by construction. If instead the path ends at u , then this can occur only when u is an output node and $p \in B$; in this case, the final edge $p \rightarrow u$ is deleted, so the path now ends at the corresponding output node p of G' . The resulting paths lie in G' and remain pairwise node-disjoint.

Thus $\mathcal{P}$ yields a family of pairwise node-disjoint directed paths from A to B in G' . Since A and B are arbitrary, G' is an (n, m) -superconcentrator. $\square$

Figure 2 illustrates the construction of G' in Lemma 2. The indegree-1 node u is bypassed by connecting its unique predecessor p directly to its successors.

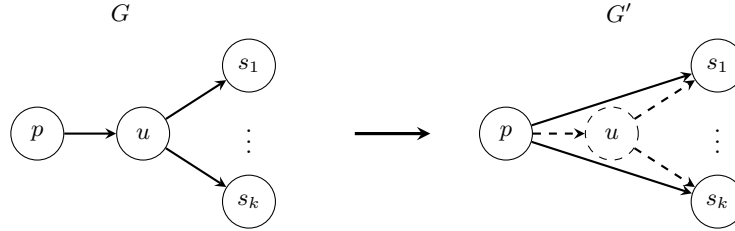


Fig. 2. The construction of G' in Lemma 2. The indegree-1 node u is removed, and each successor of u is connected directly from its predecessor p ; if u is an output node, then p becomes the corresponding output node in G' .

We also clarify that in Step 3 of the construction of G' from G , if the edge (p, s_i) is already present, as shown in Figure 3, then no new edge is added for that successor.

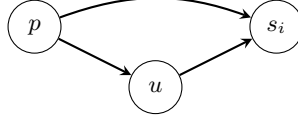


Fig. 3. A local pattern where the edge (p, s_i) is already present before removing u .

We next introduce another lemma used in the size-reduction argument.

Lemma 3. *Let G be an (n, m) -superconcentrator such that $n, m \geq 2$. Suppose that every internal node and every output node of G has indegree 2. Then there exists an input node of G whose outdegree is at least 2.*

Proof. Let w be the earliest non-input node in a topological ordering of G . Since w is the earliest such node, all of its predecessors must be input nodes. By the indegree assumption, w has two distinct predecessors; denote them by x_1 and x_2 .

Assume that both x_1 and x_2 have outdegree at most 1. Since both x_1 and x_2 are predecessors of w , their unique outgoing edges must be (x_1, w) and (x_2, w) . Hence every directed path starting from either x_1 or x_2 must pass through w .

Now choose any two distinct output nodes y_1 and y_2 . By the superconcentrator property, there must exist two pairwise node-disjoint directed paths from $\{x_1, x_2\}$ to $\{y_1, y_2\}$. However, this is impossible, since every such path must pass through the common node w .

This contradiction shows that the assumption was false. Therefore, at least one predecessor of w has outdegree at least 2. In particular, at least one input node of G has outdegree at least 2. $\square$

Combining Lemma 2 and Lemma 3, we obtain the relation between $s(n, m)$ and $s(n - 1, m)$ stated in Lemma 4.

Lemma 4. *For $n \geq 3$ and $m \geq 2$,*

$$s(n, m) \geq s(n - 1, m) + 2.$$

Proof. The proof starts with a minimum-size (n, m) -superconcentrator G and constructs an $(n - 1, m)$ -superconcentrator G' whose size is smaller by at least two. This directly gives the desired lower bound on $s(n, m)$.

Let G be a minimum-size (n, m) -superconcentrator. Let w be the earliest non-input node in a topological ordering of G . As shown in the proof of Lemma 3, there exists a predecessor x of w such that

$$\text{outdeg}_G(x) \geq 2.$$

Let H be the graph obtained from G by removing the input node x and all edges incident to x . Since x is an input node, no directed path starting from any other input node can pass through x . Hence, for every subset A of the remaining

input nodes and every subset B of the output nodes with $|A| = |B|$, every family of pairwise node-disjoint directed paths from A to B in G is still present in H .

Moreover, since every non-input node has indegree 2, deleting x decreases the indegree of each successor of x from 2 to 1. Therefore, H contains at least $\text{outdeg}_G(x)$ nodes of indegree 1, and in particular at least two such nodes.

Starting from H , repeatedly apply the procedure of Lemma 2 until no node of indegree 1 remains. Let G' denote the resulting graph. Each application of Lemma 2 preserves the superconcentrator property, so G' is an $(n-1, m)$ -superconcentrator.

Since the size function counts only non-input nodes, removing the input node x does not affect the size. On the other hand, the repeated application of Lemma 2 removes at least the $\text{outdeg}_G(x)$ successors of x , and thus removes at least two non-input nodes. Consequently,

$$|G'| \leq s(n, m) - 2.$$

Therefore,

$$s(n-1, m) \leq |G'| \leq s(n, m) - 2,$$

which implies

$$s(n, m) \geq s(n-1, m) + 2. \quad \square$$

In the proof of Lemma 4, the successors of the deleted input node x have indegree 1 in H , and hence these successors are removed by applying Lemma 2. The repetition may continue beyond these immediate successors. For example, suppose s is a successor of x with unique predecessor p and a successor t . If the edge (p, t) is already present, then removing s deletes (s, t) without adding a new edge into t . Thus t may become the next indegree-1 node to remove. This explains why the procedure is repeated until no indegree-1 node remains.

Using an approach similar to Lemma 4, the following lemma gives a relation between $s(n, m)$ and $s(n, m-1)$.

Lemma 5. *For $n, m \geq 2$,*

$$s(n, m) \geq s(n, m-1) + 1.$$

Proof. Let G be an (n, m) -superconcentrator of minimum size $s(n, m)$, and let y be the last node in a topological ordering of G . Since every input node has a directed path to an output node, y is a non-input node. If y were an internal node, then removing y would produce an (n, m) -superconcentrator of size $s(n, m) - 1$, contradicting the minimality of $s(n, m)$. Hence, y must be an output node.

Let G' be the graph obtained from G by removing y and all edges incident to y . It remains to show that G' is an $(n, m-1)$ -superconcentrator. Consider arbitrary subsets A of the input nodes and B of the output nodes of G' such that $|A| = |B|$. Since G is an (n, m) -superconcentrator and the output nodes of G' are also output nodes of G , there exists a family $\mathcal{P}$ of $|A|$ pairwise node-disjoint directed paths from A to B in G .

Since y is the last node in a topological ordering, it has no outgoing edge. Hence any directed path that reaches y must end at y . But every path in $\mathcal{P}$ ends in B , and $y \notin B$. Thus no path in $\mathcal{P}$ uses y , so the same family $\mathcal{P}$ is present in G' . Since A and B were arbitrary, G' is an $(n, m - 1)$ -superconcentrator.

Consequently,

$$s(n, m - 1) \leq s(n, m) - 1,$$

which is equivalent to

$$s(n, m) \geq s(n, m - 1) + 1. \quad \square$$

4 Recursive Algorithm for Obtaining Superconcentrators

The two reductions in Lemma 4 and Lemma 5 yield a recursive construction of (n, m) -superconcentrators from smaller instances. In the forward direction, Lemma 4 removes one input node from an (n, m) -superconcentrator and then repeatedly removes indegree-1 nodes, producing an $(n - 1, m)$ -superconcentrator. Similarly, Lemma 5 removes one output node and produces an $(n, m - 1)$ -superconcentrator. Reversing the constructions used in the proofs of these reductions therefore gives candidate (n, m) -superconcentrators from graphs in $SC(n - 1, m, \cdot)$ and $SC(n, m - 1, \cdot)$. This section formalizes these reverse operations as algorithms for generating candidate superconcentrators, and then combines them into a recursive algorithm for computing the family $SC(n, m, c)$.

4.1 Expansion by One Input Node

Reverse step for one non-input node. **AddNode** outputs all candidates obtained by adding one new non-input node u of indegree 1 to a superconcentrator G , whose non-input nodes may have indegree 1 or 2, so as to reverse the construction used in the proof of Lemma 2. It inserts u after a chosen predecessor p and makes selected successors of p become successors of u , as illustrated in Figure 4. If p is an output node, the algorithm also considers the case in which u becomes an output node and p becomes internal.

In **AddNode**, the set S covers the reverse operation. For a successor $s \in S$, the edge (p, s) of G is removed and replaced by the two-edge path $p \rightarrow u \rightarrow s$. However, this does not cover all graphs that can be reduced to the current graph G by applying the Lemma 2. In the third step of the construction of G' in Lemma 2, the edge (p, s_i) may already be present, and then no new edge is added. This case must also be handled in reverse operation.

The additional set T handles this case precisely. Before applying Lemma 2 to remove u , a successor t of u may already have been connected from p by the edge (p, t) . When u is removed, the edge (u, t) is deleted and no new edge needs to be added from p to t . Thus t appears in G as a successor of p with indegree 1. To reverse this situation, **AddNode** must keep the edge (p, t) and add only the missing edge (u, t) , as illustrated in Figure 5. The set of such nodes is denoted by T . It is chosen as a subset of $U_p(S)$, where $U_p(S)$ consists of the nodes in $\text{Succ}_G(p) \setminus S$ whose indegree in G is 1.

Algorithm 1: $\text{AddNode}(G, d_{\text{cand}})$ — Generate all one-step expansions of G obtained by reversing the construction used in the proof of Lemma 2 with predecessor chosen from d_{cand}

Input : An (n, m) -superconcentrator G of size c with $n, m \geq 2$ whose non-input nodes have indegree 1 or 2, and a set d_{cand} of candidate predecessor nodes of G

Output: A list of pairs (G', u) , where G' is an (n, m) -superconcentrator of size $c + 1$ whose non-input nodes have indegree 1 or 2, and u is the newly added node

```

1 Initialize  $L$  as the empty list
2 for each node  $p \in d_{\text{cand}}$  do
3   for each set  $S \subseteq \text{Succ}_G(p)$  do
4     Set  $U_p(S)$  to the set  $\{t \in \text{Succ}_G(p) \setminus S \mid \text{indeg}_G(t) = 1\}$ 
5     for each set  $T \subseteq U_p(S)$  do
6       Let  $G_1$  be a copy of  $G$ 
7       Add a new internal node  $u$  to  $G_1$ 
8       Add the edge  $(p, u)$  to  $G_1$ 
9       for each node  $s \in S$  do
10        Remove the edge  $(p, s)$  from  $G_1$ 
11        Add the edge  $(u, s)$  to  $G_1$ 
12       for each node  $t \in T$  do
13        Add the edge  $(u, t)$  to  $G_1$ 
14       Append  $(G_1, u)$  to  $L$ 
15       if  $p$  is an output node then
16         Let  $G_2$  be a copy of  $G_1$ 
17         Let  $u_2$  be the copy of  $u$  in  $G_2$ 
18         Replace the output node  $p$  by  $u_2$  in  $G_2$ 
19         Append  $(G_2, u_2)$  to  $L$ 
20 return  $L$ 

```

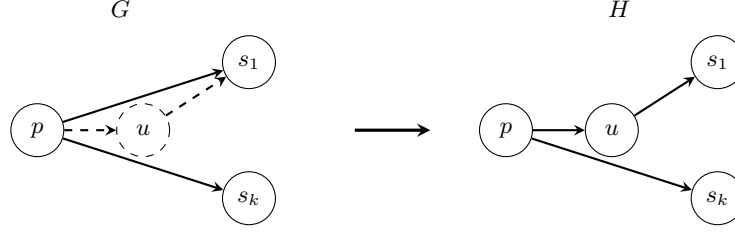


Fig. 4. The basic expansion performed by **AddNode**. A new indegree-1 node u is inserted after p . A selected successor of p such as s_1 becomes a successor of u . Another successor such as s_k may remain a successor of p . The dashed node and edges on the left indicate the part added by the expansion.

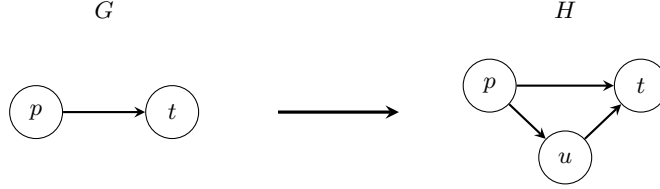


Fig. 5. The case accounted for by the set T in **AddNode**.

Proposition 1. *Let G be an (n, m) -superconcentrator of size c such that $n, m \geq 2$ and every non-input node has indegree 1 or 2, and let d_{cand} be a set of candidate predecessor nodes of G . Then **AddNode** outputs all pairs (H, u) satisfying the following conditions:*

1. H is an (n, m) -superconcentrator of size $c + 1$ obtained from G by adding a single new non-input node u whose unique predecessor p lies in d_{cand} ;
2. every non-input node of H has indegree 1 or 2;
3. applying the reduction of Lemma 2 to u recovers G .

*In particular, **AddNode** outputs the admissible one-step expansions of G with a predecessor chosen from d_{cand} .*

Proof. We first show that every output satisfies the three stated conditions. Then, we prove completeness by showing that every pair satisfying these conditions is output by the algorithm. Let (H, u) be an output of **AddNode**. By construction, this output is determined by some node $p \in d_{\text{cand}}$, some set $S \subseteq \text{Succ}_G(p)$, and some set $T \subseteq U_p(S)$.

(Condition 1.) The algorithm adds a single new non-input node u to G and no other node, so H has size $c + 1$. The only edge into u is (p, u) , where $p \in d_{\text{cand}}$, and hence u has unique predecessor p . It remains to check that H is an (n, m) -superconcentrator. Let A be a set of input nodes of H and let B be a set of output nodes of H with $|A| = |B|$. If $u \in B$, let $B_G = (B \setminus \{u\}) \cup \{p\}$; otherwise let $B_G = B$. Then B_G is the corresponding output set of G . Since

G is a superconcentrator, there is a family of pairwise node-disjoint paths in G from A to B_G . We lift these paths to H by replacing each edge (p, s) with $s \in S$ by the path $p \rightarrow u \rightarrow s$. If $u \in B$, the path ending at p is extended by (p, u) . The only new node used in this lifting is u , and it is used only on the path that contains p . Since at most one path in a node-disjoint family can contain p , the lifted paths remain pairwise node-disjoint in H .

(Condition 2.) By construction, u has unique predecessor p , nodes in S keep their indegrees, nodes in T have indegree 1 in G and indegree 2 in H , and all other indegrees are unchanged. Thus every non-input node of H has indegree either 1 or 2.

(Condition 3.) Apply the reduction of Lemma 2 to the node u in H . This removes u and its incident edges, and adds the edge (p, v) for each successor v of u if that edge is not already present. For every $s \in S$, this restores the edge (p, s) that was replaced by the algorithm. For every $t \in T$, the edge (p, t) was never removed, because $T \subseteq \text{Succ}_G(p) \setminus S$; hence no new edge beyond those of G is introduced. All other edges are unchanged, and therefore the resulting graph is G .

(Completeness.) It remains to show that every pair (H, u) satisfying the three stated conditions is output by **AddNode**. Let (H, u) be such a pair. Let p be the unique predecessor of u in H . By assumption, $p \in d_{\text{cand}}$. Partition the successors of u in H into

$$\begin{aligned} S &:= \{v \in \text{Succ}_H(u) \mid (p, v) \notin E(H)\}, \\ T &:= \{v \in \text{Succ}_H(u) \mid (p, v) \in E(H)\}. \end{aligned}$$

Since applying the reduction of Lemma 2 to u recovers G , every node in $S \cup T$ is a successor of p in G . For every $t \in T$, both edges (p, t) and (u, t) belong to H . Since indegrees in H are at most 2, removing (u, t) leaves t with indegree 1 in G . Hence $T \subseteq U_p(S)$. Therefore the corresponding p , S , and T can be chosen by the algorithm. With these choices, the construction reproduces the edge set of H .

If u is an output node of H , then the reduction of Lemma 2 makes its unique predecessor p the corresponding output node in G . Thus the output-node case of the algorithm applies and produces (H, u) . If u is not an output node, the ordinary case produces (H, u) . Therefore every admissible one-step expansion is output by **AddNode**. $\square$

Expansion by one input node. **ExpSCByInput** reverses the construction used in the proof of Lemma 4. Starting from a graph $G \in SC(n, m, c)$, it first reverses the repeated removals of indegree-1 nodes by applying **AddNode** α times. In the forward construction used in the proof of Lemma 4, deleting the input node decreases the indegree of its successors to 1, and these successors are removed during the subsequent indegree-1 reductions. Thus, after the α reverse steps in **ExpSCByInput**, the newly added nodes that still have indegree 1 become the successors of the new input node.

As shown in the proof of Lemma 3, the input node deleted in Lemma 4 can be chosen as a predecessor x of the earliest non-input node w with $\text{outdeg}(x) \geq 2$.

Algorithm 2: ExpSCByInput(G, α) — Expand a graph $G \in SC(n, m, c)$ by attaching one input node and α non-input nodes

Input : $G \in SC(n, m, c)$ and an integer $\alpha \geq 2$
Output: A set of candidate graphs for $SC(n + 1, m, c + \alpha)$ that reduce to G by Lemma 4

```

1 Initialize  $L_0$  as the list containing the pair  $(G, [])$ 
  // Step 1: reverse the repeated indegree-1 removals
2 for each integer  $i$  from 1 to  $\alpha$  do
3   Initialize  $L_i$  as the empty list
4   for each pair  $(H, R) \in L_{i-1}$  do
5     if  $i = \alpha$  then
6       Set  $d_{\text{cand}}$  to the set of input nodes of  $H$ 
7     else
8       Set  $d_{\text{cand}}$  to the node set of  $H$ 
9     for each  $(H', u) \in \text{AddNode}(H, d_{\text{cand}})$  do
10      Set  $R'$  to the list obtained by appending  $u$  to  $R$ 
11      Append  $(H', R')$  to  $L_i$ 
  // Step 2: add the new input node
12 Initialize  $S_{\text{out}}$  as the empty set
13 for each pair  $(H, R) \in L_\alpha$  do
14   Set  $S_x$  to the set of nodes  $s$  such that  $s \in R$  and  $\text{indeg}_H(s) = 1$ 
15   if  $|S_x| \geq 2$  then
16     Add a new input node  $x$  to  $H$ 
17     for each node  $s \in S_x$  do
18       Add the edge  $(x, s)$  to  $H$ 
19     Add  $H$  to  $S_{\text{out}}$ 
20 return  $S_{\text{out}}$ 

```

After deleting x , the node w has indegree 1 and its remaining predecessor is also an input node. The forward reduction may remove this node w first among the indegree-1 nodes, which corresponds in the reverse direction to adding w in the last reverse step before restoring the input node. Therefore, in that last reverse step, only input nodes are used as candidate predecessors.

Proposition 2. *Let $G \in SC(n, m, c)$ and let $\alpha \geq 2$. Then ExpSCByInput outputs all the graphs H satisfying the following conditions:*

1. H has $n + 1$ input nodes, its size is $c + \alpha$, and every non-input node of H has indegree 2;
2. there exist a non-input node w and an input node x such that $x \in \text{Pre}_H(w)$, every node in $\text{Pre}_H(w)$ is an input node, $\text{outdeg}_H(x) \geq 2$, and deleting x and then applying the indegree-1 reductions beginning with w removes α non-input nodes and yields G .

In particular, `ExpSCByInput` outputs all the admissible expansions of G obtained by adding one input node and α non-input nodes.

Proof. We first show that every output satisfies the two stated conditions. Then, we prove completeness by showing that every graph satisfying these conditions is output by the algorithm. Let H be an output of `ExpSCByInput`. Then there is a sequence of pairs

$$(H_0, R_0), (H_1, R_1), \dots, (H_\alpha, R_\alpha)$$

constructed in Step 1 such that $H_0 = G$, $R_0 = []$, and H_i is obtained from H_{i-1} by one call to `AddNode`. Let u_i be the new node added in the i -th step. Then $R_i = (u_1, \dots, u_i)$. By Proposition 1, applying the reduction of Lemma 2 to u_i in H_i recovers H_{i-1} .

In Step 2, the algorithm adds a new input node x and connects it to the recorded nodes in

$$S_x = \{u_i \mid 1 \leq i \leq \alpha, \text{indeg}_{H_\alpha}(u_i) = 1\}.$$

(Condition 1.) The graph H_α is obtained from G by adding α non-input nodes, so it has size $c + \alpha$. Step 2 adds one input node and does not change the size. Hence H has $n + 1$ input nodes and size $c + \alpha$. Next, we verify the indegree condition. In the starting graph G , every non-input node has indegree 2. Consider one call to `AddNode` that adds a node u_i . In the notation of that call, the indegree of each node in S is unchanged. For each node in T , its indegree changes from 1 to 2, and all other old nodes have unchanged indegree. Thus no old node becomes a new node of indegree 1. Therefore every non-input node of indegree 1 in H_α is one of the recorded nodes in R_α . Step 2 adds an incoming edge from x to each recorded node that still has indegree 1. Therefore every non-input node of H has indegree 2.

(Condition 2.) Since u_α is added in the last reverse step, its unique predecessor in H_α is an input node. Also $\text{indeg}_{H_\alpha}(u_\alpha) = 1$, so Step 2 adds the edge (x, u_α) . Thus in H the node u_α has only input predecessors. Set $w = u_\alpha$. Since Step 2 is applied only when $|S_x| \geq 2$, we have $\text{outdeg}_H(x) \geq 2$. Deleting x from H recovers H_α , and then removing $w = u_\alpha, u_{\alpha-1}, \dots, u_1$ recovers G .

(Completeness.) It remains to show that every graph satisfying the two stated conditions is output by `ExpSCByInput`. Let H' be such a graph. Choose x' and w' as in the second condition, and denote the resulting reduction sequence by

$$H'_\alpha = H' \setminus \{x'\} \longrightarrow H'_{\alpha-1} \longrightarrow \dots \longrightarrow H'_1 \longrightarrow H'_0 = G,$$

where H'_{i-1} is obtained from H'_i by applying Lemma 2 to a node u'_i and $u'_\alpha = w'$.

We first show that Step 1 constructs the sequence up to $L_{\alpha-1}$. For every $1 \leq i < \alpha$, the candidate set is the whole node set. Since reducing H'_i at u'_i gives H'_{i-1} , the completeness of `AddNode` implies inductively that $(H'_i, (u'_1, \dots, u'_i))$ appears in L_i for every $1 \leq i < \alpha$.

We then handle the construction of L_α . In the iteration $i = \alpha$, for the current pair $(H'_{\alpha-1}, (u'_1, \dots, u'_{\alpha-1}))$, the algorithm sets d_{cand} to the set of input nodes of

Algorithm 3: $\text{GenSCByInput}(n, m, c)$ — Generate all graphs in $SC(n, m, c)$ via one-input expansion

Input : Integers n, m, c , and the sets $SC(n-1, m, c')$ for all $n \geq 3, m \geq 2$, and $0 \leq c' < c$
Output: The set $SC(n, m, c)$

- 1 Initialize S as the empty set
- 2 **for** each integer α from 2 to c **do**
- 3 **for** each graph $G \in SC(n-1, m, c-\alpha)$ **do**
- 4 **for** each graph $H \in \text{ExpSCByInput}(G, \alpha)$ **do**
- 5 **if** H is a superconcentrator **then**
- 6 Add H to S
- 7 **return** S

$H'_{\alpha-1}$. Since $x' \in \text{Pre}_{H'}(w')$ and every node in $\text{Pre}_{H'}(w')$ is an input node, deleting x' leaves w' with a unique input predecessor in H'_α . This predecessor remains an input node in $H'_{\alpha-1}$, and hence belongs to d_{cand} . Again by the completeness of AddNode , the pair $(H'_\alpha, (u'_1, \dots, u'_{\alpha-1}, w'))$ appears in L_α .

Finally, we check that Step 2 reconstructs the original input node x' . Since $H'_\alpha = H' \setminus \{x'\}$ is obtained by deleting only x' and its outgoing edges, and every non-input node of H' has indegree 2, the nodes of indegree 1 in H'_α are the successors of x' in H' . Moreover, since the reductions of Lemma 2 do not increase indegrees, any such node must be one of $u'_1, \dots, u'_\alpha$. Hence the set chosen by Step 2 is $\text{Succ}_{H'}(x')$. Since $\text{outdeg}_{H'}(x') \geq 2$ by the second condition, Step 2 accepts this set and reconstructs H' . Thus H' is output by ExpSCByInput . $\square$

Generation by one input reduction. GenSCByInput reconstructs all graphs in $SC(n, m, c)$ by reversing the construction used in the proof of Lemma 4. If $H \in SC(n, m, c)$, then by Lemma 4, removing a suitable input node and repeatedly removing nodes of indegree 1 yields a graph $G \in SC(n-1, m, c-\alpha)$ for some $\alpha \geq 2$. Conversely, for each such graph G , the procedure ExpSCByInput on input (G, α) first outputs all reverse candidates obtained by adding one input node and α non-input nodes, and the final superconcentrator test retains those candidates that belong to $SC(n, m, c)$.

Proposition 3. *Let $n \geq 3$ and $m \geq 2$. Assume that the sets $SC(n-1, m, c')$ are known for all $0 \leq c' \leq c-1$. Then GenSCByInput outputs the set $SC(n, m, c)$.*

Proof. Let H be a graph output by GenSCByInput . Then H is produced by $\text{ExpSCByInput}(G, \alpha)$ for some $\alpha \geq 2$ and some graph $G \in SC(n-1, m, c-\alpha)$. Since the algorithm keeps only those outputs that are superconcentrators, it follows that $H \in SC(n, m, c)$.

Conversely, let $H \in SC(n, m, c)$. By the reduction used in the proof of Lemma 4, it is possible to choose a non-input node w and an input node x of H such that $x \in \text{Pre}_H(w)$, every node in $\text{Pre}_H(w)$ is an input node, and

Algorithm 4: $\text{ExpSCByOutput}(G)$ — Expand a graph $G \in SC(n, m, c)$ by attaching one output node

Input : $G \in SC(n, m, c)$
Output: A collection of candidate graphs with n input nodes, $m + 1$ output nodes, and size $c + 1$

- 1 Initialize S as the empty set
// Add the new output node
- 2 **for** each choice of a predecessor pair $P_y \subseteq V(G)$ with $|P_y| = 2$ **do**
- 3 Let G' be a copy of G
- 4 Add a new output node y to G'
- 5 Add edges (p, y) for all $p \in P_y$
- 6 Add G' to S
- 7 **return** S

$\text{outdeg}_H(x) \geq 2$. Deleting x and then repeatedly removing nodes of indegree 1 yields a graph $G \in SC(n - 1, m, c - \alpha)$ for some $\alpha \geq 2$. By the completeness of ExpSCByInput , the graph H is generated by $\text{ExpSCByInput}(G, \alpha)$. Since H is itself a superconcentrator, it passes the final test and is included in the output. Therefore GenSCByInput outputs the set $SC(n, m, c)$. $\square$

4.2 Expansion by One Output Node

ExpSCByOutput reverses the construction used in the proof of Lemma 5. Since the forward reduction removes one output node, the reverse construction adds a new output node. In the present graph model, this is done by choosing two distinct predecessor nodes and introducing a new output node with these two nodes as its predecessors.

Proposition 4. *Let $G \in SC(n, m, c)$. Then ExpSCByOutput outputs all the candidate graphs H for which there exists an output node y with no outgoing edge such that removing y and its incident edges yields G .*

Proof. Let H be a graph output by ExpSCByOutput . Then H is obtained from a copy of G by choosing a two-element set $P_y \subseteq V(G)$, adding a new output node y , and adding precisely the edges (p, y) for $p \in P_y$. The new output node y has no outgoing edge, and removing y together with its incident edges recovers the original graph G .

Conversely, let H' be such a candidate graph, and let y' be the corresponding output node with no outgoing edge. When ExpSCByOutput chooses $P_{y'} = \text{Pre}_{H'}(y')$, it reconstructs H' and adds it to its output collection S . Hence every such candidate graph is output by the algorithm. $\square$

GenSCByOutput uses the one-output expansion above together with the fact that an internal node with no outgoing edge can be added or removed without affecting the superconcentrator property. The algorithm therefore has two cases.

Algorithm 5: GenSCByOutput(n, m, c) — Generate all graphs in $SC(n, m, c)$ via one-output expansion

Input : Integers n, m, c , and the sets $SC(n, m - 1, c - 1)$ and $SC(n, m, c - 1)$
Output: The set $SC(n, m, c)$

```

1 Initialize  $S$  as the empty set
  // Add an output node with no outgoing edge
2 for each graph  $G \in SC(n, m - 1, c - 1)$  do
3   for each graph  $H \in \text{ExpSCByOutput}(G)$  do
4     if  $H$  is a superconcentrator then
5       Add  $H$  to  $S$ 
  // Add an internal node with no outgoing edge
6 for each graph  $G \in SC(n, m, c - 1)$  do
7   for each choice of a predecessor pair  $P_u \subseteq V(G)$  with  $|P_u| = 2$  do
8     Let  $H$  be a copy of  $G$ 
9     Add a new internal node  $u$  to  $H$ 
10    Add edges  $(p, u)$  for all  $p \in P_u$ 
11    Add  $H$  to  $S$ 
12 return  $S$ 

```

If the target graph has an output node with no outgoing edge, it is obtained by adding this output node to a graph in $SC(n, m - 1, c - 1)$ and then testing the superconcentrator condition. If the target graph has an internal node with no outgoing edge, it is obtained by adding one such internal node to a graph in $SC(n, m, c - 1)$. These two cases cover all graphs because every finite directed acyclic graph has a node with no outgoing edge, and such a node cannot be an input node in a superconcentrator.

Proposition 5. Given $SC(n, m - 1, c - 1)$ and $SC(n, m, c - 1)$, GenSCByOutput outputs $SC(n, m, c)$.

Proof. First, every returned graph belongs to $SC(n, m, c)$. In the output-node case, the algorithm inserts a graph only after testing the superconcentrator condition. In the internal-node case, the algorithm starts from a graph in $SC(n, m, c - 1)$ and adds one internal node with no outgoing edge and with two predecessors. This preserves acyclicity, the indegree-2 condition, and the superconcentrator property. Hence every returned graph lies in $SC(n, m, c)$.

Conversely, let $H' \in SC(n, m, c)$. First suppose that H' has an output node y' with no outgoing edge. Removing y' and its incident edges gives a graph $G \in SC(n, m - 1, c - 1)$: paths to the remaining output nodes do not use y' , since y' has no outgoing edge. By the completeness of ExpSCByOutput, the algorithm outputs H' from G , and the output-node case of GenSCByOutput inserts H' because H' is a superconcentrator.

It remains to consider the case in which no output node of H' has outdegree 0. Then H' has an internal node u' with no outgoing edge. Removing u' gives

a graph $G \in SC(n, m, c - 1)$, because u' cannot lie on any path from an input node to an output node. Let $P_{u'} = \text{Pre}_{H'}(u')$. When **GenSCByOutput** chooses G and $P_{u'}$ in the internal-node case, it reconstructs H' and adds it to S . The two inclusions prove that the output of **GenSCByOutput** is $SC(n, m, c)$. $\square$

4.3 Generating Superconcentrators

Sections 4.1 and 4.2 describe how to construct the next family of superconcentrators from the families obtained after deleting one input node or one output node, respectively. Combining these two recursive steps gives an algorithm for generating the required families $SC(n, m, c)$ for $m \in \{n, n + 1\}$.

Algorithm 6 computes the families $SC(n, m, c)$ recursively, with the base case handled directly inside the procedure when $(n, m) = (2, 2)$. In this base case, the algorithm outputs all admissible graphs of size c with two input nodes and two output nodes, and then retains those graphs that are superconcentrators. For larger values of (n, m) , the recursion follows the alternating pattern:

$$(n, n) \xrightarrow{\text{GenSCByOutput}} (n, n + 1) \xrightarrow{\text{GenSCByInput}} (n + 1, n + 1).$$

Proposition 6. *GenSC outputs the set $SC(n, m, c)$ for every triple (n, m, c) such that $m \in \{n, n + 1\}$, $n, m \geq 2$, and $c \geq 0$.*

Proof. The proof is by strong induction on c , for all admissible pairs (n, m) . If $c < m$, then no graph with m output nodes has size c , so $SC(n, m, c) = \emptyset$, and the algorithm returns $\emptyset$. Thus we may assume $c \geq m$.

For $c = 2$, the only admissible pair is $(n, m) = (2, 2)$, and the claim follows from the base case, where the algorithm outputs all admissible graphs.

Now assume $c > 2$ and that the claim holds for every admissible triple with size parameter smaller than c . Fix an admissible triple (n, m, c) . If $(n, m) = (2, 2)$, the procedure is nonrecursive, and the claim follows directly because it outputs all admissible graphs in this case.

It remains to consider the recursive cases. First suppose that $m = n + 1$. Before applying **GenSCByOutput**, the required sets $SC(n, m - 1, c - 1)$ and $SC(n, m, c - 1)$ are either empty for size reasons or computed from smaller-size instances. Hence the induction hypothesis implies that the sets supplied to **GenSCByOutput** are correct. Thus, by the completeness of **GenSCByOutput**, the returned set is $SC(n, m, c)$.

Finally suppose that $m = n$ and $m \geq 3$. Before applying **GenSCByInput**, the required nonempty sets $SC(n - 1, m, c')$ with $c' < c$ are computed recursively. Since their size parameters are all smaller than c , the induction hypothesis implies that they are computed correctly. By the completeness of **GenSCByInput**, its output is $SC(n, m, c)$.

This completes the induction and proves the claim for all admissible triples (n, m, c) . $\square$

Algorithm 6: GenSC(n, m, c) — Recursively generate $SC(n, m, c)$

Input : Integers n, m, c with $m \in \{n, n+1\}$, $n, m \geq 2$, and $c \geq 0$

Output: Set $SC(n, m, c)$

```

1 if  $c < m$  then
2   return  $\emptyset$ 
3 if  $n = 2$  and  $m = 2$  then
4   Initialize  $S$  as the empty set
   // Directly generate all admissible graphs of size  $c$  with two
   // input nodes and two output nodes
5   for each directed acyclic graph  $G$  with input nodes  $x_1, x_2$ , output nodes
   //  $y_1, y_2$ ,  $c - 2$  internal nodes, and every non-input node having indegree 2
   do
6     if  $G$  is a superconcentrator then
7       Add  $G$  to  $S$ 
8   return  $S$ 
9 if  $m = n + 1$  then
   // Construct  $SC(n, n+1, c)$  via one-output expansion
10  Compute  $SC(n, m-1, c-1)$  by calling GenSC( $n, m-1, c-1$ )
11  Compute  $SC(n, m, c-1)$  by calling GenSC( $n, m, c-1$ )
12  Set  $S$  to GenSCByOutput( $n, m, c$ )
13  return  $S$ 
14 if  $m = n$  then
   // Construct  $SC(n, n, c)$  from  $SC(n-1, n, \cdot)$  via one-input
   // expansion
15  for each integer  $c'$  from  $m$  to  $c-1$  do
16    Compute  $SC(n-1, m, c')$  by calling GenSC( $n-1, m, c'$ )
17  Set  $S$  to GenSCByInput( $n, m, c$ )
18  return  $S$ 

```

5 Computational Results

We implemented GenSC and its auxiliary procedures from Section 4 in C++. The experiments were run on a machine with an Intel(R) Core(TM) i7-12700F @ 2.10 GHz and 48.0 GB of RAM.

5.1 Computing $SC(n, m, c)$

For each computed triple (n, m, c) , the implementation outputs the full set $SC(n, m, c)$ and records the number of graphs and the elapsed time. Graphs belonging to the same isomorphism class are counted only once, with isomorphisms required to preserve the input, internal, and output node sets. Table 4 summarizes the results. The entries with zero graphs correspond to parameter sets for which no superconcentrator of the prescribed size exists. In particular, the complete computation summarized in Table 4 determines the exact values of $s(t, t)$ for $t \leq 6$. Since $SC(6, 7, 17) = \emptyset$, we obtain $s(6, 7) \geq 18$, and Lemma 4 gives $s(7, 7) \geq 20$. Applying Lemmas 4 and 5 once more gives $s(8, 8) \geq 23$.

Table 4. Computational results of the recursive superconcentrator search. The vertical blocks group parameter patterns corresponding to $SC(n, n, c)$ to $SC(n, n + 1, c + 1)$ and $SC(n, n + 1, c + 1)$ to $SC(n + 1, n + 1, c + 3)$.

(n, m, c)	# of graphs	Time	(n, m, c)	# of graphs	Time	(n, m, c)	# of graphs	Time
(2, 2, 2)	2	0.01 s	(2, 2, 3)	19	0.01 s	(2, 2, 4)	220	0.04 s
(2, 3, 3)	8	0.03 s	(2, 3, 4)	150	0.05 s	(2, 3, 5)	2946	0.18 s
(3, 3, 5)	33	0.03 s	(3, 3, 6)	3744	0.91 s	(3, 3, 7)	225 604	56.81 s
(3, 4, 6)	44	0.05 s	(3, 4, 7)	16 383	1.36 s	(3, 4, 8)	1 921 403	125.50 s
(4, 4, 8)	8	0.13 s	(4, 4, 9)	49 947	75.24 s	(4, 4, 10)	26 494 537	2.24 h
(4, 5, 9)	0	0.04 s	(4, 5, 10)	15 614	39.41 s	(4, 5, 11)	42 923 185	9.15 h
			(5, 5, 12)	330	139.03 s	(5, 5, 13)	30 970 528	84.83 h
			(5, 6, 13)	0	0.49 s	(5, 6, 14)	1 044 957	19.70 h
						(6, 6, 16)	2147	3.58 h
						(6, 7, 17)	0	8.16 s

5.2 Partial Search for Upper Bounds

As shown in Section 5.1, the complete computation gives tight results for $t \leq 6$. For larger values of t , the inequality $s(t, t) \geq s(t - 1, t - 1) + 3$ can be used to propagate lower bounds. However, computing the full family $SC(t, t, c)$ to determine $s(t, t)$ becomes computationally infeasible for $t \geq 7$. Instead, this section uses partial search, whose goal is to obtain explicit graphs for upper bounds rather than to compute the full family $SC(t, t, c)$. It finds a partial

collection of graphs and extends them recursively until at least one graph in $SC(n, m, c)$ is obtained for selected triples (n, m, c) . Whenever such a graph is found, it immediately implies the upper bound

$$s(n, m) \leq c.$$

The implementation is based on $\text{GenSC}(n, m, c)$, but retains only a selected subset of candidates instead of constructing the entire output set.

After approximately one day of computation, the implementation found elements in $SC(7, 7, 21)$ and $SC(8, 8, 26)$, which implies $s(7, 7) \leq 21$ and $s(8, 8) \leq 26$. We also ran the partial search for approximately one month in an attempt to find an element of $SC(8, 8, 25)$. During this search, the implementation found 66,217 elements of $SC(7, 7, 22)$ and 71 elements of $SC(7, 8, 23)$. However, no element of $SC(8, 8, 25)$ was found.

6 Concrete MDS Matrices

In Section 5, we identify superconcentrator graphs for each t . We now instantiate these graphs as concrete MDS matrices by assigning coefficients to the edges of each non-input node. This section describes how the coefficients are chosen over large prime fields and for MDS matrices with k -bit words. The resulting matrices and constraints, together with the selected matrices L used for MDS matrices with k -bit words, are listed in Supplementary Material A.

6.1 MDS Matrices over Large Prime Fields

For $t = 3$, we use the efficient MDS matrix and addition constraints of [5] without modification. For each $t \geq 4$, we first choose a graph at random from $SC(t, t, c)$, where c is the smallest value found in our search. We then assign edge coefficients at most 3 and select an assignment that minimizes the largest entry of the resulting matrix. Finally, we verify the MDS property by computing the determinant of every square submatrix.

For the matrices obtained by this procedure, every square submatrix has nonzero determinant over the integers, and the largest prime divisor among these determinants is less than 2^{19} . Therefore, the matrices satisfy the MDS property over every prime field $\mathbb{F}_p$ with $p \geq 2^{19}$.

6.2 MDS Matrices with k -bit Words

For a fixed superconcentrator graph, each non-input node can be treated as a constraint on k -bit words

$$w_c = C_a w_a \oplus C_b w_b,$$

where C_a and C_b are $k \times k$ matrices over $\mathbb{F}_2$. If the two coefficient maps are implemented separately, this constraint can be evaluated using

$$k + \#\text{XOR}(C_a) + \#\text{XOR}(C_b)$$

XOR gates, where the term k accounts for the word-wise XOR of two k -bit words. Here $\#XOR(C)$ denotes the XOR count of the linear circuit used to implement the map $z \mapsto Cz$.

Following the general framework of [13], we mainly choose the coefficient blocks as powers of a sparse invertible $k \times k$ binary matrix L . The matrix L contains the cyclic-shift part

$$L[i+1, i] = 1 \quad \text{for } 0 \leq i < k-1, \quad L[0, k-1] = 1,$$

and one or two additional ones are randomly placed in the remaining entries, keeping only invertible choices. Hence, in our search, each coefficient block is of the form L^e for such a sparse matrix L and a small integer e . For such an L , if the rows of L have Hamming weights $r_1, \dots, r_k$, then

$$\#XOR(L) = \sum_{i=1}^k \max\{r_i - 1, 0\}.$$

In our construction, the cyclic-shift part contributes one nonzero entry in each row, so $\#XOR(L)$ is the number of additional ones placed outside this cyclic-shift part.

For a coefficient block L^e with $e > 0$, we may implement L^e by applying the implementation of L exactly e times. If $e < 0$, then $L^e = (L^{-1})^{|e|}$, and L^{-1} can be implemented with the same number of XORs as L by reversing the sequence of XOR updates. Hence this implementation gives the upper bound

$$\#XOR(L^e) \leq |e| \#XOR(L).$$

Thus a graph with N non-input nodes uses N word-wise XORs, contributing Nk bit-level XORs, plus the XORs required by the selected powers of L .

Compared to Tian et al. [24], our graphs have the same word-level XOR counts for $t = 5, 6$ and require one fewer word-level XOR for $t = 7$. We conduct our search for these values of t . For $t = 7$, the saving of one word-level XOR corresponds to k bit-level XORs. Moreover, we further reduce the coefficient-block cost through a heuristic search for optimal exponent patterns and sparse matrices L , coupled with bit-level post-optimizations that relax the cyclic-shift requirement of L and exploit cases where $\#XOR(L^e) < |e| \#XOR(L)$. Combining these effects, we obtain state-of-the-art XOR counts for most parameter choices. The exceptions are $(t, k) = (6, 4)$ and $(7, 4)$, for which our search did not find MDS matrices. The final XOR counts are summarized in Table 3. The detailed heuristic method and bit-level post-optimization used to reduce XOR counts are described in Supplementary Material A.2.

7 Conclusion

We studied addition-efficient MDS matrices from a graph-theoretic perspective. We represented fan-in-two addition constraints as a computation graph and used

the established result that any such graph implementing an MDS matrix must be a superconcentrator [4]. This connection enables a recursive search for lower bounds on the size of the superconcentrator, yielding tight bounds for $t \leq 6$ and improved bounds and explicit constructions for $t = 7, 8$ over large prime fields. The same graph templates also lead to XOR-efficient MDS matrices with k -bit words, improving several previously reported XOR counts.

We believe that the addition-efficient MDS matrices found in this work are particularly useful for arithmetization-oriented permutations. For instance, Poseidon2 [10] employs non-MDS linear layers for $t \geq 5$ to reduce implementation costs. When $t = 8$, such a linear layer requires only 24 additions, which is significantly more efficient than the $8 \times (8 - 1) = 56$ additions needed for a naive matrix-vector multiplication with an 8×8 MDS matrix. In contrast, our 8×8 MDS matrix requires only 26 additions, introducing an overhead of just two additions per round. Consequently, full MDS diffusion can be achieved with negligible overhead, demonstrating that security and linear-layer efficiency can be simultaneously realized. Furthermore, MDS matrices are also known as *hyperinvertible matrices* and are used in MPC to generate and verify batches of random sharings [3, 19]. Our addition-efficient constructions can reduce the local arithmetic cost of applying these matrices to shares.

On the other hand, there are gaps between the lower and the upper bounds for $t = 7, 8$, and computational search seems to be infeasible when $t \geq 9$. Closing these gaps and extending the recursive search to larger states remain open for future work.

Acknowledgments

Parts of the implementation described in Section 5 and 6 were developed with the assistance of Codex, including implementation-level suggestions. The authors manually reviewed, modified, and tested all AI-assisted code. The final implementation was validated by independently running the experimental pipeline and confirming that the reported experimental results were reproducible.

Parts of the manuscript were edited with the assistance of generative AI tools (ChatGPT, Gemini, and Codex) for language polishing, including phrasing, grammar, typo correction, and clarity. All AI-assisted suggestions were reviewed and revised by the authors, who take full responsibility for the correctness and integrity of the manuscript.

References

1. Alon, N., Milman, V.D.: Eigenvalues, expanders and superconcentrators. In: 25th Annual Symposium on Foundations of Computer Science, 1984. pp. 320–322. IEEE (1984)
2. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szeponiec, A.: Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols. ToSC 2020 2020(3) (Sep 2020). <https://doi.org/10.13154/tosc.v2020.i3.1-45>

3. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-Secure MPC with Linear Communication Complexity. In: Theory of Cryptography – TCC 2008. Lecture Notes in Computer Science, vol. 4948, pp. 213–230. Springer (2008)
4. Drucker, A., Li, Y.: On the Minimum Depth of Circuits with Linear Number of Wires Encoding Good Codes. In: International Computing and Combinatorics Conference. pp. 392–403. Springer (2023)
5. Duval, S., Leurent, G.: MDS matrices with lightweight circuits. ToSC 2018 2018(2), 48–78 (2018)
6. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive (2019)
7. Grassi, L., Hao, Y., Rechberger, C., Schafneggger, M., Walch, R., Wang, Q.: Horst meets Fluid-SPN: Griffin for zero-knowledge applications. In: CRYPTO 2023. pp. 573–606. Springer (2023)
8. Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schafneggger, M., Walch, R.: Reinforced concrete: a fast hash function for verifiable computation. In: CCS 2022. pp. 1323–1335 (2022)
9. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schafneggger, M.: Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In: USENIX 2021. pp. 519–535. USENIX Association (Aug 2021)
10. Grassi, L., Khovratovich, D., Schafneggger, M.: Poseidon2: A faster version of the poseidon hash function. In: AFRICACRYPT 2023. pp. 177–203. Springer (2023)
11. Gupta, K.C., Pandey, S.K., Samanta, S.: Construction of recursive MDS matrices using DLS matrices. In: International Conference on Cryptology in Africa. pp. 3–27. Springer (2022)
12. Ha, J., Hwang, S., Lee, J., Park, S., Son, M.: Polocolo: A ZK-Friendly Hash Function Based on S-Boxes Using Power Residues. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 303–332. Springer (2025)
13. He, Y., Cui, T., Ling, Q., Han, X.: New General MDS Matrix Construction Method Towards Low Area. IACR Transactions on Symmetric Cryptology 2025(3), 868–890 (2025). <https://doi.org/10.46586/tosc.v2025.i3.868-890>
14. Junod, P., Vaudenay, S.: Perfect diffusion primitives for block ciphers. In: SAC 2004. pp. 84–99. Springer (2005)
15. Kesarwani, A., Sarkar, S., Venkateswarlu, A.: Exhaustive Search for Various Types of MDS Matrices. IACR Transactions on Symmetric Cryptology 2019(3), 231–256 (2019)
16. Lev, G.: Size Bounds and Parallel Algorithms for Networks. Doctoral thesis, University of Edinburgh (1980), report CST-8-80, Department of Computer Science
17. Lev, G., Valiant, L.G.: Size bounds for superconcentrators. Theoretical Computer Science 22(3), 233–251 (1983)
18. Li, S., Sun, S., Shi, D., Li, C., Hu, L.: Lightweight Iterative MDS Matrices: How Small Can We Go? IACR Transactions on Symmetric Cryptology 2019(4), 147–170 (2019)
19. Liu, H., Xing, C., Yang, Y., Yuan, C.: Ramp Hyper-invertible Matrices and Their Applications to MPC Protocols. In: Advances in Cryptology – ASIACRYPT 2023: 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4–8, 2023, Proceedings, Part I. p. 204–236. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-981-99-8721-4_7, https://doi.org/10.1007/978-981-99-8721-4_7

20. Merz, S.P., Rodríguez García, À.: Skipping Class: Algebraic Attacks Exploiting Weak Matrices and Operation Modes of Poseidon2(b). *Cryptology ePrint Archive*, Paper 2026/306 (2026), <https://eprint.iacr.org/2026/306>
21. Pippenger, N.: Superconcentrators. *SIAM Journal on Computing* 6(2), 298–304 (1977)
22. Roy, A., Steiner, M.J., Trevisani, S.: Arion: Arithmetization-oriented permutation and hashing from generalized triangular dynamical systems. *arXiv preprint arXiv:2303.04639* (2023)
23. Sajadieh, M., Mousavi, M.: Construction of MDS Matrices from Generalized Feistel Structures. *Designs, Codes and Cryptography* 89(7), 1433–1452 (2021)
24. Tian, Y., Feng, X., Li, G.: On the construction of ultra-light MDS matrices. *Cryptography and Communications* 18(3), 901–949 (2026)
25. Toh, D., Teo, J., Khoo, K., Sim, S.M.: Lightweight MDS Serial-Type Matrices with Minimal Fixed XOR Count. In: *AFRICACRYPT 2018*. LNCS, vol. 10831, pp. 51–71. Springer (2018)
26. Valiant, L.G.: Graph-theoretic properties in computational complexity. *Journal of Computer and System Sciences* 13(3), 278–285 (1976)
27. Wang, S., Li, Y., Tian, S., Zeng, X.: Four by Four MDS Matrices with the Fewest XOR Gates Based on Words. *Advances in Mathematics of Communications* 17(4), 845–872 (2023)
28. Wu, S., Wang, M., Wu, W.: Recursive Diffusion Layers for (Lightweight) Block Ciphers and Hash Functions. In: *SAC 2012*. LNCS, vol. 7707, pp. 355–371. Springer (2013)

Supplementary Material

A Linear Layers

A.1 Linear Layers and Constraints for MDS Matrices over Large Prime Fields

This section presents the linear layers and constraints for MDS matrices over large prime fields discussed in Section 6.1.

– **t = 3.**

$$M = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix},$$

$$\text{Constraints} = \begin{cases} w_1 = x_1 + x_2, & w_2 = w_1 + x_3, & y_1 = w_2 + x_1, \\ y_2 = w_2 + x_2, & y_3 = w_2 + x_3 \end{cases}.$$

– **t = 4.**

$$M = \begin{pmatrix} 2 & 1 & 1 & 3 \\ 1 & 2 & 3 & 1 \\ 3 & 1 & 2 & 4 \\ 1 & 3 & 4 & 2 \end{pmatrix},$$

$$\text{Constraints} = \begin{cases} w_1 = x_2 + x_3, & w_2 = x_4 + x_1, & w_3 = x_4 + w_1 \\ w_4 = w_2 + x_3, & y_1 = 2w_2 + w_3, & y_2 = w_4 + 2w_1 \\ y_3 = w_4 + y_1, & y_4 = y_2 + w_3 \end{cases}.$$

– **t = 5.**

$$M = \begin{pmatrix} 4 & 6 & 5 & 5 & 3 \\ 2 & 6 & 4 & 8 & 5 \\ 3 & 5 & 3 & 4 & 2 \\ 1 & 5 & 4 & 6 & 4 \\ 4 & 4 & 3 & 1 & 1 \end{pmatrix},$$

$$\text{Constraints} = \begin{cases} w_1 = x_4 + x_5, & w_2 = x_1 + x_2, & w_3 = x_3 + 2w_2 \\ w_4 = x_3 + w_1, & w_5 = 2x_2 + 2w_4, & y_5 = 2w_3 + w_4 \\ w_6 = 2x_4 + w_5, & w_7 = w_2 + w_6, & y_1 = w_6 + y_5 \\ y_2 = x_5 + 2w_7, & y_3 = w_3 + w_7, & y_4 = w_5 + w_7 \end{cases}.$$

– $\mathbf{t} = 6$.

$$M = \begin{pmatrix} 1 & 14 & 16 & 12 & 13 & 19 \\ 5 & 38 & 40 & 40 & 36 & 49 \\ 8 & 36 & 37 & 44 & 38 & 46 \\ 1 & 5 & 7 & 4 & 5 & 8 \\ 2 & 17 & 17 & 18 & 16 & 21 \\ 8 & 32 & 33 & 40 & 34 & 41 \end{pmatrix},$$

$$\text{Constraints} = \begin{cases} w_9 = x_3 + x_6, & w_3 = x_2 + w_9, & w_4 = x_5 + w_3 \\ w_1 = 2x_4 + 2w_4, & w_{10} = x_6 + 2w_1, & w_5 = w_3 + 2w_{10} \\ w_6 = x_1 + w_{10}, & w_7 = w_6 + 2w_9, & w_8 = 2x_4 + 2w_6 \\ w_2 = x_5 + 2w_8, & y_4 = w_4 + w_7, & y_5 = w_5 + w_8 \\ y_1 = w_5 + y_4, & y_2 = w_7 + 2y_5, & y_6 = 2w_2 + w_9 \\ y_3 = w_{10} + y_6 \end{cases}.$$

– $\mathbf{t} = 7$.

$$M = \begin{pmatrix} 6 & 92 & 21 & 24 & 20 & 80 & 26 \\ 10 & 104 & 30 & 18 & 25 & 96 & 29 \\ 26 & 256 & 82 & 38 & 62 & 240 & 70 \\ 16 & 189 & 50 & 44 & 43 & 168 & 53 \\ 15 & 157 & 47 & 29 & 37 & 146 & 43 \\ 12 & 121 & 38 & 20 & 29 & 114 & 35 \\ 18 & 187 & 56 & 41 & 43 & 170 & 52 \end{pmatrix},$$

$$\text{Constraints} = \begin{cases} w_1 = x_2 + 2x_4, & w_2 = 2x_2 + 2x_6, & w_3 = x_5 + x_7 \\ w_4 = w_3 + 2w_2, & w_5 = 3x_3 + 2w_4, & w_6 = x_1 + w_5 \\ w_7 = x_4 + 3w_6, & w_8 = w_6 + x_7, & w_9 = 2w_1 + w_8 \\ w_{10} = 2w_9 + 2w_4, & w_{11} = w_7 + w_{10}, & y_1 = 3w_{10} + w_5 \\ y_2 = 2w_{11} + w_3, & w_{12} = y_2 + 2x_3, & w_{13} = w_{12} + w_1 \\ y_3 = 2w_7 + 2w_{12}, & w_{14} = w_{13} + 2x_6, & y_4 = 3w_{10} + w_{13} \\ y_5 = w_{11} + w_{14}, & y_6 = w_{14} + 2w_8, & y_7 = y_5 + 3w_9 \end{cases}.$$

– $\mathbf{t} = 8$.

$$M = \begin{pmatrix} 12 & 4 & 3 & 5 & 3 & 8 & 6 & 3 \\ 36 & 11 & 12 & 26 & 16 & 12 & 20 & 18 \\ 116 & 46 & 69 & 109 & 77 & 24 & 68 & 71 \\ 170 & 66 & 75 & 120 & 99 & 74 & 97 & 78 \\ 378 & 158 & 195 & 320 & 235 & 124 & 221 & 202 \\ 332 & 148 & 195 & 319 & 221 & 75 & 197 & 201 \\ 644 & 284 & 372 & 623 & 424 & 150 & 384 & 387 \\ 388 & 168 & 214 & 355 & 248 & 107 & 229 & 221 \end{pmatrix},$$

$$\text{Constraints} = \begin{cases} w_1 = x_3 + x_5, & w_2 = 2x_1 + x_7, & w_3 = x_4 + x_8 \\ w_4 = 2x_6 + w_2, & w_5 = 3w_3 + 2w_2, & w_6 = x_2 + 2w_4 \\ w_7 = 3w_1 + w_5, & w_8 = 2x_2 + w_7, & w_9 = 2x_4 + w_8 \\ w_{10} = x_7 + 2w_9, & y_1 = 2w_6 + w_9, & w_{11} = 2x_5 + w_{10} \\ w_{12} = 3w_6 + 2w_{11}, & w_{13} = w_{12} + x_8, & y_2 = 2w_5 + w_{12} \\ w_{14} = 3w_{10} + w_{13}, & w_{15} = w_1 + 2w_{13}, & w_{16} = 2w_{14} + x_6 \\ y_3 = 2w_{14} + 3w_7, & y_4 = 3w_{15} + w_4, & w_{17} = 3w_{16} + w_{11} \\ y_5 = 2w_{16} + y_4, & w_{18} = w_{17} + w_{15}, & y_6 = w_{17} + 3w_8 \\ y_7 = 2w_{17} + w_5, & y_8 = y_1 + w_{18} \end{cases}.$$

A.2 Linear Layers and Constraints for MDS Matrices with k -bit Words

This section presents the linear layers and constraints for MDS matrices with k -bit words discussed in Section 6.2. We first describe the heuristic methods used to assign coefficient blocks to the superconcentrator graphs, and then describe the bit-level post-processing used to reduce the final XOR count.

The first method is a direct randomized search. We choose a candidate graph among the superconcentrator graphs obtained in Section 5, choose a sparse invertible linear map L , and randomly assign exponents to the coefficient blocks. Thus the coefficients are represented as powers L^e . For the smallest word sizes used as starting points, namely $(t, k) = (5, 4)$, $(6, 8)$, and $(7, 8)$, we also used the corresponding fixed sparse linear maps from [13]. Among the MDS candidates, we retain candidates with low XOR count.

The second method lifts a construction from a smaller word size. Given a construction for (t, k') with $k' < k$, we reuse its graph and exponent pattern, and search over admissible sparse choices of L for the target word size k . Among the choices of L that yield an MDS matrix, we select one minimizing the estimated XOR count.

We then apply an exponent-normalization post-processing step. For each node v in the computation graph, we introduce an integer shift r_v and replace each edge coefficient L^e on an edge $u \rightarrow v$ by $L^{e+r_v-r_u}$. This corresponds to rescaling the variables by powers of L , and changes the resulting linear layer only by invertible block-diagonal scalings on the input and output variables. Hence the MDS property is preserved. We choose the shifts by solving a Mixed-Integer Linear Program (MILP) that minimizes the sum of the absolute values of the resulting exponents.

Finally, we perform a bit-level XOR post-optimization. The previous steps estimate the cost of L^e by the repeated-application bound $|e| \cdot \#\text{XOR}(L)$. This bound is not always tight. Therefore, for the selected candidates, we explicitly synthesize linear circuits for the coefficient maps L^e and use the best circuits found by the synthesis procedure.

In addition, we compute each selected constraint $w_c = C_a w_a \oplus C_b w_b$ as a whole, so that common bit-level XOR operations are performed only once and

their results are reused. We use this implementation when it requires fewer than $k + \#XOR(C_a) + \#XOR(C_b)$ XOR gates.

Moreover, we attempt to relax the cyclic-shift requirement for L while keeping the graph and exponent pattern fixed. For each candidate sparse invertible matrix L , we recompute the coefficient blocks and verify the MDS property of the resulting linear layer. Changing L alters the bit-level expressions of its powers and can enable more efficient sharing of intermediate values, even when $\#XOR(L)$ remains unchanged. We therefore select L according to the cost of the resulting bit-level circuits. As an example, we found an improvement for $(t, k) = (6, 8)$. For the selected L , the constraint $w_{10} = Lw_1 \oplus L^{-2}w_9$ is implemented with 13 XOR gates, compared with $2 + 4 + 8 = 14$ for separate implementations of the coefficient maps and the final word-wise XOR. The complete implementation, which also computes Lw_6 and $L^{-2}w_6$ jointly, uses 144 XOR gates.

Through our heuristic search and exponent normalization, we initially achieve state-of-the-art XOR counts for all evaluated (t, k) pairs, with the exception of $(6, 4)$, $(7, 4)$ and $(6, 8)$. While no MDS matrices were found for $(6, 4)$ and $(7, 4)$, our search successfully matches the current best results of [24] for $(6, 8)$. Furthermore, our post-optimization reduces the XOR counts for a wide range of parameters, specifically for $(t, k) = (5, 4), \{(6, k'), (7, k')\}_{k' \in \{8, 16, 32, 64\}}$. Consequently, our complete methodology establishes new state-of-the-art XOR counts for every (t, k) pair where MDS matrices were found based on our approach.

We now present the resulting linear layers and constraints. For $t > 5$, we omit the symbolic matrices due to their large size. The given constraints directly determine the corresponding MDS matrices. We write a binary matrix by listing the nonzero column positions in each row. More precisely, $L = [S_1, \dots, S_k]$ denotes the matrix whose i -th row has ones in the columns listed in S_i and zeros elsewhere, with row and column indices starting from 1. For example,

$$L = [[4], [1, 4], [2], [3]]$$

denotes the binary matrix

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

– **t = 5, k = 4.**

$$L = [[4], [1, 4], [2], [3]],$$

$$M = \begin{pmatrix} L & I \oplus L^{-4} \oplus L^{-2} & L^{-1} \oplus L & L^{-4} \oplus L^{-2} & I \oplus L^{-4} \oplus L^{-2} \oplus L^{-1} \\ I \oplus L L^{-3} \oplus L^{-4} \oplus L^{-2} & L^{-1} \oplus L & I \oplus L^{-3} \oplus L^{-4} \oplus L^{-2} & L^{-3} \oplus L^{-4} \oplus L^{-2} \oplus L^{-1} \\ I & I \oplus L^{-5} & L^{-2} \oplus L^{-1} & L^{-5} \oplus L^{-1} & L^{-5} \oplus L^{-2} \\ L & L^{-2} & L & I \oplus L^{-2} & L^{-2} \\ I & I \oplus L^{-5} \oplus L^{-3} & I \oplus L^{-2} \oplus L^{-1} & L^{-5} \oplus L^{-3} \oplus L^{-1} & I \oplus L^{-5} \oplus L^{-3} \oplus L^{-2} \end{pmatrix}.$$

$$\text{Constraints} = \begin{cases} w_1 = x_2 \oplus x_5, & w_2 = x_4 \oplus w_1 \\ w_3 = x_3 \oplus L^{-3}w_2, & w_4 = x_1 \oplus w_3 \\ w_5 = x_5 \oplus w_3, & y_4 = x_4 \oplus Lw_4 \\ w_6 = L^{-1}w_5 \oplus y_4, & w_7 = x_3 \oplus w_6 \\ y_1 = w_2 \oplus w_6, & y_2 = w_4 \oplus w_7 \\ y_5 = w_1 \oplus L^{-1}w_7, & y_3 = w_5 \oplus y_5 \end{cases}.$$

– **t = 5, k = 8.**

$$L = [[6, 8], [1], \dots, [7]],$$

$$M = \begin{pmatrix} L^{-2} & L^{-2} & L^{-3} \oplus I & L^{-2} \oplus I & L^{-2} \oplus I \\ L^{-4} \oplus I & L^{-4} \oplus L^{-1} \oplus L & L^{-5} \oplus L^{-2} & L^{-4} \oplus L & L^{-4} \oplus L^{-2} \oplus L^{-1} \oplus L \\ L^{-4} \oplus L^{-2} \oplus I & L^{-4} \oplus L^{-2} \oplus L^{-1} & L^{-5} \oplus L^{-3} \oplus L^{-2} \oplus I & L^{-4} \oplus L^{-2} & L^{-4} \oplus L^{-1} \\ I & L^{-1} \oplus I & L^{-1} & I & L^{-1} \oplus I \\ L^{-4} & L^{-4} \oplus I & L^{-5} \oplus L^{-2} \oplus L^{-1} & L^{-4} \oplus I & L^{-4} \oplus L^{-2} \oplus I \end{pmatrix},$$

$$\text{Constraints} = \begin{cases} w_1 = x_2 \oplus x_5, & w_2 = x_4 \oplus w_1 \\ w_3 = L^{-1}x_3 \oplus w_2, & w_4 = x_1 \oplus w_3 \\ w_5 = x_3 \oplus L^{-2}w_4, & y_4 = L^{-1}w_1 \oplus w_4 \\ w_6 = x_5 \oplus w_5, & y_1 = x_4 \oplus w_6 \\ y_5 = w_3 \oplus L^{-2}w_6, & w_7 = y_4 \oplus y_5 \\ y_2 = Lw_2 \oplus w_7, & y_3 = w_5 \oplus w_7 \end{cases}.$$

– **t = 5, k = 16.**

$$L = [[1, 16], [1], \dots, [15]],$$

$$M = \begin{pmatrix} L & L^{-1} \oplus I \oplus L \oplus L^2 & I \oplus L & L \oplus L^2 & L^{-1} \oplus L \oplus L^2 \\ L^2 & L \oplus L^2 \oplus L^3 & I \oplus L^2 & I \oplus L^2 \oplus L^3 & L^2 \oplus L^3 \\ L^{-1} \oplus I \oplus L^2 & I \oplus L^2 \oplus L^3 & L^2 & L^{-1} \oplus I \oplus L \oplus L^2 \oplus L^3 & I \oplus L \oplus L^2 \oplus L^3 \\ I & L^{-1} \oplus I & I & I & L^{-1} \oplus I \\ L^{-1} \oplus L^2 & L^{-1} \oplus L \oplus L^2 \oplus L^3 & I \oplus L^2 & L^{-1} \oplus L^2 \oplus L^3 & L^{-1} \oplus L^2 \oplus L^3 \end{pmatrix},$$

$$\text{Constraints} = \begin{cases} w_1 = x_2 \oplus x_5, & w_2 = x_3 \oplus L^{-1}w_1 \\ w_3 = x_4 \oplus w_1, & w_4 = x_1 \oplus w_3 \\ y_4 = w_2 \oplus w_4, & w_5 = Lw_3 \oplus y_4 \\ w_6 = x_5 \oplus Lw_5, & w_7 = x_3 \oplus Lw_6 \\ y_1 = w_2 \oplus w_6, & y_2 = x_4 \oplus w_7 \\ y_5 = L^{-1}w_4 \oplus w_7, & y_3 = w_5 \oplus y_5 \end{cases}.$$

– **t = 5, k = 32.**

$$L = [[11, 32], [1], \dots, [31]],$$

$$M = \begin{pmatrix} L & L^{-1} \oplus I \oplus L \oplus L^2 & I \oplus L & L \oplus L^2 & L^{-1} \oplus L \oplus L^2 \\ L^2 & L \oplus L^2 \oplus L^3 & I \oplus L^2 & I \oplus L^2 \oplus L^3 & L^2 \oplus L^3 \\ L^{-1} \oplus I \oplus L^2 & I \oplus L^2 \oplus L^3 & L^2 & L^{-1} \oplus I \oplus L \oplus L^2 \oplus L^3 & I \oplus L \oplus L^2 \oplus L^3 \\ I & L^{-1} \oplus I & I & I & L^{-1} \oplus I \\ L^{-1} \oplus L^2 & L^{-1} \oplus L \oplus L^2 \oplus L^3 & I \oplus L^2 & L^{-1} \oplus L^2 \oplus L^3 & L^{-1} \oplus L^2 \oplus L^3 \end{pmatrix},$$

$$\text{Constraints} = \begin{cases} w_1 = x_2 \oplus x_5, & w_2 = x_3 \oplus L^{-1}w_1 \\ w_3 = x_4 \oplus w_1, & w_4 = x_1 \oplus w_3 \\ y_4 = w_2 \oplus w_4, & w_5 = Lw_3 \oplus y_4 \\ w_6 = x_5 \oplus Lw_5, & w_7 = x_3 \oplus Lw_6 \\ y_1 = w_2 \oplus w_6, & y_2 = x_4 \oplus w_7 \\ y_5 = L^{-1}w_4 \oplus w_7, & y_3 = w_5 \oplus y_5 \end{cases}.$$

– **t = 5, k = 64.**

$$L = [[21, 64], [1], \dots, [63]],$$

$$M = \begin{pmatrix} L & L^{-1} \oplus I \oplus L \oplus L^2 & I \oplus L & L \oplus L^2 & L^{-1} \oplus L \oplus L^2 \\ L^2 & L \oplus L^2 \oplus L^3 & I \oplus L^2 & I \oplus L^2 \oplus L^3 & L^2 \oplus L^3 \\ L^{-1} \oplus I \oplus L^2 & I \oplus L^2 \oplus L^3 & L^2 & L^{-1} \oplus I \oplus L \oplus L^2 \oplus L^3 & I \oplus L \oplus L^2 \oplus L^3 \\ I & L^{-1} \oplus I & I & I & L^{-1} \oplus I \\ L^{-1} \oplus L^2 & L^{-1} \oplus L \oplus L^2 \oplus L^3 & I \oplus L^2 & L^{-1} \oplus L^2 \oplus L^3 & L^{-1} \oplus L^2 \oplus L^3 \end{pmatrix},$$

$$\text{Constraints} = \begin{cases} w_1 = x_2 \oplus x_5, & w_2 = x_3 \oplus L^{-1}w_1 \\ w_3 = x_4 \oplus w_1, & w_4 = x_1 \oplus w_3 \\ y_4 = w_2 \oplus w_4, & w_5 = Lw_3 \oplus y_4 \\ w_6 = x_5 \oplus Lw_5, & w_7 = x_3 \oplus Lw_6 \\ y_1 = w_2 \oplus w_6, & y_2 = x_4 \oplus w_7 \\ y_5 = L^{-1}w_4 \oplus w_7, & y_3 = w_5 \oplus y_5 \end{cases}.$$

– **t = 6, k = 8.**

$$L = [[3], [6], [5], [4, 5], [4, 7], [1], [8], [2]],$$

$$\text{Constraints} = \begin{cases} w_1 = x_1 \oplus x_4, & w_2 = x_2 \oplus x_6 \\ w_3 = x_5 \oplus w_2, & w_4 = w_1 \oplus Lw_3 \\ w_5 = x_6 \oplus w_4, & w_6 = x_3 \oplus w_5 \\ w_7 = x_4 \oplus L^{-2}w_6, & y_5 = w_3 \oplus Lw_6 \\ w_8 = x_5 \oplus L^{-1}w_7, & w_9 = x_3 \oplus L^{-1}w_8 \\ y_1 = w_4 \oplus w_8, & w_{10} = Lw_1 \oplus L^{-2}w_9 \\ y_2 = w_9 \oplus y_5, & y_3 = w_7 \oplus w_{10} \\ y_6 = w_2 \oplus w_{10}, & y_4 = w_5 \oplus y_6 \end{cases}.$$

– **t = 6, k = 16.**

$$L = [[1, 16], [1], \dots, [15]],$$

$$\text{Constraints} = \begin{cases} w_1 = x_1 \oplus x_4, & w_2 = x_2 \oplus x_6 \\ w_3 = x_5 \oplus w_2, & w_4 = w_1 \oplus Lw_3 \\ w_5 = x_6 \oplus w_4, & w_6 = x_3 \oplus w_5 \\ w_7 = x_4 \oplus L^{-2}w_6, & y_5 = w_3 \oplus Lw_6 \\ w_8 = x_5 \oplus L^{-1}w_7, & w_9 = x_3 \oplus L^{-1}w_8 \\ y_1 = w_4 \oplus w_8, & w_{10} = Lw_1 \oplus L^{-2}w_9 \\ y_2 = w_9 \oplus y_5, & y_3 = w_7 \oplus w_{10} \\ y_6 = w_2 \oplus w_{10}, & y_4 = w_5 \oplus y_6 \end{cases}.$$

– **t = 6, k = 32.**

$$L = [[21, 32], [1], \dots, [31]],$$

$$\text{Constraints} = \begin{cases} w_1 = x_1 \oplus x_4, & w_2 = x_2 \oplus x_6 \\ w_3 = x_5 \oplus w_2, & w_4 = w_1 \oplus Lw_3 \\ w_5 = x_6 \oplus w_4, & w_6 = x_3 \oplus w_5 \\ w_7 = x_4 \oplus L^{-2}w_6, & y_5 = w_3 \oplus Lw_6 \\ w_8 = x_5 \oplus L^{-1}w_7, & w_9 = x_3 \oplus L^{-1}w_8 \\ y_1 = w_4 \oplus w_8, & w_{10} = Lw_1 \oplus L^{-2}w_9 \\ y_2 = w_9 \oplus y_5, & y_3 = w_7 \oplus w_{10} \\ y_6 = w_2 \oplus w_{10}, & y_4 = w_5 \oplus y_6 \end{cases}.$$

– **t = 6, k = 64.**

$$L = [[21, 64], [1], \dots, [63]],$$

$$\text{Constraints} = \begin{cases} w_1 = x_1 \oplus x_4, & w_2 = x_2 \oplus x_6 \\ w_3 = x_5 \oplus w_2, & w_4 = w_1 \oplus Lw_3 \\ w_5 = x_6 \oplus w_4, & w_6 = x_3 \oplus w_5 \\ w_7 = x_4 \oplus L^{-2}w_6, & y_5 = w_3 \oplus Lw_6 \\ w_8 = x_5 \oplus L^{-1}w_7, & w_9 = x_3 \oplus L^{-1}w_8 \\ y_1 = w_4 \oplus w_8, & w_{10} = Lw_1 \oplus L^{-2}w_9 \\ y_2 = w_9 \oplus y_5, & y_3 = w_7 \oplus w_{10} \\ y_6 = w_2 \oplus w_{10}, & y_4 = w_5 \oplus y_6 \end{cases}.$$

– **t = 7, k = 8.**

$$L = [[8], [1, 2], [2, 8], [3], \dots, [7]],$$

$$\text{Constraints} = \left\{ \begin{array}{ll} w_1 = x_2 \oplus x_4, & w_2 = x_6 \oplus x_7 \\ w_3 = L^{-4}w_2 \oplus x_5, & w_4 = x_2 \oplus w_3 \\ w_5 = x_3 \oplus L^{-1}w_4, & w_6 = x_1 \oplus L^{-1}w_5 \\ w_7 = x_4 \oplus L^{-2}w_6, & w_8 = w_6 \oplus x_7 \\ w_9 = w_1 \oplus w_8, & w_{10} = L^5w_9 \oplus Lw_4 \\ w_{11} = w_7 \oplus w_{10}, & y_1 = w_{10} \oplus w_5 \\ y_2 = w_{11} \oplus w_2, & w_{12} = y_2 \oplus x_3 \\ w_{13} = w_{12} \oplus w_1, & y_3 = L^{-4}w_7 \oplus w_{12} \\ w_{14} = L^{-1}w_{13} \oplus x_5, & y_4 = Lw_{10} \oplus w_{13} \\ y_5 = L^2w_{11} \oplus L^{-1}w_{14}, & y_6 = w_{14} \oplus w_8 \\ y_7 = y_5 \oplus w_9 \end{array} \right. .$$

– **t = 7, k = 16.**

$$L = [[13, 16], [1], \dots, [10], [9, 11], [12], \dots, [15]],$$

$$\text{Constraints} = \left\{ \begin{array}{ll} w_1 = x_2 \oplus x_4, & w_2 = x_2 \oplus L^{-1}x_6 \\ w_3 = x_5 \oplus x_7, & w_4 = L^{-1}w_3 \oplus w_2 \\ w_5 = x_3 \oplus L^{-1}w_4, & w_6 = x_1 \oplus w_5 \\ w_7 = x_4 \oplus L^{-2}w_6, & w_8 = w_6 \oplus x_7 \\ w_9 = w_1 \oplus w_8, & w_{10} = L^2w_9 \oplus w_4 \\ w_{11} = L^{-2}w_7 \oplus w_{10}, & y_1 = w_{10} \oplus w_5 \\ y_2 = w_{11} \oplus w_3, & w_{12} = y_2 \oplus x_3 \\ w_{13} = w_{12} \oplus L^{-2}w_1, & y_3 = w_7 \oplus Lw_{12} \\ w_{14} = w_{13} \oplus Lx_6, & y_4 = w_{10} \oplus L^{-3}w_{13} \\ y_5 = w_{11} \oplus Lw_{14}, & y_6 = w_{14} \oplus w_8 \\ y_7 = y_5 \oplus w_9 \end{array} \right. .$$

– **t = 7, k = 32.**

$$L = [[21, 32], [1], \dots, [31]],$$

$$\text{Constraints} = \left\{ \begin{array}{l} w_1 = x_2 \oplus x_4, \quad w_2 = x_2 \oplus L^{-1}x_6 \\ w_3 = x_5 \oplus x_7, \quad w_4 = L^{-1}w_3 \oplus w_2 \\ w_5 = x_3 \oplus L^{-1}w_4, \quad w_6 = x_1 \oplus w_5 \\ w_7 = x_4 \oplus L^{-2}w_6, \quad w_8 = w_6 \oplus x_7 \\ w_9 = w_1 \oplus w_8, \quad w_{10} = L^2w_9 \oplus w_4 \\ w_{11} = L^{-2}w_7 \oplus w_{10}, \quad y_1 = w_{10} \oplus w_5 \\ y_2 = w_{11} \oplus w_3, \quad w_{12} = y_2 \oplus x_3 \\ w_{13} = w_{12} \oplus L^{-2}w_1, \quad y_3 = w_7 \oplus Lw_{12} \\ w_{14} = w_{13} \oplus Lx_6, \quad y_4 = w_{10} \oplus L^{-3}w_{13} \\ y_5 = w_{11} \oplus Lw_{14}, \quad y_6 = w_{14} \oplus w_8 \\ y_7 = y_5 \oplus w_9 \end{array} \right. .$$

– **t = 7, k = 64.**

$$L = [[21, 64], [1], \dots, [63]],$$

$$\text{Constraints} = \left\{ \begin{array}{l} w_1 = x_2 \oplus x_4, \quad w_2 = x_2 \oplus L^{-1}x_6 \\ w_3 = x_5 \oplus x_7, \quad w_4 = L^{-1}w_3 \oplus w_2 \\ w_5 = x_3 \oplus L^{-1}w_4, \quad w_6 = x_1 \oplus w_5 \\ w_7 = x_4 \oplus L^{-2}w_6, \quad w_8 = w_6 \oplus x_7 \\ w_9 = w_1 \oplus w_8, \quad w_{10} = L^2w_9 \oplus w_4 \\ w_{11} = L^{-2}w_7 \oplus w_{10}, \quad y_1 = w_{10} \oplus w_5 \\ y_2 = w_{11} \oplus w_3, \quad w_{12} = y_2 \oplus x_3 \\ w_{13} = w_{12} \oplus L^{-2}w_1, \quad y_3 = w_7 \oplus Lw_{12} \\ w_{14} = w_{13} \oplus Lx_6, \quad y_4 = w_{10} \oplus L^{-3}w_{13} \\ y_5 = w_{11} \oplus Lw_{14}, \quad y_6 = w_{14} \oplus w_8 \\ y_7 = y_5 \oplus w_9 \end{array} \right. .$$