Determining the Minimum Degree of an S-box

P. R. Mishra
SAG, DRDO, Delhi, INDIA
prasanna.r.mishra@gmail.com

Sumanta Sarkar
TCS Innovation Labs, Hyderabad, INDIA
sumanta.sarkar1@tcs.com

Indivar Gupta
SAG, DRDO, Delhi, INDIA
indivargupta@sag.drdo.in

Abstract

S-boxes are important building blocks in block ciphers. For secure design one should not choose an S-box that has low degree. In this work we consider *minimum degree* of an S-box which is the minimum value of the degree of the nonzero component functions of the S-box. For an S-box $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, there are $2^m - 1$ nonzero component functions, we show that there is a better way to determine the minimum degree of an S-box which does not require to check all the $2^m - 1$ component functions. To the best of our knowledge, this is the best algorithm for determining the minimum degree of an S-box in the literature.

keyword: Boolean function, S-box, degree, row echelon form, linear span.

1 Introduction

Suppose $\mathbb{F}_2^n$ be the vector space formed by 2^n binary n-tuples. An n-variable Boolean function is a mapping $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$. Boolean functions play an important role in the design of Linear Feedback Shift Registers based stream ciphers Boolean functions should maintain some cryptographic properties such as high nonlinearity, high correlation immunity, high degree, etc. in

order to withstand some cryptanalytic attacks. High nonlinearity resists the best affine approximation attack [4], high correlation immunity is required to resist the correlation attack [10]. There is another attack that uses Berlekamp-Massey algorithm [8] to find the linear complexity of the sequence generated by the stream cipher. If the Boolean function is of high degree, then it can increase the linear complexity and thus can raise the difficulty in mounting this particular attack.

An (n, m) S-box $F = (F_1, \ldots, F_m)$ is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$, where $F_1, \ldots, F_m$ are n-variable Boolean functions. S-boxes are one of the key parts in block ciphers (for example, that follow the design of substitution-permutation network) that are used to build the confusion layer. S-boxes are also used in constructing secure hash functions, for example Keccak [1]. For a secure design, an S-box should have some cryptographic properties. For example high nonlinearity of the S-box is required to thwart linear cryptanalysis [9], and high order resilient S-boxes are useful against the correlation attack like [10].

The S-box should have high algebraic degree and minimum degree; an S-box with low degree is susceptible to cryptanalytic attacks, namely algebraic attack, higher-order differential, interpolation, cube attacks etc. One may read [6] and [5] for details. Therefore, it is important to check the degree of an S-box that is chosen to be used in a cipher. In this paper, we design an efficient algorithm to determine the minimum degree of an S-box. Note that finding the algebraic degree of an S-box is easy as one just needs to check the maximum degree of the coordinate functions. However, challenge lies in determining the minimum degree for which one has to consider all the component functions.

In this paper we design an efficient algorithm to determine the minimum degree of an S-box.

1.1 Preliminaries

Suppose $u_0, \ldots, u_{2^n-1}$ are the distinct elements of $\mathbb{F}_2^n$. Then f can be represented by the binary string $[f(u_0), \ldots, f(u_{2^n-1})]$. This string is called the *truth table* of f. The *Hamming weight* of a binary string S is the number of 1's in S and it is denoted by $wt(S)$.

An n-variable Boolean function f can be written as a polynomial of $x_1, \ldots, x_n$ variables as follows,

$$f(x_1, x_2, \ldots, x_n) = \bigoplus_{a=(a_1,\ldots,a_n)\in\mathbb{F}_2^n} \left(\mu_a \prod_{i=1}^{n} x_i^{a_i} \right),$$

where $\mu_a = \bigoplus_{x \preceq a} f(x)$. This is called the *algebraic normal form (ANF)* of f. The algebraic degree [7], $\deg(f)$, of f is defined as

$$\max_{a \in \mathbb{F}_2^n} \{wt(a) | \mu_a \neq 0\}.$$

Given the truth table of an n-variable Boolean function f, the ANF can be obtained by Möbius transformation in $O(n2^n)$ steps [3]. For the sake of completeness we describe the algorithm below:

1. Write the truth table of f, in lexicographic order of input vectors.
2. Divide the truth table into two halves say f_0 and f_1.
3. $f_1 \leftarrow f_0 \oplus f_1$.
4. If length of f_0 is more than one bit, apply step 2 to the functions f_0 and f_1.

The algorithm ends when each of the functions f_0 and f_1 contains one bit in its truth table. The final table gives the values of the ANF of f.

Let A_f denote the ANF vector of f, i.e., for $a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$,

$$A_f[a] = \begin{cases} 1, \text{ if } x_1^{a_1} \ldots x_n^{a_n} \text{ is present in the ANF of } f, \\ 0, \text{ otherwise.} \end{cases}$$

An (n, m) S-box $F = (F_1, \ldots, F_m)$ is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ denoted as $F = (F_1, \ldots, F_m)$, where each F_i is an n-variable Boolean function. The functions $F_1, \ldots, F_m$ are called the *coordinate functions* of F. Let $\mathtt{LS}(F_1, \ldots, F_m)$ denote the linear span obtained from $\{F_1, \ldots, F_m\}$. We use $\mathtt{LS}^*(F_1, \ldots, F_m)$ to denote the set of functions that are nonzero linear combinations of $\{F_1, \ldots, F_m\}$. Then the functions that belong to $\mathtt{LS}^*(F_1, \ldots, F_m)$ are called the component functions of F. Clearly the number of component functions is $2^m - 1$. The *minimum degree* of the S-box F is the minimum among all the degrees of the component functions of F [3, Page9]. We denote the minimum degree of the S-box F by $\mathtt{MinDeg}(F)$. Then

$$\mathtt{MinDeg}(F) = \min_{f \in \mathtt{LS}^*(F_1, \ldots, F_m)} \{deg(f)\}.$$

On the other hand the maximum degree of the coordinate functions is defined as the algebraic degree of the S-box [3, Page 9]. In this paper, we are interested only in the degree which is the minimum of the component functions.

1.2 Our contribution

We propose an efficient algorithm (Algorithm 2) for checking the minimum degree of an S-box. The naive algorithm (describes as Algorithm 1)that follows the definition of minimum degree, has to consider $2^m - 1$ component functions and then check the degree of each function. To do it in a clever way we apply a property of row echelon matrix, and design an improved algorithm (Algorithm 2) that does not require to check all the $2^m - 1$ component functions. The complexity of our proposed algorithm is $O(mn + m^2)2^n)$ as opposed to $O(n2^{m+n})$ which is due to the naive algorithm. for an (n, m) S-box. Our proposed algorithm has asymptotically far better complexity than the naive one, precisely it drops the factor $n2^m$ down to $poly(n, m)$.

Algorithm	Complexity
Naive (Algorithm 1)	$O(n2^{m+n})$
Proposed (Algorithm 2)	$O((mn + m^2)2^n)$

Table 1: Comparison of complexities of the proposed algorithm and the naive algorithm for checking the minimum degree of an S-box

To the best of our knowledge, this is the most efficient algorithm for checking the minimum degree of an S-box.

2 Algorithm for determining the minimum degree of an S-box

In this section we present a very efficient algorithm for determining the minimum degree of an S-box. First we would like to present the naive algorithm that directly follows the definition.

Algorithm 1: Naive algorithm that calculates the minimum degree of an S-box F

Input: An (n, m) S-box $F = (F_1, \ldots, F_m)$.
Output: The minimum degree `MinDeg`(F).

1. Take all possible component functions and check their degree.
2. Return the minimum degree of the functions that belong to these linear combinations.

Let us now analyze the complexity of this algorithm.
In Step 1, given a component function, it takes $O(n2^n)$ steps to determine the degree of the function using Möbius transformation. As there are total $2^m - 1$ component function, the complexity becomes $O(n2^n(2^m - 1)) = O(n2^{m+n})$. Certainly when m and n grows this algorithm will perform worse.

We investigate how we can improve the naive algorithm so that the minimum degree of an S-box of higher dimensions can be determined efficiently. Below we present an improved algorithm for checking the minimum degree of an S-box.

2.1 The improved algorithm for checking the minimum degree of an S-box

First we present some results from linear algebra which are required to develop the algorithm.

Suppose M is a $t \times s$ matrix defined over real numbers. The *row echelon* form of M is another $t \times s$ matrix M' obtained by applying row operations on M such that M' has the following properties:

1. All the zero rows occur at the bottom of the matrix.
2. The first nonzero element of every nonzero row is 1, termed as the leading 1 of the corresponding row.
3. The position of the leading 1 in a row vector is strictly to the right of the leading 1's of the previous row vectors.

Note that the rank of the matrix M is the number of nonzero rows that are present in the matrix M'. Moreover, the nonzero row vectors of M' form a basis of the linear span of all the row vectors of the matrix M.

Suppose $pos(v)$ denotes the position of the leading 1 of the vector v. If v_i and v_k are the i-th and k-th row vectors, respectively of M', then $pos(v_k) < pos(v_i)$ for all $k < i$.

Proposition 1. *Suppose M is a $t \times s$ binary matrix with rank r, and the row vectors are $v_1, \ldots, v_t$, where v_i is the i-th row vector. Let M' be the row echelon form of M, where $u_1, \ldots, u_r$ are the first r vectors (which are also all the nonzero vectors) of M'. Let w be any vector in the linear span $\mathtt{LS}(v_1, \ldots, v_t)$ of $\{v_1, \ldots, v_t\}$. Then*

$$pos(w) \leq pos(u_r).$$

Proof. It is easy to check that $\texttt{LS}(v_1, \ldots, v_t) = \texttt{LS}(u_1, \ldots, u_r)$. Then the vector $w \in \texttt{LS}(v_1, \ldots, v_t)$, can be written as $w = c_1u_1 \oplus \ldots \oplus c_ru_r$, where each $c_i \in \{0, 1\}$. Let c_k be first nonzero coefficient. The vector u_k has leading 1 at $pos(u_k)$. As the $pos(u_k)$-th coordinates of all the vectors $u_{k+1}, \ldots, u_r$ are zero, therefore, the $pos(u_k)$-th coordinate of w is 1 which is also the leading 1 of w. Since $k \leq r$, therefore, $pos(w) \leq pos(u_r)$. □

Now we present our algorithm 2 which improves the naive algorithm immensely.

Algorithm 2: Improved algorithm that calculates the minimum degree of an S-box F

Input: An (n, m) S-box $F = (F_1, \ldots, F_m)$.
Output: The minimum degree $\texttt{MinDeg}(F)$ of the S-box F.

1. Determine the ANF vectors $A_{F_1}, \ldots, A_{F_m}$ of $F_1, \ldots, F_m$ from their respective truth tables.
2. (a) Create $n + 1$ arrays $V_0, \ldots, V_n$ such that each V_i contains $\binom{n}{i}$ cells, and each cell can contain an n-bit vector.
   (b) For a vector $a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$, put a into the next available cell of $V_{wt(a)}$; repeat this for all the vectors of $\mathbb{F}_2^n$.
   (c) Create the array V of length 2^n as $V = V_n || \cdots || V_0$.
3. Form the $m \times 2^n$ binary matrix M, where the (i, j)-th element is $M_{i,j} = A_{F_i}[V[j]]$.
4. Apply row operations on M to get the row echelon form M' of M.
5. Find the last nonzero row vector in M', say A'_{LAST}.
6. If A'_{LAST} is not the m-th row vector of M', then return 0, else return $wt(V[pos(A'_{LAST})])$.

Correctness of Algorithm 2

Suppose $M_1, \ldots, M_m$ are the row vectors of M, then they are the sorted vector of $A_{F_1}, \ldots, A_{F_m}$, respectively, that follows the order of V. That means, $M_1, \ldots, M_m$ are the sorted ANF vectors of $F_1, \ldots, F_m$ respectively. For each M_i, the coefficients corresponding to the same degree reside in the adjacent

positions, and coefficients corresponding to lower degrees are shifted to the right. If for the i-th row, the leading 1 occurs at the ℓ-th position, and $V[\ell] = (d_i, \ldots, d_n)$, then F_i has degree equal to the weight of $(d_1, \ldots, d_n)$.

Suppose $F_{min} \in \mathtt{LS}^*(F_1, \ldots, F_m)$ that has the lowest degree, i.e., $\mathtt{MinDeg}(F)$. Let $A'_{F_{min}}$ be the ANF vector of F_{min}, which is sorted according to the order of V. Then $A'_{F_{min}}$ belongs to the linear span of the row vectors of M, therefore, it is also some linear combination of the vectors of the row echelon matrix M'.

Clearly, if the rank of M is not m, then the last row of M' is zero, hence the minimum degree in the linear span of them is zero. Suppose F_{LAST} is the function whose ANF vector (which follows the order of V) is A'_{LAST}. Using Proposition 1, we have $pos(A_{F_{min}}) \leq pos(A'_{LAST})$, i.e., the degree of F_{min} cannot be smaller than the degree of F_{LAST}. Moreover, A'_{LAST} itself belongs to $\mathtt{LS}^*(M_1, \ldots, M_m)$, hence the minimum degree of the functions in $\mathtt{LS}(F_1, \ldots, F_m)$ is the degree of the F_{LAST}, i.e., $deg_S(F) = wt(V[pos(A'_{LAST})])$.

Complexity analysis

In Step 1, determining the ANF vector of each F_i requires $O(n2^n)$ steps, i.e., $O(mn2^n)$ in total. Creating the vector V in Step 2 takes $O(2^n)$ steps, and creating the matrix in Step 3 takes $O(m2^n)$ steps. To get the row echelon matrix M' in Step 4 from the $m \times 2^n$ matrix M requires $O(m^2 2^n)$ steps. Then a linear check for each row to find the last nonzero row in Step 5 requires $O(m2^n)$ steps. Therefore, in total the complexity is $O((mn+m^2)2^n)$. Certainly the new algorithm gives far better complexity than the naive one (refer to Table 1).

2.2 Illustration of the Algorithm with an Example

For a better understanding, we illustrate our algorithm by taking an example of the PRESENT S-box $F = (F_1, F_2, F_3, F_4)$ which is an $(4, 4)$ S-box [2]. Each F_i is a Boolean function of 4-variables, x_1, x_2, x_3, x_4. The co-ordinate functions of F namely F_1, F_2, F_3 and F_4 are given in table 2.

Step 1: Computation of ANFs of co-ordinate functions

Using the algorithm given in section (1.1) we calculate ANF vectors of F_1, F_2, F_3 and F_4 as given table 3 below.

Variables				Co-ordinate functions			
x_1	x_2	x_3	x_4	F_1	F_2	F_3	F_4
0	0	0	0	1	1	0	0
0	0	0	1	0	1	0	1
0	0	1	0	0	1	1	0
0	0	1	1	1	0	1	1
0	1	0	0	1	0	0	1
0	1	0	1	0	0	0	0
0	1	1	0	1	0	1	0
0	1	1	1	1	1	0	1
1	0	0	0	0	0	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	0	1	0	0
1	1	0	1	0	1	1	1
1	1	1	0	0	0	0	1
1	1	1	1	0	0	1	0

Table 2: PRESENT S-box

Step 2: Formation of array V

$$
\begin{aligned}
V_0 &= (0,0,0,0) \\
V_1 &= (1,0,0,0),(0,0,0,1),(0,1,0,0),(0,0,1,0) \\
V_2 &= (0,1,0,1),(0,1,1,0),(0,0,1,1),(1,1,0,0), \\
&\quad (1,0,0,1),(1,0,1,0) \\
V_3 &= (1,1,0,1),(1,0,1,1),(0,1,1,1),(1,1,1,0) \\
V_4 &= (1,1,1,1)
\end{aligned}
$$

Variables				ANF vector			
x_1	x_2	x_3	x_4	A_{F_1}	A_{F_2}	A_{F_3}	A_{F_4}
0	0	0	0	1	1	0	0
0	0	0	1	1	0	0	1
0	0	1	0	1	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	1	1	1
1	0	0	1	0	1	0	0
1	0	1	0	0	1	1	0
1	0	1	1	1	1	1	0
1	1	0	0	0	0	1	0
1	1	0	1	1	1	1	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0

Table 3: ANFs of the Co-ordinate Functions

Therefore,

$$\begin{aligned} V[0-15] &= V_4 \| V_3 \| V_2 \| V_1 \| V_0 \\ &= (1,1,1,1),(1,1,0,1),(1,0,1,1),(0,1,1,1), \\ &\quad (1,1,1,0),(0,1,0,1),(0,1,1,0),(0,0,1,1), \\ &\quad (1,1,0,0),(1,0,0,1),(1,0,1,0),(1,0,0,0), \\ &\quad (0,0,0,1),(0,1,0,0),(0,0,1,0),(0,0,0,0). \end{aligned}$$

Step 3: Formation of matrix
First the ANF vectors are rearranged with respect to input array V (Refer to table 4).

V				ANF vector			
x_1	x_2	x_3	x_4	A_{F_1}	A_{F_2}	A_{F_3}	A_{F_4}
1	1	1	1	0	0	0	0
1	1	0	1	1	1	1	0
1	0	1	1	1	1	1	0
0	1	1	1	1	0	1	0
1	1	1	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	1	0	0	1
0	0	1	1	0	1	0	0
1	1	0	0	0	0	1	0
1	0	0	1	0	1	0	0
1	0	1	0	0	1	1	0
1	0	0	0	1	1	1	1
0	0	0	1	1	0	0	1
0	1	0	0	0	1	0	1
0	0	1	0	1	0	1	0
0	0	0	0	1	1	0	0

Table 4: Rearrangement of ANFs with respect to input array V

The arranged ANF vectors are made the rows of a matrix M as

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

Step 4: Row echelon form of the matrix M

The matrix M when transformed to row echelon form becomes

$$M' = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Step 5: Finding A'_{LAST}

Here

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

Step 6: Determination of the minimum degree of S-box

As A'_{LAST} is the last row of matrix M', the minimum degree of the S-box is non-zero, note that $pos(A'_{LAST})$ is the position of leftmost one in A'_{LAST} which is 8 (here the counting starts from zero). We have $V[8] = (1, 1, 0, 0)$. So the minimum degree of F is

$$\texttt{MinDeg}(F) = wt(V[pos(A'_{LAST})]) = wt(V[8]) = 2.$$

3 Conclusions

In this paper we have presented a novel algorithm for checking the minimum degree of an S-box which improves the complexity of the naive algorithm immensely. We are not aware of any existing algorithm other than the naive one. In this regard we also checked SageMath which is an open source software for mathematical computations. It has a function called *min_degree*() that computes the minimum degree of an S-box. We checked that they use the naive algorithm. We implemented our algorithm in Sage and compared the runtime of our algorithm with that of the Sage library. We saw that our algorithm was much faster. For example, we considered the $(8, 8)$ S-box generated by the multiplicative inverse $X \mapsto X^{-1}$ over the field $GF(2^8)$. The minimum degree of this S-box is 7. Our algorithm took 0.023 seconds whereas Sage's own algorithm took 8.624 seconds to finish. This establishes the impact of our algorithm [1]. In case of checking the minimum degree of a large class of S-boxes or S-boxes with larger dimensions the impact will be significant.

[1] Interested readers can approach us for the Sage implementation of our algorithm.

Note that the nonlinearity of the S-box $F = (F_1, \ldots, F_m)$ is the minimum nonlinearity of the functions in $\mathtt{LS}(F_1, \ldots, F_m)$. Thus to determine the nonlinearity one has to consider 2^m functions and then measure the nonlinearity of each of those functions. Therefore, the complexity is $O(n2^{m+n})$. Similar to our algorithm that has improved the naive algorithm to check the minimum degree, improving the naive algorithm for determining the nonlinearity of the S-box will be very interesting.

Acknowledgments: This work has been partially supported by Centre of Excellence in Cryptology, Indian Statistical Institute.

References

[1] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak specifications. *Submission to NIST (Round 3)*, January 2011.

[2] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007.

[3] C. Carlet. Vectorial boolean functions for cryptography. In P. H. Y. Crama, editor, *Boolean Methods and Models*. Cambridge University Press, 2010.

[4] C. Ding, G. Xiao, and W. Shan. *The Stability Theory of Stream Ciphers*, volume 561 of *Lecture Notes in Computer Science*. Springer, 1991.

[5] I. Dinur and A. Shamir. Cube attacks on tweakable black box polynomials.

[6] K. C. Gupta. *Cryptographic and combinatorial properties of Boolean functions and S-boxes*. PhD thesis, Indian Statistical Institute, 2004.

[7] K. C. Gupta and P. Sarakar. Construction of high degree resilient s-boxes with improved nonlinearity. *Information Processing Letters*, 95(3):413–417, Aug. 2005.

[8] J. L. Massey. Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969.

[9] M. Matsui. Linear cryptoanalysis method for des cipher. In T. Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[10] T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, 30(5):776–780, 1984.