Three Input Exclusive-OR Gate Support For Boyar-Peralta's Algorithm

Anubhab Baksi[1] Vishnu Asutosh Dasu[2] Banashri Karmakar[3]
Anupam Chattopadhyay[1] Takanori Isobe[4]

[1]Nanyang Technological University, Singapore

[2]TCS Research and Innovation, Bangalore, India

[3]Indian Institute of Technology, Bhilai, India

[4]University of Hyogo, Kobe, Japan

Contact: vishnu.dasu1@tcs.com

Outline

Introduction

- Linear layers constitute an important part of modern ciphers as they are responsible in spreading diffusion to the entire state.
- A linear layer can be expressed as a binary non-singular matrix and can be implemented using XOR gates only.
- The Boyar-Peralta's algorithm [Boyar and Peralta, 2010] aims at finding efficient implementation of a linear layer using XOR2 gates.
- The original version is presented over a decade ago, but there is a renewed interest [Maximov, 2019, Tan and Peyrin, 2019, Banik et al., 2019].

Notions of XOR Count

Definition (d-XOR Count)

The d-XOR count of the binary matrix $M^{m \times n}$ is defined as $d(M) = \mathrm{HW}(M) - m$, where $\mathrm{HW}(\cdot)$ denotes the Hamming weight.

Definition (s-XOR Count)

A binary non-singular matrix $M^{n \times n}$, can be implemented by a sequence of in-place XOR operations of the form: $x_i \leftarrow x_i \oplus x_j$ for $0 \leq i, j \leq n - 1$. The s-XOR count is defined as the minimum number of XOR operations of this form.

Notions of XOR Count

Definition (b_ϵ-XOR Count)

Given a cost vector $c = [c_0, c_1, \ldots, c_\epsilon]$ where $\epsilon \geq 1$ and $c_i \geq 0\ \forall i$, the b_ϵ-XOR count of the matrix $M^{m \times n}$ over $\mathbb{F}_2$ is defined as $\min\left(c_0 e_0 + c_1 e_1 + \cdots + c_\epsilon e_\epsilon\right)$, given M can be expressed by using equations of the following types in any order:

$$
\begin{aligned}
t_i &= t_{j_0} && \}\ e_0 \text{ times}, \\
t_i &= t_{j_0} \oplus t_{j_1} && \}\ e_1 \text{ times}, \\
t_i &= t_{j_0} \oplus t_{j_1} \oplus t_{j_2} && \}\ e_2 \text{ times}, \\
&\vdots \\
t_i &= t_{j_0} \oplus t_{j_1} \oplus t_{j_2} \oplus \cdots \oplus t_{j_\epsilon} && \}\ e_\epsilon \text{ times}.
\end{aligned}
$$

Notions of XOR Count

Example

Consider the binary matrix, $M^{5\times5} = \begin{array}{c c} & \begin{array}{ccccc} \mathrm{x}_0 & \mathrm{x}_1 & \mathrm{x}_2 & \mathrm{x}_3 & \mathrm{x}_4 \end{array} \\ \begin{array}{c} \mathrm{y}_0 \\ \mathrm{y}_1 \\ \mathrm{y}_2 \\ \mathrm{y}_3 \\ \mathrm{y}_4 \end{array} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$

d-XOR $= 6$	b_1-XOR $= 4$	s-XOR $= 5$
$\mathrm{y}_0 = x_0$ $\mathrm{y}_1 = x_1$ $\mathrm{y}_2 = x_0 + x_1 + x_2$ $\mathrm{y}_3 = x_0 + x_1 + x_3$ $\mathrm{y}_4 = x_0 + x_1 + x_4$	$\mathrm{y}_0 = x_0$ $\mathrm{y}_1 = x_1$ $\mathrm{t}_0 = x_0 + x_1$ $\mathrm{y}_2 = x_2 + t_0$ $\mathrm{y}_3 = x_3 + t_0$ $\mathrm{y}_4 = x_4 + t_0$	$\mathrm{x}_1 = x_1 + x_0$ $\mathrm{x}_3 = x_3 + x_1 \ (y_3)$ $\mathrm{x}_4 = x_4 + x_1 \ (y_4)$ $\mathrm{x}_2 = x_2 + x_1 \ (y_2)$ $\mathrm{x}_1 = x_1 + x_0 \ (y_1)$

Straight Linear Program (SLP) and Depth

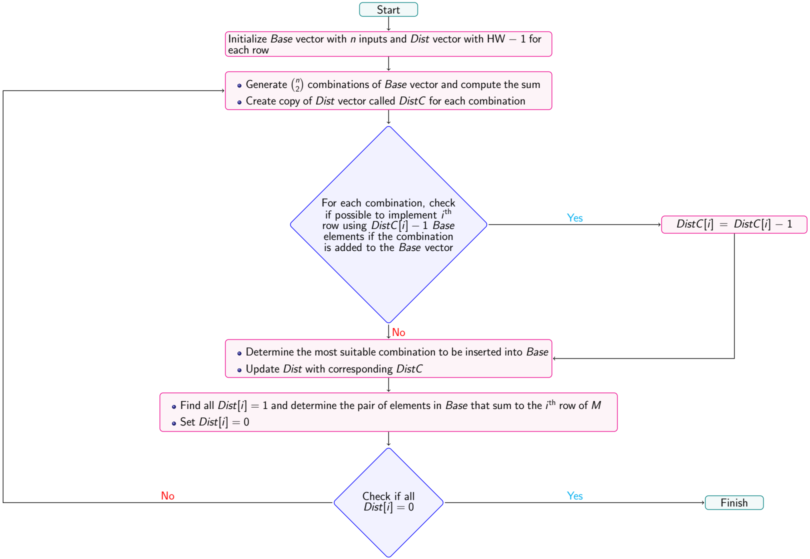
Definition (Straight Linear Program (SLP))

The implementation of the linear circuits is shown as a sequence of operations where every step is of the form: $u \leftarrow \bigoplus_{i=1}^{\epsilon} \lambda_i v_i$ where $\lambda_i \in \{0, 1\}$ $\forall i$ are constants and rest are variables.

Definition (Depth)

The depth of a logical circuit is defined as the number of combinational logic gates along the longest path of the circuit. The input variables are at depth 0; and for an SLP, depth can be computed as the maximum of depths for the variables in RHS plus 1.

Boyar-Peralta's Algorithm



Boyar-Peralta's Algorithm

Example

Consider the binary matrix, $M^{5\times 5} = \begin{array}{c c} & \begin{array}{ccccc} x_0 & x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{array}{c} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{array} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$

The output of the Boyar-Peralta's Algorithm in SLP form for M is:

```
y0 = x0
y1 = x1
t0 = x0 + x1
y2 = x2 + t0
y3 = x3 + t0
y4 = x4 + t0
```

Logic Libraries

Table 1: Logic libraries with gates and corresponding cost

Gate Library	Gate Count (GC)	STM 90nm (ASIC1)	STM 65nm (ASIC2)	TSMC 65nm (ASIC3)	STM 130nm (ASIC4)
XOR2	1	2.00 GE	1.981 GE	2.50 GE	3.33 GE
XOR3	1	3.25 GE	3.715 GE	4.20 GE	4.66 GE

XOR3 Support

Banik et al. (IWSEC 2019)

- Look for an instance where a t variable has the fan-out of 1 in the SLPs returned by the Boyar-Peralta's algorithm. For example:

```
t4 = x0 + x6 // t4 has fan-out of 1
t20 = x1 + t4 // t20 is the only variable that uses t4
```

- Replace with XOR3 operation:

```
// t4 = x0 + x6 (omitted)
t20 = x1 + x0 + x6 // t4 is substituted, t20 uses an XOR3 operation
```

- XOR3 operation is introduced by omitting one SLP where LHS has fan-out of 1. Repeat for all such variables.
- Disregards the relative cost for XOR3.

XOR3 Support

Ours

1. Generate all $\binom{n}{2}$ pairs and $\binom{n}{3}$ triplets of the *Base* vector elements, compute the XOR2 and XOR3 respectively, and assign the corresponding cost from the cost vector.
2. For each of the XOR2 combinations, determine whether it is possible to reduce $DistC[i]$ by 1. Similarly, for each of the XOR3 combinations, first check it is possible to reduce $DistC[i]$ by 2; if it is not, then check if $DistC[i]$ can be reduced by 1.
3. Determine the most suitable combination to be included to the *Base* vector based on heuristic.
4. If $Dist[i] = 1$ or $Dist[i] = 2$, then the i row of M can be implemented by adding two or three elements of the *Base* vector respectively. Check every pair/triplet of the *Base* vector to determine the elements which sum to $M[i]$. Once found, set $Dist[i]$ to 0, and include $M[i]$ to the *Base* vector.
5. Repeat until $Dist[i] = 0$ for all i.

Results: AES MixColumn Implementations

Table 2: Summary of recent AES MixColumn implementations

	Representation	# XOR2	# XOR3	Depth	GC
Banik, Funabiki, Isobe [Banik et al., 2019] (Available within this work)	b_1	95	–	6	95
	b_2	39	28	6	67
Tan, Peyrin [Tan and Peyrin, 2019]	b_1	94	–	9	94
Maximov [Maximov, 2019]	b_1	92	–	6	92
Xiang, Zeng, Lin, Bao, Zhang [Xiang et al., 2020]	s	92	–	6	92
Lin, Xiang, Zeng, Zhang [Lin et al., 2021]	b_1	91	–	7	91
Exclusively in this work	b_2	12	47	4	59

Results: 16×16 matrices

Table 3: Implementations of few 16×16 matrices in b_2

Matrix	GC	ASIC1	ASIC2	ASIC3	ASIC4
JOLTIK-BC	28 (6, 22)	83.0 (9, 20)	91.14 (16, 16)	106.5 (9, 20)	122.50 (6, 22)
MIDORI	16 (0, 16)	45.0 (16, 4)	46.56 (16, 4)	56.8 (16, 4)	71.92 (16, 4)
PRINCE M_0, M_1	16 (0, 16)	45.0 (16, 4)	46.56 (16, 4)	56.8 (16, 4)	71.92 (16, 4)
PRIDE $L_0 - L_3$	16 (0, 16)	45.0 (16, 4)	46.56 (16, 4)	56.8 (16, 4)	71.92 (16, 4)
QARMA-64	16 (0, 16)	45.0 (16, 4)	46.56 (16, 4)	56.8 (16, 4)	71.92 (16, 4)
SMALLSCALE-AES	24 (0, 24)	78.0 (0, 24)	85.93 (19, 13)	100.8 (0, 24)	111.84 (0, 24)

Number of (XOR2, XOR3) gates are given within parenthesis

Results: 32×32 matrices

Table 4: Implementations of few 32×32 matrices in b_2

Matrix	GC	ASIC1	ASIC2	ASIC3	ASIC4
AES	59 (12, 47)	169.0 (39, 28)	181.28 (39, 28)	215.1 (39, 28)	258.98 (12, 47)
ANUBIS	62 (11, 51)	185.0 (60, 20)	193.16 (60, 20)	234.0 (60, 20)	274.29 (11, 51)
CLEFIA M_0	62 (13, 49)	185.0 (60, 20)	193.16 (60, 20)	234.0 (60, 20)	271.63 (13, 49)
CLEFIA M_1	65 (3, 62)	193.0 (38, 36)	209.00 (38, 36)	246.2 (38, 36)	294.30 (38, 36)
TWOFISH	73 (17, 56)	215.5 (20, 54)	240.23 (20, 54)	276.8 (20, 54)	317.57 (17, 56)

Number of (XOR2, XOR3) gates are given within parenthesis

Results: AES MixColumn Graphical Form

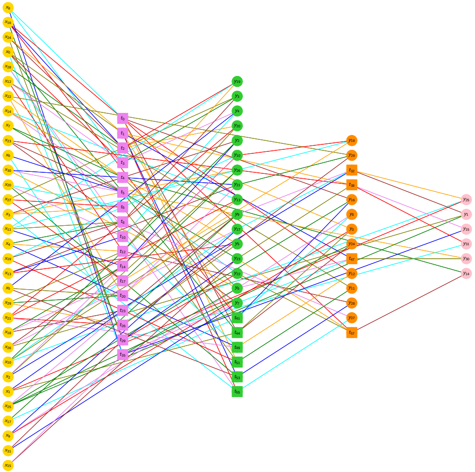


Figure 1: AES linear layer (MixColumn) in b_2 with 59 GC/depth 4 in graphical form

Conclusion and Future Works

- Our work achieves the best known results in terms of a logic library comprising of {XOR2, XOR3} gates, several of which are reported for the first time. E.g. `AES` `MixColumn` matrix with 59 gate count/4 depth/258.98 GE in STM 130nm process (ASIC4).
- We may consider the XNOR gates in the library and higher input XOR gates (XOR4 and beyond)
- Reversible implementation, depth optimization, and cost for the inverse matrices is an interesting direction of study.

Thank You!

https://bitbucket.org/vdasu_edu/boyar-peralta-xor3/

References I

[Banik et al., 2019] Banik, S., Funabiki, Y., and Isobe, T. (2019).
More results on shortest linear programs.
Cryptology ePrint Archive, Report 2019/856.
https://eprint.iacr.org/2019/856.

[Boyar and Peralta, 2010] Boyar, J. and Peralta, R. (2010).
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.

[Lin et al., 2021] Lin, D., Xiang, Z., Zeng, X., and Zhang, S. (2021).
A framework to optimize implementations of matrices.
In Paterson, K. G., editor, *Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings*, volume 12704 of *Lecture Notes in Computer Science*, pages 609–632. Springer.

[Maximov, 2019] Maximov, A. (2019).
Aes mixcolumn with 92 xor gates.
Cryptology ePrint Archive, Report 2019/833.
https://eprint.iacr.org/2019/833.

[Tan and Peyrin, 2019] Tan, Q. Q. and Peyrin, T. (2019).
Improved heuristics for short linear programs.
Cryptology ePrint Archive, Report 2019/847.
https://eprint.iacr.org/2019/847.

[Xiang et al., 2020] Xiang, Z., Zeng, X., Lin, D., Bao, Z., and Zhang, S. (2020).
Optimizing implementations of linear layers.
Cryptology ePrint Archive, Report 2020/903.
https://eprint.iacr.org/2020/903.

Working Example of Boyar-Peralta's Algorithm I

Consider the binary matrix, $M^{5\times5} = \begin{array}{c} \\ y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{array} \begin{array}{c} \begin{array}{ccccc} x_0 & x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$

1. Initial situation:
   - $Base = [x_0, x_1, x_2, x_3, x_4]$
   - $Dist = [0, 0, 2, 2, 2]$
   - SLP:

```
y0 = x0
y1 = x1
```

Working Example of Boyar-Peralta's Algorithm II

2. Iteration 1:
    - Pick all $\binom{n}{2}$ combinations of *Base* vector and compute the corresponding *DistC* vectors.
    - Ideal candidate: $t_0 = x_0 \oplus x_1$
    - $Base = [x_0, x_1, x_2, x_3, x_4, x_0 \oplus x_1]$
    - $Dist = [0, 0, 1, 1, 1]$
    - SLP:

```
y0 = x0
y1 = x1
t0 = x0 + x1
```

Working Example of Boyar-Peralta's Algorithm III

Table 5: Exemplary execution of Boyar-Peralta's Algorithm

Candidate *Base* Elements	$DistC$	$\lVert DistC \rVert_1$	$\lVert DistC \rVert_2^2$
$t_0 = x_0 \oplus x_1$	$[0, 0, 1, 1, 1]$	3	3
$t_1 = x_0 \oplus x_2$	$[0, 0, 1, 2, 2]$	5	9
$t_2 = x_0 \oplus x_3$	$[0, 0, 2, 1, 2]$	5	9
$t_3 = x_0 \oplus x_4$	$[0, 0, 2, 2, 1]$	5	9
$t_4 = x_1 \oplus x_2$	$[0, 0, 1, 2, 2]$	5	9
$t_5 = x_1 \oplus x_3$	$[0, 0, 2, 1, 2]$	5	9
$t_6 = x_1 \oplus x_4$	$[0, 0, 2, 2, 1]$	5	9
$t_7 = x_2 \oplus x_3$	$[0, 0, 2, 2, 2]$	6	12
$t_8 = x_2 \oplus x_4$	$[0, 0, 2, 2, 2]$	6	12
$t_9 = x_3 \oplus x_4$	$[0, 0, 2, 2, 2]$	6	12

Working Example of Boyar-Peralta's Algorithm IV

3. Iteration 2:
    - Since $Dist[2] = 1$, y_2 is implemented
    - $Base = [x_0, x_1, x_2, x_3, x_4, x_0 \oplus x_1, x_0 \oplus x_1 \oplus x_2]$
    - $Dist = [0, 0, 0, 1, 1]$
    - SLP:

```
y0 = x0
y1 = x1
t0 = x0 + x1
y2 = x2 + t0
```

Working Example of Boyar-Peralta's Algorithm V

4. Iteration 3:
   - Since $Dist[3] = 1$, y_3 is implemented
   - $Base = [x_0, x_1, x_2, x_3, x_4, x_0 \oplus x_1, x_0 \oplus x_1 \oplus x_2, x_0 \oplus x_1 \oplus x_3]$
   - $Dist = [0, 0, 0, 0, 1]$
   - SLP:

```
y0 = x0
y1 = x1
t0 = x0 + x1
y2 = x2 + t0
y3 = x3 + t0
```

Working Example of Boyar-Peralta's Algorithm VI

5. Iteration 4:
   - Since $Dist[4] = 1$, y_4 is implemented
   - $Base = [x_0, x_1, x_2, x_3, x_4, x_0 \oplus x_1, x_0 \oplus x_1 \oplus x_2, x_0 \oplus x_1 \oplus x_3, x_0 \oplus x_1 \oplus x_4]$
   - $Dist = [0, 0, 0, 0, 0]$, so the algorithm terminates after this step
   - SLP:

```
y0 = x0
y1 = x1
t0 = x0 + x1
y2 = x2 + t0
y3 = x3 + t0
y4 = x4 + t0
```

AES MixColumn operation in $GF(2^8)$

The AES MixColumn operation is over a 4×4 matrix in $GF(2^8)$:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 2 \cdot w_0 \oplus 3 \cdot w_1 \oplus w_2 \oplus w_3 \\ w_0 \oplus 2 \cdot w_1 \oplus 3 \cdot w_2 \oplus w_3 \\ w_0 \oplus w_1 \oplus 2 \cdot w_2 \oplus 3 \cdot w_3 \\ 3 \cdot w_0 \oplus w_1 \oplus w_2 \oplus 2 \cdot w_3 \end{bmatrix}, w_i \in GF(2^8)$$

Slide courtesy of Quan-Quan Tan [Tan and Peyrin, 2019] (https://iacr.org/submit/files/slides/2019/tches/ches2020/29960/slides.pdf)

From $GF(2^n)$ to $GF(2)$

Multiplication by a fixed element in $GF(2^n)$ can be replaced by a $n \times n$ binary matrix multiplication.

$w_0 = x_7x_6x_5x_4x_4x_2x_1x_0$

irreducible polynomial $= p^8 + p^3 + p^3 + p + 1$

$$3 \times w_0 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix}$$

Slide courtesy of Quan-Quan Tan [Tan and Peyrin, 2019] (https://iacr.org/submit/files/slides/2019/tches/ches2020/29960/slides.pdf)