Optimizing Implementations of Boolean Functions

Meltem Sönmez Turan

National Institute of Standards and Technology

Abstract

Symmetric cryptography primitives are constructed by iterative applications of linear and nonlinear layers. Constructing efficient circuits for these layers, even for the linear one, is challenging. In 1997, Paar proposed a heuristic to minimize the number of XORs (modulo 2 addition) necessary to implement linear layers. In this study, we slightly modify Paar's heuristics to find implementations for nonlinear Boolean functions, in particular to homogeneous Boolean functions. Additionally, we show how this heuristic can be used to construct circuits for generic Boolean functions with small number of `AND` gates, by exploiting affine equivalence relations.

1 Introduction

Symmetric cryptography primitives are constructed by iterative applications of linear and nonlinear layers. Linear layers are typically composed of binary matrices, and are used for *diffusion*, whereas the nonlinear layers are composed of nonlinear substitution boxes (s-box), and are used for *confusion*. Constructing efficient circuits for these layers, even for the linear ones, is challenging. There are various metrics to measure the efficiency of the circuits such as number of specific gates (e.g., `AND`, `XOR`), or the depth of the circuits.

Multiplicative Complexity. The metric *Multiplicative Complexity* (MC) is defined as the minimum number of `AND` gates required to implement a function with a circuit over the basis {`AND, XOR, NOT`}. This complexity measure is relevant for many advanced cryptographic protocols (e.g., [1]), fully homomorphic encryption (e.g., [2]), and zero-knowledge proofs (e.g., [3]), where processing nonlinear gates such as `AND`, `NAND`, is more expensive than processing linear gates such as `XOR`. These protocols benefit from new symmetric-key primitives that can be implemented with small number of `AND` gates (e.g., Rasta [4], LowMC [5]).

There is no known asymptotically efficient technique to compute the MC of a random Boolean function. In 2000, Boyar et al. [6] showed that the MC of an n-variable random Boolean function is at least $2^{n/2} - \mathcal{O}(n)$ with high probability. For arbitrary n, it is known that under standard cryptographic assumptions, it is not possible to compute the MC in polynomial time in the length of the truth table [7]. The *degree bound* states that the MC of a Boolean function having degree d is at least $d - 1$ [8].

Although there are no efficient techniques to find MC of for random Boolean functions, the MC distribution has been established for Boolean functions having up to 6 variables [9, 10]. There are also known techniques specific for Boolean functions with low degree (e.g., less then or equal to three) or structure (e.g., symmetric). The MC of affine Boolean functions is zero. Mirwald and Schnorr [11] showed that the MC of a quadratic function f is k, iff f is affine equivalent to the canonical form $\bigoplus_{i=1}^{k} x_{2i-1}x_{2i}$. This implies the MC of quadratic functions is at most $\lfloor \frac{n}{2} \rfloor$. Turan and Peralta [12] improved the bounds on

MC of cubic Boolean functions. Brandão et al. [13] studied the MC of symmetric Boolean functions and constructed circuits for all such functions with up to 25 variables. In 2017, Find et al. [14] characterized the Boolean functions with MC 2 by using the fact that MC is invariant with respect to affine transformations. In 2020, Çalık et al. extended the result to Boolean functions with MC up to 4 [15]. In 2022, Häner and Soeken [16] showed the MC of interval checking.

XOR complexity. In addition to the optimization of `AND` gates for Boolean function, another line of research focuses on optimizing the implementations of linear matrices over $\mathbb{F}_2$, where the goal is to minimize the number of `XOR` gates necessary to implement the matrices. There are three metrics used while optimizing the number of `XOR` gates: direct `XOR` (`d-XOR`), sequential `XOR` (`s-XOR`) and general `XOR` (`g-XOR`). `d-XOR` is the direct `XOR` count and corresponds to the the number of 1's in the binary matrix representation of the linear layer. The `s-XOR` metric counts the number of `XOR` operations of the form $x_i = x_i \oplus x_j$, that updates the value of input x_i, whereas, `g-XOR` metric corresponds to the number of operations of the form $x_i = x_j \oplus x_k$. Determining optimal implementations for `s-XOR` and `g-XOR` is a hard problem. Boyar et al. [17] argue that minimizing the number of XORs to compute a binary matrix is equivalent to solving the Shortest Linear Program problem over GF(2), which is known to be NP-hard. One of the early heuristics for XOR optimization is by Paar[18] in 1997, which is cancellation-free, i.e., the circuits generated by Paar's heuristic does not include cancellation of identical input bits. Since ability to cancellation leads to better circuit, many new heuristics were suggested (e.g., [19, 20, 21]).

Contributions. In this study, we propose a modification to Paar's heuristics so that it can also be applied to nonlinear functions, in particular to homogeneous Boolean functions. Additionally, we show how this heuristic can be used to construct circuits for generic Boolean functions with small number of `AND` gates, by exploiting affine equivalance relations.

2 Preliminaries

2.1 Boolean functions

Let $\mathbb{F}_2$ be the finite field with two elements. An n-variable Boolean function f is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. Let B_n be the set of n-variable Boolean functions.The *algebraic normal form* (ANF) of f is the multivariate polynomial

$$f(x_1, \ldots, x_n) = \sum_{u \in \mathbb{F}_2^n} a_u x^u, \tag{1}$$

where $a_u \in \mathbb{F}_2$ and $x^u = x_1^{u_1} x_2^{u_2} \cdots x_n^{u_n}$ is a *monomial* containing the variables x_i where $u_i = 1$. The degree of the monomial x^u is the number of variables appearing in x^u. The *algebraic degree* of a Boolean function, denoted $deg(f)$, is the highest degree among the monomials appearing in its ANF. A Boolean function is called *homogeneous*, if all the monomials in its algebraic normal form have the same algebraic degree.

Two functions $f, g \in \mathcal{B}_n$ are *affine equivalent* if f can be written as

$$f(\mathbf{x}) = g(A\mathbf{x} + \mathbf{a}) + \mathbf{b}^\top \mathbf{x} + c, \tag{2}$$

where A is a non-singular $n \times n$ matrix over $\mathbb{F}_2$, $\mathbf{a}, \mathbf{b}$ are column vectors in $\mathbb{F}_2^n$, and $c \in \mathbb{F}_2$. We use $[f]$ to denote the affine equivalence class of the function f. Degree and MC are invariant under affine transformations.

2.2 Boolean Circuits

A *Boolean circuit* C with n inputs and m outputs is a directed acyclic graph, where the inputs and the gates are the nodes, and the edges correspond to the Boolean-valued *wires*. The *fanin* and *fanout* of a node is the number of wires going in and out of the node, respectively. The nodes with fanin zero are called the *input nodes* and are labeled with an input variable from $\{x_0, \ldots, x_{n-1}\}$. The circuits considered in this study only contain gates from the complete basis {`AND, XOR, NOT`} and have exactly one node with fanout zero (i.e., $m = 1$), which is called the *output node*. For our purposes, we assume `AND` gates have fan-in two, but `XOR` gates have arbitrary fan-in (i.e., > 0).

2.3 Paar's Heuristics

The linear layers of symmetric key primitives can be represented by a $m \times n$ binary matrix M, where there are n input variables $(x_0, \ldots, x_{n-1})$ and m output variables $(y_0, \ldots, y_{m-1})$. An upper bound for the number of `XOR` operations is $w - m$, where w is the *weight* of M (i.e., the number of ones).

Paar [18] proposed two heuristics to implement linear layers with small number of `XOR` operations. Both heuristics operate on the matrix representation of the linear layer. The heuristic determines the frequency for each possible pairs of input variable x_i, x_j $(i \neq j)$ that are XORed together in m linear functions. The pair with highest frequency is computed and placed to the matrix as a new variable. In the next iteration, the operation is repeated on the matrix of size $m \times (n+1)$. This procedure is repeated until all outputs have been computed (i.e., the weight of the resulting matrix is m).

Example. Let the linear layer to implement be given as follows:

$$\begin{aligned} x_0 + x_1 + x_2 &= y_0 \\ x_1 + x_3 + x_4 &= y_1 \\ x_0 + x_2 + x_3 + x_4 &= y_2 \\ x_1 + x_2 + x_3 &= y_3 \\ x_0 + x_1 + x_3 &= y_4 \\ x_1 + x_2 + x_3 + x_4 &= y_5 \end{aligned}$$

The matrix representation of the linear layer is

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}$$

Frequency of each pair of inputs appearing in the linear layer is

Pair	Frequency	Pair	Frequency
(x_0, x_1)	2	(x_1, x_3)	4
(x_0, x_2)	2	(x_1, x_4)	2
(x_0, x_3)	2	(x_2, x_3)	3
(x_0, x_4)	1	(x_2, x_4)	2
(x_1, x_2)	3	(x_3, x_4)	3

The first selected pair is (x_1, x_3) with frequency 4. So, the first step of the implementation is $t_0 = x_1 \oplus x_3$. Then the matrix is updated as follows.

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

and the updated frequency table is

Pair	Frequency	Pair	Frequency
(x_0, x_1)	1	(x_1, t_0)	0
(x_0, x_2)	2	(x_2, x_3)	1
(x_0, x_3)	1	(x_2, x_4)	2
(x_0, x_4)	1	(x_2, t_0)	2
(x_0, t_0)	1	(x_3, x_4)	1
(x_1, x_2)	1	(x_3, t_0)	0
(x_1, x_3)	0	(x_4, t_0)	2
(x_1, x_4)	0	-	-

There is a tie for the the pairs (x_0, x_2), (x_2, x_4), (x_2, t_0), and (x_4, t_0). For this example the next pair is selected randomly among these pairs as (x_0, x_2), and the next step of the implementation becomes $t_1 = x_0 \oplus x_2$. Continuing this way, the implementation of the layer is found as:

$$\begin{aligned} t_0 &= x_1 \oplus x_3 \\ t_1 &= x_0 \oplus x_2 \\ t_2 &= x_4 \oplus t_0 \\ t_3 &= x_1 \oplus t_1 \\ t_4 &= x_3 \oplus x_4 \\ t_5 &= t_1 \oplus t_4 \\ t_6 &= x_2 \oplus t_0 \\ t_7 &= x_0 \oplus t_0 \\ t_8 &= x_2 \oplus t_2 \end{aligned}$$

The output $(y_0, y_1, y_2, y_3, y_4, y_5)$ is obtained as $(t_3, t_2, t_5, t_6, t_7, t_8)$.

3 Application of Paar's Heuristic to Nonlinear Boolean Functions

Although Paar's heuristic is proposed to find implementations for linear layers, it can also be applied to nonlinear Boolean functions, with a slight modification. An n-variable Boolean function with m monomials can be represented by a $m \times n$ binary matrix, where each row corresponds to a monomial in the ANF of the function. For example, the following row (1 1 0 1 0 1) represents the monomial $x_0x_1x_3x_5$ for a 6-variable Boolean function. Instead of modulo 2 addition of each terms in the row, we are now interested in modulo 2 multiplication of each term. This method, in general, would not be efficient (in

terms of number of multiplications), especially for Boolean functions with large number of monomials, as the heuristic computes each monomials independently.

Next we propose a variation of the heuristic that decomposes Boolean functions into homogeneous Boolean functions and exploit affine equivalence relations to find efficient circuits.

Let $f \in B_n$, with degree d. The proposed heuristic to find efficient circuit for f is as follows:

1. Decompose f into d homogeneous Boolean functions,

$$f = a + f_1 \oplus f_2 \oplus \ldots \oplus f_d,$$

   where f_i is the sum of monomials of f with degree i, and a corresponds to the constant term.

2. Apply a number of affine equivalence transformations to the highest-degree homogeneous function, (i.e., f_d) to construct f'_d with smaller number of monomials with degree d. Note that if $d = n$, no affine transformation would decrease the number of monomials, as there is only one monomial with degree n. If f'_d includes monomials with degree smaller than d, those monomials are added to the corresponding f_i depending on their degree.

3. Apply modified Paar's heuristic to find an implementation for the degree d terms of f'_d. (Note that in modified Paar's heuristic each iteration corresponds to modulo 2 multiplication, instead of modulo 2 addition.) Apply the inverse affine transformation to the circuit to construct an implementation for the degree d monomials of f.

4. Repeat the procedure to find an implementation for f'_{d-1} where f'_{d-1} is the XOR of f_d and the new degree $d-1$ monomials generated during Step 2.

5. The procedure is repeated until implementations for each homogoeneous function is obtained and these sub-circuits are compined to find an implementation for f.

The combined implementations can further be improved by eliminating the common operations done in each independent implementations of the homogeneous functions.

4 Discussion

In this study, we proposed a modification of the Paar's heuristic to find efficient implementations for Boolean functions (in particular to reduce the number of nonlinear gates). In general, Paar's heuristic provides better solutions when the representation matrix has low weight, which may not be true for nonlinear Boolean functions. Decomposing the Boolean function into homogeneous Boolean functions, and applying affine transformations to the specific degree terms makes it easier to reduce the number of target monomials, since smaller degree terms are handled in the next iterations of the algorithm.

References

[1] Vladimir Kolesnikov and Thomas Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg,

Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[2] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325. ACM, 2012.

[3] Joan Boyar, Ivan Damgård, and René Peralta. Short Non-Interactive Cryptographic Proofs. *J. Cryptology*, 13(4):449–472, 2000.

[4] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A Cipher with Low ANDdepth and Few ANDs per Bit. In *CRYPTO (1)*, volume 10991 of *Lecture Notes in Computer Science*, pages 662–692. Springer, 2018.

[5] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015.

[6] Joan Boyar, René Peralta, and Denis Pochuev. On the Multiplicative Complexity of Boolean Functions over the Basis ($\wedge$, $\oplus$, 1). *Theor. Comput. Sci.*, 235(1):43–57, 2000.

[7] Magnus Gausdal Find. On the Complexity of Computing Two Nonlinearity Measures. In *Computer Science - Theory and Applications - 9th International Computer Science Symposium in Russia, CSR 2014, Moscow, Russia, June 7-11, 2014. Proceedings*, pages 167–175, 2014.

[8] C. P. Schnorr. The Multiplicative Complexity of Boolean Functions. In Teo Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC 1988)*, volume 357 of *LNCS*, pages 45–58, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[9] Meltem Turan Sönmez and René Peralta. *The Multiplicative Complexity of Boolean Functions on Four and Five Variables*, pages 21–33. Springer International Publishing, Cham, 2015.

[10] Çağdaş Çalık, Meltem Sönmez Turan, and René Peralta. The Multiplicative Complexity of 6-variable Boolean Functions. *Cryptogr. Commun.*, 11(1):93–107, 2019.

[11] Roland Mirwald and Claus-Peter Schnorr. The Multiplicative Complexity of Quadratic Boolean Forms. *Theor. Comput. Sci.*, 102(2):307–328, 1992.

[12] Meltem Sonmez Turan and Rene Peralta. On the Multiplicative Complexity of Cubic Boolean Functions. The 6th International Workshop on Boolean Functions and their Applications (BFA), 2021.

[13] Luís T. A. N. Brandão, Çağdaş Çalık, Meltem Sönmez Turan, and René Peralta. Upper Bounds on the Multiplicative Complexity of Symmetric Boolean Functions. *Cryptogr. Commun.*, 11(6):1339–1362, 2019.

[14] Magnus Gausdal Find, Daniel Smith-Tone, and Meltem Sönmez Turan. The Number of Boolean Functions with Multiplicative Complexity 2. *IJICoT*, 4(4):222–236, 2017.

[15] Çağdaş Çalık, Meltem Sönmez Turan, and René Peralta. Boolean Functions with Multiplicative Complexity 3 and 4. *Cryptogr. Commun.*, 12(5):935–946, 2020.

[16] Thomas Häner and Mathias Soeken. The multiplicative complexity of interval checking, 2022.

[17] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *J. Cryptol.*, 26(2):280–312, 2013.

[18] C. Paar. Optimized arithmetic for Reed-Solomon encoders. *Proceedings of IEEE International Symposium on Information Theory*, pages 250–, 1997.

[19] Alexander Maximov and Patrik Ekdahl. New circuit minimization techniques for smaller and faster AES sboxes. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):91–125, 2019.

[20] Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. More results on shortest linear programs. In Nuttapong Attrapadung and Takeshi Yagi, editors, *Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28-30, 2019, Proceedings*, volume 11689 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2019.

[21] Zejun Xiang, Xiangyoung Zeng, Da Lin, Zhenzhen Bao, and Shasha Zhang. Optimizing implementations of linear layers. *IACR Transactions on Symmetric Cryptology*, 2020(2):120–145, Jul. 2020.