More results on Shortest Linear Programs

Subhadeep Banik[1], Yuki Funabiki[2] and Takanori Isobe[3,4]

[1] LASEC, École Polytechnique Fédérale de Lausanne, Switzerland,
subhadeep.banik@epfl.ch
[2] Sony Corporation, Japan, yuki.funabiki@sony.com
[3] National Institute of Information and Communications Technology, Japan
[4] University of Hyogo, Japan, takanori.isobe@ai.u-hyogo.ac.jp

Abstract. At the FSE conference of ToSC 2018, Kranz et al. presented their results on shortest linear programs for the linear layers of several well known block ciphers in literature. Shortest linear programs are essentially the minimum number of 2-input xor gates required to completely describe a linear system of equations. In the above paper the authors showed that the commonly used metrics like d-xor/s-xor count that are used to judge the "lightweightedness" do not represent the minimum number of xor gates required to describe a given MDS matrix. In fact they used heuristic based algorithms of Boyar/Peralta and Paar to find implementations of MDS matrices with even fewer xor gates than was previously known. They proved that the AES mixcolumn matrix can be implemented with as little as 97 xor gates. In this paper we show that the values reported in the above paper are not optimal. By suitably including random bits in the instances of the above algorithms we can achieve implementations of almost all matrices with lesser number of gates than were reported in the above paper. As a result we report an implementation of the AES mixcolumn matrix that uses only 95 xor gates.
In the second part of the paper, we observe that most standard cell libraries contain both 2 and 3-input xor gates, with the silicon area of the 3-input xor gate being smaller than the sum of the areas of two 2-input xor gates. Hence when linear circuits are synthesized by logic compilers (with specific instructions to optimize for area), most of them would return a solution circuit containing both 2 and 3-input xor gates. Thus from a practical point of view, reducing circuit size in presence of these gates is no longer equivalent to solving the shortest linear program. In this paper we show that by adopting a graph based heuristic it is possible to convert a circuit constructed with 2-input xor gates to another functionally equivalent circuit that utilizes both 2 and 3-input xor gates and occupies less hardware area. As a result we obtain more lightweight implementations of all the matrices listed in the ToSC paper.

1 Introduction

Shortest linear programs are essentially the minimum number of 2-input xor gates required to completely describe a linear system of equations. The advantages to having a short linear program solution of a given matrix over $GF(2)$

are obvious. Since such linear matrices are used in the diffusion layer of block ciphers, they lead to more lightweight implementations of the block cipher circuit in hardware.

There has been extensive study on construction of lightweight diffusion layers with Maximum diffusion property [GR15, SKOP15, SS16, SS17, LS16, LW16, BKL16, JPST17] that guarantees optimal diffusion of differentials across the linear layer. MDS matrices ensure that the sum of the number of active cells before and after the linear layer is at least equal to one more than the number of rows/columns of the matrix. The advent of recursive constructions for MDS matrices [AF14,GPV17], made block cipher and hash function circuits more compact as was evidenced in the designs of LED [GPPR11] and Photon [GPP11]. Recent years have seen MDS matrices being constructed using several underlying structures like Toeplitz matrices, Hadamard matrices Cauchy matrices, Vandermonde matrices etc. The end goal for all these approaches is to minimize the xor gate count of the matrices. However since the problem of finding the minimal xor gate count of any linear system of equations is known to be NP-complete [BMP08], the authors resorted to heuristic methods of evaluating the gate count of such matrices. Some such metrics like d-xor and s-xor count have been proposed earlier [JPST17]. However, in [KLSW18b], the authors showed that such heuristic metrics do not reflect accurately the minimum number of xor gates required to completely describe any linear system. Instead, the approach followed in [KLSW18b], was to try and find the shortest linear program of a given matrix by using approximation algorithms like the one proposed by Boyar-Peralta [BP10] and Paar [Paa97]. As a result, they proposed instantiations of several well known matrices in crypto-literature with a smaller number of xor gates than was previously known. In particular, they proposed a circuit for the AES mixcolumn matrix with only 97 xor gates, which was considerably lower than the best construction of 103 gates known at the time [JMPS17].

Shortest linear program is a well known hard problem in computer science. It is known that the problem is NP-complete (polynomially reducible to the Vertex-Cover problem): in fact it was proven in [BMP08], the problem is MAX-SNP complete, which roughly means that there are no good approximation algorithms for the problem unless P = NP. Nevertheless, over the years there have been many attempts at proposing approximation algorithms to solve the problem when the size of the input matrix is limited. One of the first such attempts was by Paar in [Paa97]. The algorithm is a essentially a greedy one, which at every stage finds the pair of operands that appear most frequently in the set of equations and replaces them with a new variable. The process continues until all operands appear exactly once. For obvious reasons, the algorithm only produces cancellation-free solutions to the problem. This basically means that if one takes any two intermediate operands in the algorithm and writes out the expression of each operand as a linear equation of the input variables, then the two expressions will not contain any common term. It is well-known that cancellation-free solutions are sub-optimal. There have been attempts to solve the problem using SAT solvers [FS10]. The authors of this paper showed that

the problem can be formulated as a SAT instance, i.e. if one wants to know if a given linear system can be described using t xor gates, one may frame the problem in such a manner so that a solution returned by the SAT solver would be a unique encoding of the underlying t-xor gate circuit. An optimal solution is reached when the solver returns a solution for some value of t but finds the instance unsatisfiable for $t-1$. SAT based solutions have been used before to minimize gate complexities of Sboxes [Sto16] using similar approaches. But the problem is that the running time of the solver itself is exponential in the size of the input and for input sizes larger than 10, it is difficult to get a solution from the solver in reasonable time. Another algorithm for the problem is due to Boyar-Peralta [BP10]. Unlike Paar's method, the algorithm may produce solutions with cancellation.

1.1 Contribution and Organization

In this paper we first show that both the Boyar-Peralta and the Paar algorithm can be executed with additional randomness to produce shorter linear programs for any given matrix. We explain how to efficiently incorporate additional randomness and give an intuitive explanation of why our approach works. As a result we produce shorter programs for almost all the matrices listed in [KLSW18b]. In particular, we propose an implementation of the AES mixcolumn matrix that takes only 95 2-input xor gates. Very recently however, there have been 2 papers which report implementation using only 94 xor gates [Max19, TP19].

As mentioned in the abstract, most standard cell libraries contain dedicated two input and three input xor gates. The hardware area of the 3-input xor gate is generally smaller than the sum of the areas of two 2-input xor gates. And if a logic synthesizer is presented with a functional description of any linear system in any hardware description language like VHDL or Verilog, and asked to produce a circuit that is optimized for area, it generally comes up with a circuit that utilizes both types of xor gates. In such a scenario, minimizing the area of the circuit implementing the linear system can no longer be achieved by computing the SLP solution. Indeed the solution that minimizes the hardware area would depend on the individual areas of the 2-input and 3-input xor gates, and this does not appear to be easier to solve than the SLP problem. In the second part of the paper, we present a graph based approximation algorithm that does the following: it takes as input an SLP solution and then encodes it as a directed graph. It then recursively alters the edges of the graph till a certain stopping criterion is reached. In the end we obtain a solution comprising both 2 and 3-input xor gates, which is smaller in area than the initial SLP solution that we started with. As a result we provide improved circuit implementation of all the matrices listed in [KLSW18b].

The rest of the paper is organized in the following manner. In Section 2, we give a brief description of the Boyar-Peralta and Paar algorithms and explain how randomness can be incorporated in the algorithm execution to produce shorter linear programs for a given linear system. In Section 3, we explain the working of our graph based approximation algorithm that produces a circuit for

a given linear system utilizing both 2 and 3-input xor gates. Section 4, concludes the paper.

2 Approximation algorithms

2.1 Boyar-Peralta method [BP10]

Before we proceed let us take a look at the Boyar-Peralta algorithm. The problem is to find a short linear program that computes $f(x) = Mx$ where M is an $m \times n$ matrix over $GF(2)$. The basic idea is as follows. A "base" S of known linear functions is first constructed. Initially S is just the set of input variables $x_1, x_2, \ldots, x_n$. The vector $Dist[\cdot]$ is the set of distances from S to the linear functions given by the rows of M. That is, if f_i is the linear function given by the i^{th} row of M then $Dist[i]$ represents the minimum number of functions from S that can add to give f_i. Consequently, we have that initially, $Dist[i]$ is just one less than the hamming weight of row i. The following steps are then performed in a loop:

- Choose a new base element by adding two existing base elements and add it to S.
- Update $Dist[i]$ since S has been modified.
- Do the above until $Dist[i] = 0$ for all i.

At any stage if the size of S is t there are $\binom{t}{2}$ options to choose a new base. The criterion for picking the new base element is

1. Pick one that minimizes the sum of elements of the updated $Dist[\cdot]$ array.
2. If there is a tie between two choices of the new base element, then resolve it by choosing the base element that maximizes the Euclidean norm of updated $Dist[\cdot]$ array.

This tie resolution criterion, may seem counter-intuitive. The basic idea is that a distance vector like 0,0,3,1 is preferred to one like 1,1,1,1. In the latter case, we would need 4 more gates to finish. In the former, 3 might do it. The bulk of the time of the heuristic is spent on picking the new base element.

Example 1. Before proceeding it may be instructive to look at a small example of the working of the above algorithm using an example take directly form [BP10]. Suppose we need a circuit that computes the system of equations defined as follows. This is equivalent to finding a circuit for multiplication by the 6×5 matrix, M given on the left.

$$\begin{aligned} x_0 \oplus x_1 \oplus x_2 &= y_0 \\ x_1 \oplus x_3 \oplus x_4 &= y_1 \\ x_0 \oplus x_2 \oplus x_3 \oplus x_4 &= y_2 \\ x_1 \oplus x_2 \oplus x_3 &= y_3 \\ x_0 \oplus x_1 \oplus x_3 &= y_4 \\ x_1 \oplus x_2 \oplus x_3 \oplus x_4 &= y_5 \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} 1\,1\,1\,0\,0 \\ 0\,1\,0\,1\,1 \\ 1\,0\,1\,1\,1 \\ 0\,1\,1\,1\,0 \\ 1\,1\,0\,1\,0 \\ 0\,1\,1\,1\,1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

The target functions to be computed are given rows of M. The initial base is given by $\{x_0, x_1, x_2, x_3, x_4\}$, which corresponds to $S = \{10000, 01000, 00100, 00010, 00001\}$. The initial distance vector is $Dist = [2, 2, 3, 2, 2, 3]$. The algorithm finds two base vectors whose sum, when added to the base, minimizes the sum of the new distances. It turns out the right choice is to calculate $t_5 = x_1 \oplus x_3$. So the new base S is expanded to contain the signal 01010. The new distance vector is $Dist = [2, 1, 3, 1, 1, 2]$.

Step 2 According to the algorithm we have a choice between $x_1 \oplus x_2$ and $x_0 \oplus t_5$. The updated $Dist$ vectors for the above choices are respectively $[1, 1, 3, 1, 1, 2]$ and $[2, 1, 3, 1, 0, 2]$, both of which sum to 9. However the second choice gives a Euclidean norm of $\sqrt{19}$. So we choose $t_6 = x_0 \oplus t_5 = y_4$.
Step 3 $t_7 = x_2 \oplus t_5 = y_3$, new $Dist = [2, 1, 3, 0, 0, 1]$.
Step 4 $t_8 = x_4 \oplus t_5 = y_1$, new $Dist = [2, 0, 3, 0, 0, 1]$.
Step 5 $t_9 = x_2 \oplus t_8 = y_5$, new $Dist = [2, 0, 2, 0, 0, 0]$.
Step 6 $t_{10} = x_0 \oplus x_1$, new $Dist = [1, 0, 1, 0, 0, 0]$.
Step 7 $t_{11} = x_2 \oplus t_{10} = y_0$, new $Dist = [0, 0, 1, 0, 0, 0]$.
Step 8 $t_{12} = t_8 \oplus t_{11} = y_2$, new $Dist = [0, 0, 0, 0, 0, 0]$.

This therefore gives a circuit with eight gates.

2.2 Our Experiments

The authors of [KLSW18b] were kind enough to make all the codes used by them freely available in the public domain [KLSW18a]. We downloaded the C++ code for the Boyar-Peralta algorithm which is based on the code available at [BP18] written by the authors of [BP10]. We ran the code for the AES mixcolumn matrix and found that it returned a solution with **96** xor gates. This was surprising for us, since the authors of [KLSW18b] claim that their implementation of the AES mixcolumn matrix takes 97 xor gates. Initially we concluded that it must have been an error by the authors of [KLSW18b]. But on closer inspection we started to make a sense of why the discrepancy arose.

In the code used by [KLSW18a], the ordering of the input byte in terms of bits is as follows: $[x_0, x_1, x_2, \ldots, x_7]$ which essentially means that they place the least significant bit first, whereas the ordering we used is $[x_7, x_6, x_5, \ldots, x_0]$ which essentially means most significant bit first and arranging bits in decreasing index order. This means that the AES mixcolumn matrices we and the authors of [KLSW18b] have used in our respective experiments would be column and row shuffled versions of each other. However there is nothing in the steps of the Boyar-Peralta algorithm that suggests that if we present column/row shuffled instances of the same matrix to the algorithm, it would output different solutions. In fact the algorithm picks out a new base element at each step which minimizes the sum of given distance vector, and computes a Euclidean norm to resolve ties. Since this sum or norm should not change no matter how the columns/rows are arranged, there is every reason to believe that that the output of the algorithm should be independent of how the matrix is arranged, if the underlying linear system is unchanged.

However as it turns out, the way in which the algorithm has been implemented in [KLSW18a], it **does** output different results when different column shuffled instances of the same matrix is input. The reason this happens is as follows. Following is a snippet of the C++ code where the algorithm implements resolution of ties via Euclidean norm:

```
MinDistance = BaseSize*NumTargets; //i.e. something big
OldNorm = 0; //i.e. something small
for (int i = 0; i < BaseSize - 1; i++) {
       for (int j = i+1; j < BaseSize; j++) {
           NewBase = Base[i] ^ Base[j];

           ThisDist = TotalDistance();//also calculates NDist[]
           if (ThisDist <= MinDistance) {
               //calculate Norm
               ThisNorm = 0;
               for (int k = 0; k < NumTargets; k++) {
                   d = NDist[k];
                   ThisNorm = ThisNorm + d*d;
               }
               //resolve tie in favor of largest norm
               if((ThisDist < MinDistance)||(ThisNorm > OldNorm))

               {
                   besti = i;
                   bestj = j;
                   TheBest = NewBase;
                   for (int uu = 0; uu < NumTargets; uu++) {
                       BestDist[uu] = NDist[uu];
                   }
                   MinDistance = ThisDist;
                   OldNorm = ThisNorm;
               }
           }
       }
   }
  //update Dist array
   NewBase = TheBest;
   for (int i = 0; i < NumTargets; i++) {
       Dist[i] = BestDist[i];
   }
   //update Base with TheBest
   Base[BaseSize] = TheBest;
```

The above code is intuitively easy to understand and follow. We describe it briefly. The variable `BaseSize` stores the current size of S. The algorithm then loops over all choices of the new base element. A candidate base element

is placed in the global variable `NewBase`, and then the code computes the temporary updated distance vector `NDist[]` via the function `TotalDistance` which returns the sum of `NDist[]` in `ThisDist`. If `ThisDist` is less than or equal to the current minimum stored in `MinDistance`, the code then computes the Euclidean norm of `NDist[]` in `ThisNorm`. A final choice of new base element is made in the variable `TheBest`, depending on whether the current candidate produces a sum that is absolutely less than the current minimum `MinDistance` or it produces a Euclidean norm strictly greater than the current maximum norm `OldNorm`. Consider what happens when two candidates for the new base element say for $\texttt{i=i}_0\texttt{,j=j}_0$ and $\texttt{i=i}_1\texttt{,j=j}_1$ produce identical values of `ThisDist` and `ThisNorm`. According to the code, the new candidate would be the one which appears first lexicographically in the double loop traversal of `i,j`. In general if such a situation appears for n choices for the new candidate base element, the code all always chooses the one that appears first in the double loop traversal. Thus it becomes clearer why the order in which the columns are arranged matrix is crucial: shuffling of columns essentially means we shuffle the order of input variables to the system, which in turn implies we shuffle the initial placement of elements of S. This has a direct effect on how new candidates are chosen to be added to S and takes the program execution in different directions.

2.3 New idea

In the paper [BP10], the authors suggest other methods to resolve ties including choosing random candidate elements. We did not take this approach for two reasons: one it may not necessarily lead to optimal solutions and second it is not necessarily straightforward to adapt the code snippet to accommodate random candidate choices. However since the order of rows/columns seems to bring about a change in the output of this particular code execution: we tried the following idea. We take the target matrix M and multiply with randomly generated permutation matrices P and Q to get $M_R = P \cdot M \cdot Q$. This only shuffles the rows and columns of the matrix and so keeps the underlying linear system unchanged. We use M_R as input to the C++ code and extract a solution. The code could be run multiple number of times with random permutation matrices until we get a solution better than previously obtained.

We started our experiment with the AES mixcolumn matrix. After around 4 hours of execution we obtained a solution with 95 xor gates. The solution is presented in Table 1. Note that the permutation matrices $P,\ Q$ has been listed as a table. For example, $PT[0] = 4$ implies that in the 0^{th} row of P, the element in the 4^{th} column is 1 and the rest are 0.

2.4 Paar's algorithm

Paar's algorithm is essentially a greedy one, which at every stage finds the pair of operands that appear most frequently in the set of equations and replaces them with a new variable. The process continues until all operands appear exactly once. The algorithm however returns cancellation-free solutions that are known

Table 1: AES mixcolumn using 95 xor gates

#	Gate	#	Gate	#	Gate
1	t0 = x0 + x2	33	t32 = x9 + t3	65	t64 = x1 + t0
2	t1 = x10 + x24	34	y21 = t9 + t32	66	t65 = t6 + t13
3	t2 = x0 + x10	35	t34 = x19 + t6	67	y4 = t64 + t65
4	t3 = x2 + x24	36	y20 = t2 + t34	68	t67 = t10 + t32
5	t4 = x3 + x5	37	t36 = t1 + t9	69	t68 = x31 + t34
6	t5 = x4 + x11	38	y18 = t34 + t36	70	y5 = t67 + t68
7	t6 = x9 + x20	39	t38 = x20 + t9	71	t70 = y18 + y4
8	t7 = x16 + x29	40	y26 = t0 + t38	72	y27 = t67 + t70
9	t8 = x6 + x23	41	t40 = x11 + t8	73	t72 = x13 + t1
10	t9 = x19 + x21	42	y9 = t14 + t40	74	t73 = t15 + t16
11	t10 = x1 + x7	43	t42 = x13 + t5	75	y14 = t72 + t73
12	t11 = x12 + x26	44	t43 = x6 + x17	76	t75 = x15 + t2
13	t12 = x8 + x30	45	y15 = t42 + t43	77	t76 = x27 + t16
14	t13 = x18 + x31	46	t45 = x11 + y10	78	t77 = t0 + t17
15	t14 = x13 + x14	47	y16 = t29 + t45	79	t78 = t14 + t77
16	t15 = x17 + x28	48	t47 = x12 + t12	80	y24 = x28 + t78
17	t16 = x15 + x22	49	y28 = t13 + t47	81	t80 = t75 + t76
18	t17 = x25 + x27	50	t49 = x18 + t11	82	t81 = x17 + t14
19	t18 = x2 + t1	51	t50 = x1 + y28	83	y2 = t80 + t81
20	y8 = t7 + t18	52	y7 = t49 + t50	84	t83 = t3 + t11
21	t20 = x16 + t2	53	t52 = x7 + x8	85	y3 = t76 + t83
22	t21 = x2 + t2	54	y30 = t49 + t52	86	t85 = t12 + t17
23	y29 = t4 + t21	55	t54 = x18 + t10	87	y1 = t75 + t85
24	t23 = x3 + y8	56	y12 = t36 + t54	88	t87 = y14 + t78
25	y25 = t20 + t23	57	t56 = x23 + t5	89	y13 = t80 + t87
26	t25 = x5 + x24	58	y17 = t15 + t56	90	t89 = x26 + x30
27	y23 = t20 + t25	59	t58 = x28 + y9	91	t90 = t77 + t89
28	t27 = x5 + t7	60	y19 = t42 + t58	92	y6 = t76 + t90
29	y10 = t8 + t27	61	t60 = x29 + t4	93	t92 = x25 + t83
30	t29 = x23 + t4	62	y31 = t5 + t60	94	t93 = y1 + y6
31	t30 = x4 + x16	63	t62 = x30 + t10	95	y0 = t92 + t93
32	y11 = t29 + t30	64	y22 = t11 + t62		
PT=[4,12,11,28, 22,30,20,5, 16,26,9,17, 6,27,3,18, 1,10,7,2, 15,31,13,24, 19,8,23,14, 29,0,21,25]					
QT=[24,5,11,14, 15,12,31,19, 10,3,6,28, 22,8,1,21, 0,29,23,17, 27,30,7,9, 2,16,4,13, 25,26,18,20]					

to be not always optimal. But it is always useful to use this algorithm to decrease the gate count of larger matrices for which the Boyar-Peralta method is unable to return a solution in practical time.

Let us look at the details of the algorithm. Let M be the matrix whose gate count is to be minimized. Then the algorithm performs the following steps:

Step 1 Find columns whose bitwise AND has largest weight. This essentially finds two operands x_i, x_j whose xor occurs most number of times in the underlying linear system.

Step 2 Extend matrix M, by adding the above product column *newcol* to the matrix.

Step 3 For the two previous columns do $oldcol \leftarrow oldcol \cdot \overline{newcol}$. The above two steps adds the xor gate $v = x_i \oplus x_j$ to the gate list and by adding the product column to M creates a new input variable v. By doing $oldcol \leftarrow oldcol \cdot \overline{newcol}$, the algorithm removes extra xors in the matrix structure, which are no longer needed after the addition of the new column.

Example 2. It is again instructive to understand the algorithm with a small example. Given the following linear system.

$$\begin{aligned} x_1 \oplus x_2 &= y_1 \\ x_1 \oplus x_2 \oplus x_3 &= y_2 \\ x_1 \oplus x_2 \oplus x_3 \oplus x_4 &= y_3 \\ x_2 \oplus x_3 \oplus x_4 &= y_4 \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} 1&1&0&0 \\ 1&1&1&0 \\ 1&1&1&1 \\ 0&1&1&1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

In this example, the product of the second and third column has largest weight. We have $v_1 = x_2 + x_3$. The new column to be added is $(1\ 1\ 1\ 1) \cdot (0\ 1\ 1\ 1) = (0\ 1\ 1\ 1)$, and after the $oldcol \leftarrow oldcol \cdot \overline{newcol}$ step we have the following system.

$$\begin{aligned} x_1 \oplus x_2 &= y_1 \\ x_1 \oplus v_1 &= y_2 \\ x_1 \oplus x_4 \oplus v_1 &= y_3 \\ x_4 \oplus v_1 &= y_4 \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} 1&1&0&0&0 \\ 1&0&0&0&1 \\ 1&0&0&1&1 \\ 0&0&0&1&1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ v_1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

The above steps are continued until all targets are achieved.

Again, there are no steps in the above algorithm that suggest that different results would be output if the input matrices are row/column shuffled. However again due to the C++ implementation of the above algorithm in [KLSW18a], column shuffled versions of the same matrix do produce different outcomes. The reasons due too are quite similar: for candidate column pairs C_{i_0}, C_{j_0} and C_{i_1}, C_{j_1} both of whose products have the same hamming weight, the code in [KLSW18a] chooses the one which occurs first lexicographically during a standard double loop search over every pair of columns. Thus shuffling of columns of M directly impacts the outcome. Again we could multiply with randomly generated permutation matrices P and Q to get $M_R = P \cdot M \cdot Q$. The code is run multiple number of times with random permutation matrices until we get a solution better than previously obtained.

Note that in the original paper [Paa97], two algorithms were proposed of which we have discussed the first one. The second algorithm recurses on all possible choices of candidate intermediate steps and thus will output the optimal result. The probabilistic version suggested in this paper is thus something in between the two original algorithms. We did not try to implement Paar's second algorithm as it took a lot of time to run on the computing systems we had access to.

2.5 Results

We ran the the modified algorithms for all the matrices listed in [KLSW18b]. For smaller 32×32 matrices the Boyar-Peralta algorithm can be executed efficiently in reasonable time. For 64×64 and larger matrices we used the modified Paar algorithm as the Boyar-Peralta took an unreasonable amount of time just to execute the optimization of one shuffled version of the target matrix M. The results are given in Table 2 and Table 3. Table 2 contains matrices proposed in recent literature, whereas Table 3 contains matrices used in some cryptographic constructions. For almost all matrices we have improved the results of [KLSW18b].

3 Optimization with 3-input xor gates

One of the motivations of constructing circuits with lower number of xor gates is that it makes for a more lightweight implementation in hardware. However most standard cell libraries of CMOS logic processes have dedicated gates that support both the 2-input and the 3-input xor functionality. Generally the area of a 3-input xor gate is lower than the area of two 2-input xor gates. Take for example, the standard cell library `CORE90GPHVT v 2.1.a` of the STM 90nm CMOS logic process. It has two types of gates

- 2-input xor gate with area: 2 GE
- 3-input xor gate with area: 3.25 GE

where GE refers to Gate equivalents which is the area of a two input NAND gate. Our experiments started with the AES mixcolumn matrix. We presented a functional description of the matrix written in VHDL to the Synopsys design compiler and instructed it to compile a circuit optimized for area. It returned a solution with 39 3-input xor and 31 2-input xor gates. The area of the above circuit is $39 * 3.25 + 31 * 2 = 188.75$ GE which is less than $2 * 95 = 190$ GE (the area of 95 2-input xor gates).

However the situation is different for another library `CORE65GPHVT v 5.1` which is based on the STM 65nm CMOS logic process. It has two types of xor gates listed as follows:

- 2-input xor gate with area: 1.981 GE
- 3-input xor gate with area:: 3.715 GE

Table 2: Comparison of gate counts for matrices available in literature

#	Matrix	Type	Gate count in [KLSW18b]	Gate count in this paper
				4×4 matrices over $GF(2^4)$
1	[SKOP15]	Hadamard	48	46
2	[LS16]	Circulant	44	44
3	[LW16]	Circulant	44	44
4	[BKL16]	Circulant	42	42
5	[SS16]	Toeplitz	43	42
6	[JPST17]		43	42
7	[SKOP15]	Hadamard, Involutary	48	47
8	[LW16]	Hadamard, Involutary	48	46
9	[SS16]	Involutary	42	40
10	[JPST17]	Involutary	47	46
				4×4 matrices over $GF(2^8)$
11	[SKOP15]	Subfield	98	94
12	[LS16]	Circulant	112	110
13	[LW16]		102	102
14	[BKL16]	Circulant	110	108
15	[SS16]	Toeplitz	107	104
16	[JPST17]	Subfield	86	86
17	[SKOP15]	Subfield, Involutary	100	94
18	[LW16]	Hadamard, Involutary	91	90
19	[SS16]	Involutary	100	98
20	[JPST17]	Subfield, Involutary	91*	92
				8×8 matrices over $GF(2^4)$
21	[SKOP15]	Hadamard	194	192
22	[SS17]	Toeplitz	204	203
23	[SKOP15]	Hadamard, Involutary	217	212
				8×8 matrices over $GF(2^8)$
24	[SKOP15]	Hadamard	467	460
25	[LS16]	Circulant	447	443
26	[BKL16]	Circulant	498	497
27	[SS17]	Toeplitz	438	436
28	[SKOP15]	Hadamard, Involutary	428	419
29	[JPST17]	Hadamard, Involutary	599	591

* On running the code from [KLSW18a] on our PC, we got solution 92 for this matrix

Table 3: Comparison of gate counts for matrices used in cryptographic constructions

#	Cipher	Type	Gate Count [KLSW18b]	Gate Count This paper
			4×4 matrices over $GF(2^8)$	
1	AES [DR02]	Circulant	97	95
2	ANUBIS [BR00a]	Hadamard, Involutary	113	102
3	CLEFIA M_0* [SSA+07]	Hadamard, Involutary	106	102
4	CLEFIA M_1 [SSA+07]	Hadamard	111	110
5	FOX MU4 [JV04]		137	131
6	TWOFISH [SKW+98]		129	125
			8×8 matrices over $GF(2^8)$	
7	FOX MU8 [JV04]		594	592
8	GRØSTL [GKM+09]	Circulant	475	460
9	KHAZAD [BR00b]	Hadamard, Involutary	507	492
10	WHIRLPOOL [BR11]	Circulant	465	464
			4×4 matrices over $GF(2^4)$	
11	JOLTIK [JNP13]	Hadamard, Involutary	48	47
12	SMALLSCALE AES [CMR05]	Circulant	47	45
			8×8 matrices over $GF(2^4)$	
13	WHIRLWIND M_0 [BNN+10]	Hadamard, Subfield	212	210
14	WHIRLWIND M_1 [BNN+10]	Hadamard, Subfield	235	234
			Non MDS matrices	
15	QARMA128 [Ava17]	Circulant (4×4 over $GF(2^8)$)	48	48
16	ARIA [KKP+03]	Involutary (16×16 over $GF(2^8)$)	416	392
17	MIDORI [BBI+15]	Involutary (4×4 over $GF(2^4)$)	24	24
18	PRINCE M_0, M_1 [BCG+12]	(16×16 over $GF(2)$)	24	24
19	PRIDE L_0-L_3 [ADK+14]	(16×16 over $GF(2)$)	24	24
20	QARMA64 [Ava17]	Circulant (4×4 over $GF(2^4)$)	24	24
21	SKINNY64 [BJK+16]	(4×4 over $GF(2^4)$)	12	12

* ANUBIS and CLEFIA M_0 matrices are the same. [KLSW18b] gives different results for them. It might have been an error.

If we took the previous solution and tried to apply it to this library the gate area would be $39 * 3.715 + 31 * 1.981 = 206.296$ GE which is much more than $95 * 1.981 = 188.195$ GE that would be obtained by using only the 2-input xor gate. In fact when we repeated the exercise for this library and asked the design compiler to synthesize an area optimized circuit it returned a solution with 38 3-input xors and 32 2-input xors which amounts to 204.56 GE. The experiments bring out three crucial facts: **a)** The SLP solution does not always represent the optimal circuit area when the circuit compiler can additionally use 3-input xor gates, **b)** the optimal solution is heavily dependent on the target standard cell library, a solution that is optimal for a given library may not be optimal for a different library, **c)** the solutions returned by circuit compilers may also not represent the optimal solution in terms of circuit area.

3.1 Incremental graph based technique

Since a single 3-input xor is smaller in area than two 2-input xors, we started with the rule of thumb that, given any matrix, we should convert all instances of two 2-input xors to a single 3-input xor wherever possible. However, consider the following linear system before proceeding.

Example 3. $y_1 = x_1 \oplus x_2 \oplus x_3$, $y_2 = x_2 \oplus x_3 \oplus x_4$.

One could solve the above problem in a straightforward manner by using two 3-input xor gates that would cost 6.5 GE in the 90nm library and around 7.4 GE in the 65 nm library. However an SLP based solution that reuses the sum $x_2 \oplus x_3$ could give us a solution using three 2-input xors, that would cost around 6 GE in both libraries.

- Solution: $t_1 = x_2 \oplus x_3$, $y_2 = t_1 \oplus x_4$ and $y_1 = x_1 \oplus t_1$.

Let us say we already have a SLP solution for a given matrix obtained by either the Boyar-Peralta or the Paar method and we want to take this solution as a starting point and make incremental modifications to it to get a circuit that uses both 2 and 3-input xor gates. The standard way to do this would be to check if there are pairs of 2-input xors in the original SLP solution that could be replaced with a 3-input xor gate thereby reducing the area of the circuit. This approach has an additional advantage that the final solution of this approach is guaranteed to have less area than the corresponding SLP solution, irrespective of the library of synthesis. Given the information in the above example, a handy way to proceed is to check if the output of a particular 2-input xor gate is used multiple times in the circuit. If so then it is best to avoid removing this xor gate from the SLP solution to facilitate insertion of another 3-input xor gate. Let us formalize this intuitive approach.

Let L be the SLP solution for an underlying linear system given by the matrix M. Each line of L represents a 2-input xor gate used to implement the circuit. Define a directed graph $G = (V, E)$ in the following manner. Each line of L is a vertex in the graph: thus the size of $|V|$ is simply the length of L. Two

vertices v_i, v_j are connected by a directed edge in E, if the output of the xor gate represented by v_i is an input to the xor gate represented by v_j. In such a graph, a node with outdegree strictly equal to 1 are those whose outputs are used only once. Nodes with outdegree 0 represent the gates which produce the output bits of the linear system, although it may be possible that nodes with larger outdegree produces output bits of the system. All other nodes are those that are used multiple number of times. Thus our strategy would be as follows:

1. Make a list of all nodes of outdegree 1 and 0, and additionally those nodes that produce output bits but have outdegree larger than 0.
2. If there exist two nodes v_i, v_j such that $(v_i, v_j) \in E$ and outdegree$(v_i) = 1$ and v_i does not produce an output of the underlying linear system then merge them to form a new node X that represents a 3-xor gate in the following manner: the incoming edges of v_i and v_j are made the incoming edges of X and if the outdegree of v_j is 1, the outgoing edge of v_j is made the outgoing edge of X.
3. Essentially what the above step does is as follows: if v_i represents the 2-input xor gate $t = x \oplus y$ and v_j represents the 2-input xor gate $u = t \oplus z$, then the outdegree of $v_i = 1$, guarantees that t does not appear elsewhere in the SLP solution. We merge them to a 3-input xor node X representing $u = x \oplus y \oplus z$. Note that if $t = x \oplus y$ was an output of the system then the above merge procedure would have proven counter-productive because we would have lost the output signal t after the merge procedure.
4. The algorithm is recursively executed until all nodes of outdegree 1 of the required property are exhausted.

The algorithm starts with an SLP solution for a given matrix obtained by either the Boyar-Peralta or the Paar method, and runs the above steps iteratively until a solution is found. Since permutation matrices P, Q for which we get the optimal SLP solution for $M_R = P \cdot M \cdot Q$, may not necessarily lead to the optimal area after running the above algorithm, we run the above algorithm for a number of randomly generated P, Q till a solution is obtained. In particular, for the AES mixcolumn matrix we were able to get a solution with 39 2-input xor and 28 3-input xor gates. This gives an area of 169 GE with the `CORE90GPHVT v 2.1.a` library and 181.3 GE with `CORE65GPHVT v 5.1`. This is well below the hardware area of the corresponding SLP solution.

3.2 Results

We applied the above algorithm to all the SLP solutions that are listed above in Section 2. For all the matrices that we experimented with, we obtained a circuit implementation smaller in area than the area of the corresponding SLP solution. The results are presented in Tables 4 and 5. We compare the area of the circuit obtained after running the algorithm with the corresponding area of the SLP solution (for the sake of conciseness we fix this value to 2 times the length of SLP solution, in GE). Results for both the libraries `CORE90GPHVT v 2.1.a` and `CORE65GPHVT v 5.1` are tabulated.

Table 4: Comparison of areas for matrices available in literature. Lib1 and Lib2 refer to `CORE90GPHVT v 2.1.a` and `CORE65GPHVT v 5.1` respectively.

#	Matrix	Type	# 2-xor	#3-xor	Area in GE SLP	Lib 1	Lib 2
					4×4 matrices over $GF(2^4)$		
1	[SKOP15]	Hadamard	26	10	92.0	84.5	88.7
2	[LS16]	Circulant	20	12	88.0	79.0	84.2
3	[LW16]	Circulant	20	12	88.0	79.0	84.2
4	[BKL16]	Circulant	18	12	84.0	75.0	80.2
5	[SS16]	Toeplitz	18	12	84.0	75.0	80.2
6	[JPST17]		13	15	84.0	74.8	81.5
7	[SKOP15]	Hadamard, Involutary	16	16	94.0	84.0	91.1
8	[LW16]	Hadamard, Involutary	13	15	86.0	74.8	81.5
9	[SS16]	Involutary	20	10	80.0	72.5	76.8
10	[JPST17]	Involutary	16	15	92.0	80.8	87.4
					4×4 matrices over $GF(2^8)$		
11	[SKOP15]	Subfield	45	25	188.0	171.3	182.0
12	[LS16]	Circulant	28	42	220.0	192.5	211.5
13	[LW16]		31	35	204.0	175.8	191.4
14	[BKL16]	Circulant	40	34	216.0	190.5	205.6
15	[SS16]	Toeplitz	26	40	208.0	182.0	200.1
16	[JPST17]	Subfield	26	30	172.0	149.5	163.0
17	[SKOP15]	Subfield, Involutary	32	32	188.0	168.0	182.3
18	[LW16]	Hadamard, Involutary	48	21	180.0	164.3	173.1
19	[SS16]	Involutary	44	27	196.0	175.8	187.5
20	[JPST17]	Subfield, Involutary	36	28	184.0	163.0	175.3
					8×8 matrices over $GF(2^4)$		
21	[SKOP15]	Hadamard	78	57	384.0	341.3	366.2
22	[SS17]	Toeplitz	87	58	406.0	362.5	387.8
23	[SKOP15]	Hadamard, Involutary	90	61	424.0	378.3	404.8
					8×8 matrices over $GF(2^8)$		
24	[SKOP15]	Hadamard	181	141	920.0	820.3	882.2
25	[LS16]	Circulant	181	141	920.0	820.3	882.2
26	[BKL16]	Circulant	157	144	886.0	782.0	845.8
27	[SS17]	Toeplitz	153	144	872.0	782.0	837.9
28	[SKOP15]	Hadamard, Involutary	167	126	838.0	743.5	798.8
29	[JPST17]	Hadamard, Involutary	205	193	1182.0	1037.3	1022.9

Table 5: Comparison of areas for matrices used in cryptographic constructions. Lib1 and Lib2 refer to `CORE90GPHVT v 2.1.a` and `CORE65GPHVT v 5.1` respectively.

#	Matrix	Type	#2-xor	#3-xor	Area in GE SLP	Lib 1	Lib 2
					4×4 matrices over $GF(2^8)$		
1	AES [DR02]	Circulant	39	28	190.0	169.0	181.3
2	ANUBIS [BR00a]	Hadamard and Involutary	60	20	200.0	185.0	193.2
3	CLEFIA M_0 [SSA$^+$07]	Hadamard Involutary	60	20	200.0	185.0	193.2
4	CLEFIA M_1 [SSA$^+$07]	Hadamard	38	36	220.0	193.0	209.0
5	FOX MU4 [JV04]		46	43	262.0	231.8	250.9
6	TWOFISH [SKW$^+$98]		43	42	250.0	222.5	241.2
					8×8 matrices over $GF(2^8)$		
7	FOX MU8 [JV04]		212	190	1184.0	1041.5	1125.8
8	GRØSTL [GKM$^+$09]	Circulant	190	129	920.0	799.3	855.6
9	KHAZAD [BR00b]	Hadamard and Involutary	224	134	984.0	883.5	941.6
10	WHIRLPOOL [BR11]	Circulant	154	155	928.0	811.8	880.9
					4×4 matrices over $GF(2^4)$		
11	JOLTIK [JNP13]	Hadamard and Involutary	16	16	94.0	84.0	91.1
12	SMALLSCALE AES [CMR05]	Circulant	19	13	90.0	80.3	85.9
					8×8 matrices over $GF(2^4)$		
13	WHIRLWIND M_0 [BNN$^+$10]	Hadamard and Subfield	82	64	420.0	372.0	400.2
14	WHIRLWIND M_1 [BNN$^+$10]	-do-	96	69	468.0	416.3	446.5
					Non MDS matrices		
15	QARMA128 [Ava17]	Circulant (4×4 over $GF(2^8)$)	34	7	96.0	90.8	93.4
16	ARIA [KKP$^+$03]	Involutary (16×16 over $GF(2^8)$)	136	128	784.0	688.0	744.8
17	MIDORI [BBI$^+$15]	Involutary (4×4 over $GF(2^4)$)	16	4	48.0	45.0	46.6
18	PRINCE M_0, M_1 [BCG$^+$12]	(16×16 over $GF(2)$)	16	4	48.0	45.0	46.6
19	PRIDE L_0-L_3 [ADK$^+$14]	(16×16 over $GF(2)$)	16	4	48.0	45.0	46.6
20	QARMA64 [Ava17]	Circulant (4×4 over $GF(2^4)$)	16	4	48.0	45.0	46.6
21	SKINNY64 [BJK$^+$16]	(4×4 over $GF(2^4)$)	12	0	24.0	24.0	24.0

4 Conclusion

In this paper we took another look at the shortest linear program problem for implementing linear systems. We found implementation issues that may result in a situation where the order of appearance of columns in a matrix affect the outcome of heuristic based algorithms like the ones due to Boyar-Peralta and Paar. We showed that by suitably including randomness in the execution of these algorithms it is possible to obtain even more efficient solutions to the SLP problem. We applied our method to the diffusion layer matrices of well known constructions in literature. We were able to improve the number of xor gates required for the implementations for most of these matrices. We have also reported an implementation of the AES mixcolumn matrix that uses only 95 xor gates which is one of the smallest implementations of the AES mixcolumn circuit.

In the second part of the paper, we observed that most standard cell libraries contain both 2 and 3-input xor gates, with the silicon area of the 3-input xor gate being smaller than the sum of the areas of two 2-input xor gates. Hence when linear circuits are synthesized by logic compilers (with specific instructions to optimize for area), most of them would return a solution circuit containing both 2 and 3-input xor gates. Thus from a practical point of view, reducing circuit size in presence of these gates was not equivalent to solving the shortest linear program. In this paper we showed that by adopting a graph based heuristic it is possible to convert a circuit constructed with 2-input xor gates to another functionally equivalent circuit that utilizes both 2 and 3-input xor gates and occupies less hardware area. As a result we obtain more lightweight implementations of all the matrices listed in first half of the paper.

Acknowledgments: Subhadeep Banik is supported by the Ambizione Grant `PZ00P2_179921`, awarded by the Swiss National Science Foundation (SNSF). Takanori Isobe is supported by Grant-in-Aid for Scientific Research (B) (KAKENHI 19H02141) for Japan Society for the Promotion of Science.

References

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

AF14. Daniel Augot and Matthieu Finiasz. Direct construction of recursive MDS diffusion layers using shortened BCH codes. In *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, pages 3–17, 2014.

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

BBI+15. Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT (2)*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.

BCG+12. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

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*, pages 123–153, 2016.

BKL16. Christof Beierle, Thorsten Kranz, and Gregor Leander. Lightweight multiplication in gf(2^n) with applications to MDS matrices. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 625–653, 2016.

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, Torun, Poland, August 25-29, 2008, Proceedings*, pages 168–179, 2008.

BNN+10. Paulo S. L. M. Barreto, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Elmar Tischhauser. Whirlwind: a new cryptographic hash function. *Des. Codes Cryptography*, 56(2-3):141–162, 2010.

BP10. Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings*, pages 178–189, 2010.

BP18. Joan Boyar and René Peralta. C++ implementation of slp algorithm, 2018. Available at `http://www.imada.sdu.dk/~joan/xor/Improved2.cc`.

BR00a. Paulo S. L. M. Barreto and Vincent Rijmen. The anubis block cipher, 2000. Submission to NESSIE project. Available at `https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions/anubis.zip`.

BR00b. Paulo S. L. M. Barreto and Vincent Rijmen. The khazad legacy-level block cipher, 2000. Submission to NESSIE project. Available at `https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions/khazad.zip`.

BR11. Paulo S. L. M. Barreto and Vincent Rijmen. Whirlpool. In *Encyclopedia of Cryptography and Security, 2nd Ed.*, pages 1384–1385. 2011.

CMR05. Carlos Cid, Sean Murphy, and Matthew J. B. Robshaw. Small scale variants of the AES. In *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, pages 145–162, 2005.

DR02. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag, Berlin, Heidelberg, New York, 2002.

FS10. Carsten Fuhs and Peter Schneider-Kamp. Synthesizing shortest linear straight-line programs over GF(2) using SAT. In *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, pages 71–84, 2010.

GKM+09. Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl - a SHA-3 candidate. In *Symmetric Cryptography, 11.01. - 16.01.2009*, 2009.

GPP11. Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 222–239, 2011.

GPPR11. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

GPV17. Kishan Chand Gupta, Sumit Kumar Pandey, and Ayineedi Venkateswarlu. Towards a general construction of recursive MDS diffusion layers. *Des. Codes Cryptography*, 82(1-2):179–195, 2017.

GR15. Kishan Chand Gupta and Indranil Ghosh Ray. Cryptographically significant MDS matrices based on circulant and circulant-like matrices for lightweight applications. *Cryptography and Communications*, 7(2):257–287, 2015.

JMPS17. Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. Bit-sliding: A generic technique for bit-serial implementations of spn-based primitives - applications to aes, PRESENT and SKINNY. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 687–707, 2017.

JNP13. Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Joltik v1.3, 2013. Submission to Caesar competition. Available at `https://competitions.cr.yp.to/round2/joltikv13.pdf`.

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

JV04. Pascal Junod and Serge Vaudenay. FOX : A new family of block ciphers. In *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, pages 114–129, 2004.

KKP+03. Daesung Kwon, Jaesung Kim, Sangwoo Park, Soo Hak Sung, Yaekwon Sohn, Jung Hwan Song, Yongjin Yeom, E-Joong Yoon, Sangjin Lee, Jaewon Lee, Seongtaek Chee, Daewan Han, and Jin Hong. New block cipher: ARIA. In *Information Security and Cryptology - ICISC 2003, 6th International Conference, Seoul, Korea, November 27-28, 2003, Revised Papers*, pages 432–445, 2003.

KLSW18a. Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Github repository: Shorter linear slps for mds matrices, 2018. Avail-

able at https://github.com/rub-hgi/shorter_linear_slps_for_mds_matrices.

KLSW18b. Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Shorter Linear Straight-Line Programs for MDS Matrices. *IACR Trans. Symmetric Cryptol.*, 2018(4):188–211, 2018.

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, 2016.

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, 2016.

Max19. Alexander Maximov. Aes mixcolumn with 94 xor gates, 2019. Available at https://eprint.iacr.org/2019/833.pdf.

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

SKOP15. Siang Meng Sim, Khoongming Khoo, Frédérique E. 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*, pages 471–493, 2015.

SKW+98. Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-bit block cipher, 1998. Available at https://www.schneier.com/academic/paperfiles/paper-twofish-paper.pdf.

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

SS17. Sumanta Sarkar and Habeeb Syed. Analysis of toeplitz MDS matrices. In *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part II*, pages 3–18, 2017.

SSA+07. Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2007.

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*, pages 140–160, 2016.

TP19. Quan Quan Tan and Thomas Peyrin. Improved heuristics for short linear programs, 2019. Available at https://eprint.iacr.org/2019/847.pdf.