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

Haotian Shi[1, 2] Xiutao Feng[1] Shengyuan Xu[3]

[1]Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China

[2]University of Chinese Academy of Sciences, Beijing, China

[3]Department of Fundamental Courses, Shandong University of Science and Technology, Taian, 271019, China

2024.3.28

中国科学院大学
University of Chinese Academy of Sciences

Table of Contents

Outline

Lightweight Cryptography

- Application scenarios.
  - Internet of Things (IoTs), wireless sensor networks.
  - Other devices with limited resource.
- Goals.
  - Low resource cost in terms of **area**, power comsuption and **latency**.
- Research directions.
  - Designing new ciphers with lightweight building blocks.
    Constructing lightweight Maximum Distance Separable (MDS) matrices.
  - **Optimizing** the implementation of **linear** and non-linear layers of existing ciphers.

Implementation of a linear layer

- The linear layer: a linear Boolean function f.

$$f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$$
$$\boldsymbol{x}^T \mapsto \boldsymbol{y}^T = A\boldsymbol{x}^T$$

, where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$, $\boldsymbol{y} = (y_0, y_1, \ldots, y_{m-1})$ and $A = (a_{ij})_{m\times n}$.

- A "node" $t = (t_0, t_1, \ldots, t_{n-1})$ defines an intermediate value

$$t = t_0x_0 \oplus t_1x_1 \oplus \cdots \oplus t_{n-1}x_{n-1}.$$

Definition 1 (Implementation of a linear layer)

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

- A trivial implementation: $\sum_{i=0}^{m-1}(wt(y_i) - 1)$ XOR gates. Nodes can be reused.
- Area: XOR gate count.

An example

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

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

Table 1: Two implementations of A

A trivial implementation needs 6 XOR gates.

Implementation graph

- Nodes: the nodes in the implementation.
- Edges: node x_j, x_k points to node x_i, if x_i is generated by x_j, x_k.

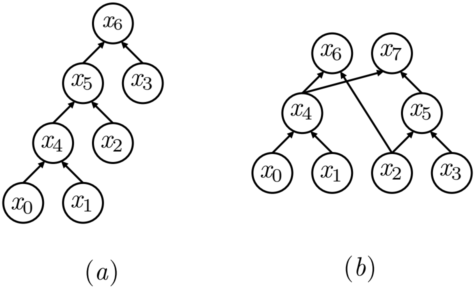


Figure 1: The implementation graphs of A's two implementations $(a), (b)$.

Depth

Definition 2 (Depth)

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

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

- $d(t_1) = \max\{d(t_2), d(t_3)\} + 1$, if t_1 is generated by t_2, t_3.
- Latency: closely related to the depth of implementations.

Minimum depth

- $d_{min}(t)$: the minimum depth of node t that t can reach.

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

- $d_{min}(A)$: the minimum depth of A that all A's implementations can achieve.

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

Definition 3

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

The SLP and SLPD problem

Definition 4

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

Definition 5

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

- The SLP problem over $\mathbb{F}_2$ has been proven to be NP-complete.
- This paper focuses on the SLPD problem.

Outline

State-of-art heuristics

- Forward search. Variants of the BP algorithm.
- Backward search.
- Local re-optimization algorithm. Re-optimize a subcircuit.

The BP algorithm

- Two key parameters: the base set $\mathcal{B}$ and the distance vector $Dist_{\mathcal{B}}$.
  $Dist_{\mathcal{B}}[i] = \min\{d \mid y_i = \bigoplus_{t=1}^{d} \mathcal{B}[i_t]\}$

- Main iterative step:
  - Select a new base element generated by two nodes in $\mathcal{B}$ and add it to $\mathcal{B}$.
  - Update the distance vector $Dist$.

- BP's strategy:
  - Cost function to minimize: $\sum_{i=0}^{m-1} Dist[i]$.
  - Tie-breaker: maximizing $\sum_{i=0}^{m-1} Dist[i] \cdot Dist[i]$.
    ((2, 2, 2, 2) is worse than (1, 1, 4, 2).)
  - Pre-emptive strategy: if $\mathcal{B}[i] \oplus \mathcal{B}[j]$ equals an output node t, t is added to $\mathcal{B}$.

Innovation points on some variants of the BP algorithm

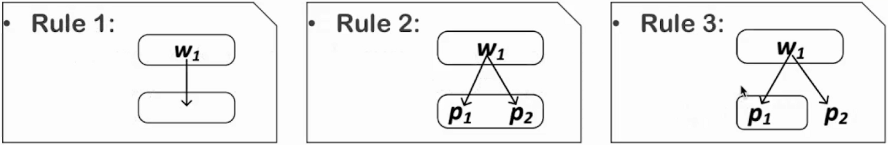
- The RNBP algorithm: every tie-breaking choice is equally possible.
- The A1 algorithm: at least one $Dist[j]$ which equals to $\min_{i, Dist[i]>1} Dist[i]$ must be reduced.
- The LSL algorithm: depth limit on base elements and $Dist$.
- The BFI algorithm: focusing on PAQ, where P, Q are permutation matrices.

Backward search

- Initiate the search from output nodes by determining how a given node w is split, i.e, $w = p \oplus p'$, until all nodes are split into input nodes.
- Parameters:
  - A working set $\mathcal{W}$.
  - A predecessor set $\mathcal{P}$.
  - A parameter s indicates the depth of elements in $\mathcal{W}$.
- Goal: maximize the reuse of predecessor nodes in $\mathcal{P}$.
- Heuristic: randomly choose one splitting operation which satisfies the rule with the highest priority.

Rules

- The authors developed five priority-based rules for splitting nodes within $\mathcal{W}$:



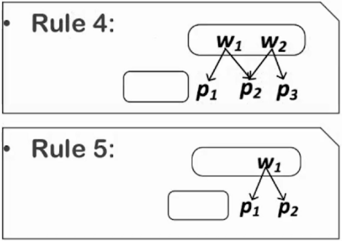


Table 4: The costs of predecessors

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

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

(The picture was used in their slide.)

Outline

Improved forward search: IBPD and IBPD-MD

- Observation: about the depth limit. Outputs and those with high depth have a smaller contribution to other outputs due to the depth limit, but this is not the case in the SLP problem. Consequently, prioritizing closer outputs has a smaller impact on approaching other outputs and may result in the loss of alternative, better pathways that generate the closer outputs.

- **Our strategy**: A new tie-breaking rule of **minimizing** new $\sum_{i=0}^{m-1} Dist[i] \cdot Dist[i]$ instead of maximizing it.
  (Reduce all $Dist[i]$'s at a relatively consistent pace.)

- New improved heuristics:
  - LSL + RNBP + **our strategy** $\longrightarrow$ **IBPD**.
  - **IBPD** + A1 $\longrightarrow$ **IBPD-MD**.

- Difference between IBPD with IBPD-MD.
  - IBPD: better suitable for a strict depth limit for all output nodes.
  - IBPD-MD: better suitable for a bit looser depth limit for some output nodes.

A new framework of combining forward search with backward search (BPBS)

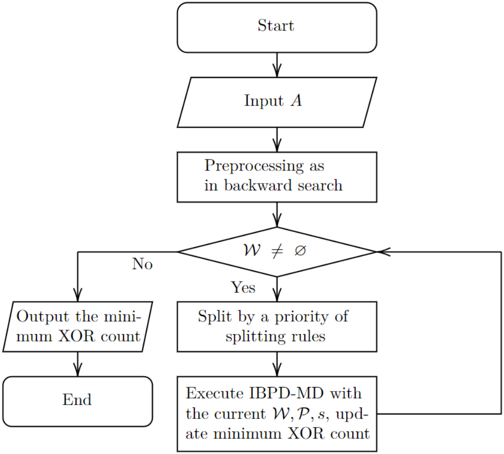


Figure 1: Framework of integrating IBPD-MD with backward search.

Remarks on the new framework

- A combination of forward search with backward search.
- Adjusting the search space of IBPD-MD is helpful for IBPD-MD to jump out of local minima.
- The IBPD-MD version works better than the IBPD version.
- A modified priority of rules: combination of **Rule 3** and **Rule 5**.
- A relaxed depth bound.

Outline

Application to AES MixColumns

Table 3: XOR/depth costs of AES MixColumns

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

Application to many proposed matrices

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

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

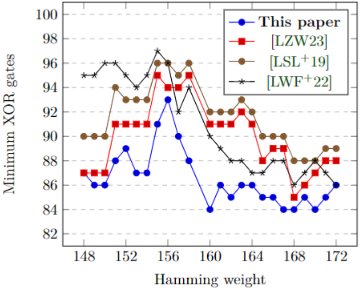
[a] The result can only be searched by BPBS.
[b] The result can be searched by IBPD.
[c] The result can be searched by IBPD-MD.

Application to many involutory MDS matrices

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

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

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



Thank you!