A history of the application of MDS matrices in cryptography

Ana Clara Zoppi Serpa
Prof. Dr. Ricardo Dahab
Dr. Jorge Nakahara Jr.

December 15, 2021

Contents

1 A history of the application of MDS matrices in cryptography 2
1.1 Notation . . . 3
1.2 Acronyms . . . 4
1.3 Preliminaries . . . 4
1.3.1 Matrices . . . 4
1.3.2 Abstract algebra . . . 5
1.3.3 Finite fields — GF(2^m) . . . 6
1.3.4 Computational cost unit . . . 7
1.4 MDS matrix catalogue . . . 8
1.5 Computing **xtime** and **xor** of the matrices . . . 11
1.5.1 SQUARE manual calculation example . . . 12
1.6 Conclusions . . . 13

Chapter 1

A history of the application of MDS matrices in cryptography

MDS matrices have been widely used in the construction of diffusion layers for block ciphers such as SHARK [19], SQUARE [9], BKSQ [10], KHAZAD [3], ANUBIS [2], Hierocrypt-3 [8], Rijndael (AES) [11] and Curupira [4]. They have also been applied in the design of hash functions (e.g Whirlwind [1] and Grøstl [12]). The choice is due to the fact that MDS codes provide transformations with optimal linear and differential branch numbers (see e.g [9] or [19]), thus contributing to security against Differential and Linear Cryptanalysis attacks.

When a matrix is MDS, optimal *branch number* (a measure of diffusion power) is ensured, therefore, from the theoretical security perspective, any two distinct $n \times n$ MDS matrices present equal contribution to a cipher's design in terms of diffusion power. However, their computational cost, which is a relevant practical implementation criterion, *is not* necessarily the same. This motivates not only the search for MDS matrices, but the search for *MDS matrices with low computational cost*. In this work, the computational cost of a matrix A is measured by the amount of **xor** and **xtime** operations required when multiplying a cipher's state column vector by A.

Due to the computational cost of matrix multiplication, there is an interest in finding MDS matrices with coefficients as small as possible, in order to minimize the required amount of **xor** and **xtime** operations required by the implementations. However, the complexity of finding MDS matrices through random search increases proportionally to the dimension, which led to the investigation of systematic methods to construct (or find) MDS matrices. One possible avenue is trying to find direct mathematical constructions which ensure the MDS property, and another is to impose restrictions to limit the random search space (e.g imposing the matrix should be circulant, as was done by the authors of [9]). Furthermore, there is an interest in finding involutory MDS matrices (as pointed by [3] and [2]), so that the encryption and the decryption computational cost are the same.

It is also worth noting that, although MDS matrices are widely used in cryptographic algorithms, there are designs which prefer not to make use of them. The block ciphers Serpent [6], IDEA [13] and PRESENT [7], for instance, do not include MDS matrices in their design. The hash function Keccak [5], which was later selected by NIST to become the SHA-3 standard, also does not use MDS matrices. The computational cost can be related to this choice.

In this chapter, we aim at providing a history of the application of MDS matrices in cryptography, listing the matrices, the ciphers in which they have been applied, the respective Finite Fields (order and irreducible polynomial), and their cost (amount of **xor** and **xtime** operations).

Note: this is a partial report. For the moment, it contains only Preliminaries and information about the ciphers SHARK and SQUARE. It will be expanded in the future.

We assume the reader is familiar with:

- Linear branch number (see Chapter X)
- Differential branch number (see Chapter X)
- Differential Cryptanalysis (see Chapter X)
- Linear Cryptanalysis (see Chapter X)
- MDS codes (see Chapter X and, for further detail, reference [15])
- Diffusion property in cryptography (see Chapter X)
- Groups, rings and fields in abstract algebra (see Chapter X)

1.1 Notation

- $det(A)$: determinant of the matrix A
- A^{-1}: inverse matrix of A
- n, k, d: parameters of a code
- $\mathcal{C}$: a code
- G: generator matrix of a code
- I_n: the $n \times n$ identity matrix
- $[I_nB]$: matrix obtained by placing the $n \times n$ matrix B to the right of the $n \times n$ identity matrix I_n. For example, for $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $[I_nB] = \begin{bmatrix} 1 & 0 & 1 & 2 \\ 0 & 1 & 3 & 4 \end{bmatrix}$

1.2 Acronyms

- MDS: Maximum Distance Separable
- AES: Advanced Encryption Standard (Rijndael's name after being chosen by NIST)
- **xor**: bitwise XOR between two bit strings
- **xtime**: refers to the multiplication by the polynomial x in the cipher's Finite Field, i.e GF(2^m) for an integer m

1.3 Preliminaries

1.3.1 Matrices

Obs: eu escrevi essas primeiras definições (matriz singular, involutória, circulante, circulante à esquerda, circulante à direita) com base no que eu lembrava de matemática mesmo, então por hora ainda não coloquei uma referência bibliográfica, já que são definições mais gerais e não chegam a ser específicas de cripto. Mas posso colocar depois se necessário.

Definition 1 (Singular matrix) *A square matrix A is singular if and only if $det(A) = 0$.*

Definition 2 (Non-singular matrix) *A is non-singular if and only if $det(A) \neq 0$.*

Definition 3 (Involutory matrix) *An $n \times n$ square matrix A is involutory if $A \times A = I_n$, where I_n is the identity matrix. In other words, A is involutory when $A = A^{-1}$.*

Definition 4 (Circulant matrix) *An $n \times n$ matrix A is circulant if each row i is formed by a cyclical shift of i positions of the same set of elements $\{a_0, a_1, a_2, ..., a_{n-1}\}$.*

Definition 5 (Left circulant matrix) *A circulant matrix in which the shift is a cyclical shift to the left, i.e*

$$A = \begin{bmatrix} a_0 & a_1 & ... & ... & a_{n-1} \\ a_1 & a_2 & ... & a_{n-1} & a_0 \\ ... & ... & ... & ... & ... \\ a_{n-1} & a_0 & ... & ... & a_{n-2} \end{bmatrix}.$$

Definition 6 (Right circulant matrix) *A circulant matrix in which the shift is a cyclical shift to the right, i.e*

$$A = \begin{bmatrix} a_0 & a_1 & ... & ... & a_{n-1} \\ a_{n-1} & a_0 & a_1 & ... & a_{n-2} \\ ... & ... & ... & ... & ... \\ a_1 & ... & a_{n-2} & a_{n-1} & a_0 \end{bmatrix}.$$

Note that circulant matrices can be defined by just one row, since all the other rows are cyclical shifts of the first. Therefore, they can be denoted as $circ(a_0, a_1, ..., a_{n-1})$. In the case of left circulant and right circulant matrices, respectively, $lcirc(a_0, ..., a_{n-1})$ and $rcirc(a_0, ..., a_{n-1})$. For example, matrices (1.10) and its inverse (1.10), used in the Rijndael cipher, can be denoted as $rcirc(02_x, 03_x, 01_x, 01_x)$ and $rcirc(0e_x, 0b_x, 0d_x, 09_x)$.

Definition 7 (Submatrix) *Given a matrix M, a submatrix of M is the matrix obtained after removing z rows and columns of M, $z \geq 1$, provided that there are sufficient rows (and columns) to be removed.*

Theorem 1 (MDS codes [15]) *An (n, k, d)-code $\mathcal{C}$ with generator matrix $G = [I_n B]$, where B is a $k \times (n-k)$ matrix, is MDS if and only if every square submatrix of B is non-singular.*

We call the B matrix of Theorem 1 the *MDS matrix* throughout this work, i.e the MDS matrices we study are the B matrices of the respective MDS codes chosen when designing them. For further detail on matrices, determinants and linear algebra, the reader may refer to [14].

Definition 8 (Cauchy matrix) *Given $x_0, ..., x_{n-1}$ and $y_0, ..., y_{n-1}$, the matrix A where $A[i][j] = \frac{1}{x_i + y_j}$ is called a Cauchy matrix. According to [20], provided that $x_i \neq x_j$ for $0 \leq i, j \leq n-1$, that $y_i \neq y_j$ for $0 \leq i, j \leq n-1$ and that $x_i + y_j \neq 0$ for all i, j, any square submatrix of a Cauchy matrix is nonsingular over any field.*

An $n \times n$ matrix posesses n^2 elements and thus, when constructing one e.g by selecting random elements, n^2 choices must be made. However, Cauchy and circulant matrices allow us to lower the number of elements we need to select. A circulant matrix can be defined by one row only, therefore only n elements are required. However, there are no guarantees about the MDS property. One must check whether Theorem 1 holds to ensure the obtained matrix is MDS. On the other hand, a Cauchy matrix construction directly ensures the MDS property, as can be seen in Definition 8, requiring $2n$ choices of elements (the x_i and the y_i) to be defined. Furthermore, it is relevant to note that, albeit most matrices in this work have their dimension n be a power of two such as 4 or 8, this is not a requirement for the construction. One can construct $n \times n$ circulant (or Cauchy) matrices for any arbitrary n.

Definition 9 (Hadamard matrix) *yet to write*

Definition 10 (Vandermonde matrix) *yet to write*

1.3.2 Abstract algebra

Aqui pretendo colocar definições de grupo, grupo abeliano, corpo etc. A parte de álgebra abstrata que não é específica de corpos finitos e que a gente geralmente vê na faculdade

1.3.3 Finite fields — GF(2^m)

(Finite field [11]) *A* finite field *is a field with a finite number of elements. The number of elements in the set is called the* order *of the field.*

(Characteristic and order [11]) *A field with order r exists if and only if r is a prime power, i.e $r = p^m$ for some integer m, where p is a prime integer. p is called the* characteristic *of the field. For each prime power there is exactly one finite field, denoted by GF(p^m).*

(Representing finite fields with prime order [11]) *Elements of a finite field GF(p) can be represented by the integers $0, 1, ..., p-1$, and the field operations are integer addition modulo p and integer multiplication modulo p.*

(Representing finite fields with non-prime order [11]) *For finite fields with an order that is not prime, addition and multiplication cannot be represented by addition and multiplication modulo a number. One of the possible representations for GF(p^m) is by means of* polynomials over GF(p).

In this chapter, we focus particularly on fields with characteristic $p = 2$, due to their wide application in cryptography.

(Polynomial [11]) *A polynomial over a field $\mathbb{F}$ is an expression of the form*

$$b(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + ... + b_2x^2 + b_1x + b_0,$$

where x is the indeterminate *and $b_i \in \mathbb{F}$ are the coefficients. The* degree *of the polynomial equals l if $b_j = 0$ for all $j > l$ and l is the smallest number with this property.*

Addition and multiplication are defined on polynomials as follows.

(Polynomial addition [11]) *Summing two polynomials $a(x)$ and $b(x)$ consists of summing the coefficients with equal powers of x, with the sum occuring in the underlying field $\mathbb{F}$. The neutral element is 0 (the polynomial with all coefficients equal to 0). The inverse element can be found by replacing each coefficient by its inverse element in $\mathbb{F}$. The degree of $a(x) + b(x)$ is at most the maximum of the degrees of $a(x)$ and $b(x)$, therefore addition is closed.*

For polynomials over GF(2) stored as integers in a cryptographic software implementation, addition can be implemented with a bitwise XOR instruction.

(Polynomial multiplication [11]) *In order to make multiplication closed, we select a polynomial $p(x)$ of degree l, called the* reduction polynomial. *Multiplication of $a(x)$ and $b(x)$ is then defined as the algebraic product of the polynomials modulo the reduction polynomial $p(x)$.*

The neutral element is 1 (the polynomial of degree 0 and with coefficient of x^0 equal to 1). The inverse element of $a(x)$ is $a^{-1}(x)$ such that $a(x) \times a^{-1}(x) = 1$. Note that $a^{-1}(x)$ exists only when $a(x) \neq 0$.

For polynomials over GF(2) stored as integers in a cryptographic software implementation, multiplication by x can be implemented as a logical bit shift followed by conditional XOR (i.e subtraction) of the reduction polynomial (the **xtime** operation). Multiplication by other polynomials can be implemented as a series of **xtime**.

The reduction polynomial is chosen as an irreducible polynomial.

(Irreducible polynomial [11]) *A polynomial $d(x)$ is irreducible over the field GF(p) if and only if there exist no two polynomials $a(x)$ and $b(x)$ with coefficients in GF(p) such that $d(x) = a(x) \times b(x)$, where $a(x)$ and $b(x)$ are of degree greater than 0.*

For further reference on abstract algebra and Finite Fields, the reader may refer to [16], [18] and [17].

1.3.4 Computational cost unit

Computational cost of multiplication in GF(2^8)

Consider T a state byte, which we multiply by the polynomial $2e_x = 00101110_2 = x^5 + x^3 + x^2 + x$ in GF(2^8). Note that

$$T \cdot 2e_x = T \cdot x^5 + T \cdot x^3 + T \cdot x^2 + T \cdot x = T \cdot x \cdot x \cdot x \cdot x \cdot x + T \cdot x \cdot x \cdot x + T \cdot x \cdot x + T \cdot x,$$

where $\cdot$ denotes multiplication and $+$ denotes addition (which, in GF(2^8), is equivalent to a bitwise XOR). Multiplication by the x polynomial is performed by **xtime**, and addition is performed by **xor**.

Let $T \cdot x = Y$. Then $T \cdot 2e_x = Y + Y \cdot x + Y \cdot x \cdot x + Y \cdot x \cdot x \cdot x \cdot$.

Let $Y \cdot x = W$. Then $T \cdot 2e_x = Y + W + W \cdot x + W \cdot x \cdot x \cdot x$.

Let $W \cdot x = Z$. Then $T \cdot 2e_x = Y + W + Z + Z \cdot x \cdot x$.

The total number of **xtime** operations in this process is 5 (1 to obtain Y from T, 1 to obtain W from Y, 1 to obtain Z from W, 2 to compute $Z \cdot x \cdot x$), since we can reuse intermediate **xtime** calls. The total number of **xor** operations is 3. For multiplication in GF(2^8), in the worst case, 7 **xtime** would be necessary, since the maximum degree of polynomials in GF(2^8) is 7.

Computational cost of a matrix

The computational cost of an n matrix A is given by the necessary **xor** and **xtime** operations when multiplying a $n \times 1$ column vector by A. As an example, we calculate the cost of matrix (1.4), used in the SQUARE [9] cipher.

A row of matrix (1.4) contains the elements $01_x = 1, 02_x = x$ and $03_x = x+1$ only. Multiplying by 01_x does not require **xtime** or **xor**, since $01_x \cdot T = T$. Computing $02_x \cdot T = x \cdot T$ requires 1 **xtime**. Computing $03_x \cdot T = (x+1) \cdot T = T \cdot x + T$ requires 1 **xtime** and 1 **xor**. Furthermore, adding the row multiplication results costs 3 **xor**. Therefore, the cost of a row is 2 **xtime** and 4 **xor**. Equation 1.1 illustrates this, with t_1, t_2, t_3 and t_4 being bytes of the state column vector.

$$
\begin{bmatrix} 02_x & 01_x & 01_x & 03_x \end{bmatrix} \cdot \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = 02_x \cdot t_1 + 01_x \cdot t_2 + 01_x \cdot t_3 + 03_x \cdot t_4 \quad (1.1)
$$

Note that matrix (1.4) contains 4 rows, yielding a total cost of 8 **xtime** and 16 **xor**.

1.4 MDS matrix catalogue

In Table 1.1, the **Ord** column refers to the matrix dimensions, the **Inv** column refers to whether they are involutory or not, **#xor** refers to the necessary amount of **xor** operations, and **#xtime** refers to the necessary amount of **xtime** operations. All finite fields of Table 1.1 have characteristic $p = 2$. The order is 2^m, with m being given by the degree of the irreducible polynomial in the column $GF(2)[x]/(p(x))$. For example, $m = 8$ for SHARK, SQUARE, BKSQ, KHAZAD, ANUBIS, Hierocrypt, Rijndael and the Cauchy matrix found by [20].

Year	**Ord**	**Type**	**Inv**	**Use**	$GF(2)[x]/(p(x))$	**#xor**	**#xtime**	**Matrices**
1996	8	—	no	SHARK [19]	$x^8+x^7+x^6+x^5$ $+x^4+x^2+1$	235 223	369 393	1.2 1.3
1997	4	right circulant	no	SQUARE [9]	$x^8+x^7+x^6+x^5$ $+x^4+x^2+1$	16 40	8 48	1.4 1.5
1997	8	Cauchy	yes	[20]	$x^8+x^4+x^3$ $+x^2+1$	240	344	1.6
1998	3	right circulant	no	BKSQ [10]	poly	9	9	1.12
1999	4	right circulant	no	Rijndael (AES) [11]	$x^8+x^4+x^3$ $+x+1$	16, 40	8, 48	1.10, 1.11
2000	8	Hadamard	yes	KHAZAD [3]	$x^8+x^4+x^3$ $+x^2+1$	112	120	1.7
2000	4	Hadamard	yes	ANUBIS [2]	$x^8+x^4+x^3$ $+x^2+1$	16	20	1.8
2000	ord	type	inv	Hierocrypt-3 [8]	poly	xor	xtime	mat
2007	ord	type	inv	Curupira [4]	poly	xor	xtime	mat
2009	ord	type	inv	Grøstl [12]	poly	xor	xtime	mat
2010	ord	type	inv	Whirlwind [1]	poly	xor	xtime	mat

Table 1.1: MDS matrix usage and cost

Matrix (1.2) and its inverse (1.3) are used in the SHARK [19] cipher.

$$\begin{bmatrix} ce_x & 95_x & 57_x & 82_x & 8a_x & 19_x & b0_x & 01_x \\ e7_x & fe_x & 05_x & d2_x & 52_x & c1_x & 88_x & f1_x \\ b9_x & da_x & 4d_x & d1_x & 9e_x & 17_x & 83_x & 86_x \\ d0_x & 9d_x & 26_x & 2c_x & 5d_x & 9f_x & 6d_x & 75_x \\ 52_x & a9_x & 07_x & 6c_x & b9_x & 8f_x & 70_x & 17_x \\ 87_x & 28_x & 3a_x & 5a_x & f4_x & 33_x & 0b_x & 6c_x \\ 74_x & 51_x & 15_x & cf_x & 09_x & a4_x & 62_x & 09_x \\ 0b_x & 31_x & 7f_x & 86_x & be_x & 05_x & 83_x & 34_x \end{bmatrix} \tag{1.2}$$

$$\begin{bmatrix} e7_x & 30_x & 90_x & 85_x & d0_x & 4b_x & 91_x & 41_x \\ 53_x & 95_x & 9b_x & a5_x & 96_x & bc_x & a1_x & 68_x \\ 02_x & 45_x & f7_x & 65_x & 5c_x & 1f_x & b6_x & 52_x \\ a2_x & ca_x & 22_x & 94_x & 44_x & 63_x & 2a_x & a2_x \\ fc_x & 67_x & 8e_x & 10_x & 29_x & 75_x & 85_x & 71_x \\ 24_x & 45_x & a2_x & cf_x & 2f_x & 22_x & c1_x & 0e_x \\ a1_x & f1_x & 71_x & 40_x & 91_x & 27_x & 18_x & a5_x \\ 56_x & f4_x & af_x & 32_x & d2_x & a4_x & dc_x & 71_x \end{bmatrix} \tag{1.3}$$

Matrix (1.4) and its inverse (1.5) are used in the SQUARE [9] cipher. They are circulant.

$$\begin{bmatrix} 02_x & 01_x & 01_x & 03_x \\ 03_x & 02_x & 01_x & 01_x \\ 01_x & 03_x & 02_x & 01_x \\ 01_x & 01_x & 03_x & 02_x \end{bmatrix} \tag{1.4}$$

$$\begin{bmatrix} 0e_x & 09_x & 0d_x & 0b_x \\ 0b_x & 0e_x & 09_x & 0d_x \\ 0d_x & 0b_x & 0e_x & 09_x \\ 09_x & 0d_x & 0b_x & 0e_x \end{bmatrix} \tag{1.5}$$

Matrix (1.6) is involutory, and was obtained by [20] with a Cauchy construction.

$$\begin{bmatrix} 93_x & 13_x & 57_x & da_x & 58_x & 47_x & 0c_x & 1f_x \\ 13_x & 93_x & da_x & 57_x & 47_x & 58_x & 1f_x & 0c_x \\ 57_x & da_x & 93_x & 13_x & 0c_x & 1f_x & 58_x & 47_x \\ da_x & 57_x & 13_x & 93_x & 1f_x & 0c_x & 47_x & 58_x \\ 58_x & 47_x & 0c_x & 1f_x & 93_x & 13_x & 57_x & da_x \\ 47_x & 58_x & 1f_x & 0c_x & 13_x & 93_x & da_x & 57_x \\ 0c_x & 1f_x & 58_x & 47_x & 57_x & da_x & 93_x & 13_x \\ 1f_x & 0c_x & 47_x & 58_x & da_x & 57_x & 13_x & 93_x \end{bmatrix} \tag{1.6}$$

Matrix (1.7) is Hadamard and involutory. It is used in the KHAZAD [3] cipher.

$$\begin{bmatrix} 01_x & 03_x & 04_x & 05_x & 06_x & 08_x & 0b_x & 07_x \\ 03_x & 01_x & 05_x & 04_x & 08_x & 06_x & 07_x & 0b_x \\ 04_x & 05_x & 01_x & 03_x & 0b_x & 07_x & 06_x & 08_x \\ 05_x & 04_x & 03_x & 01_x & 07_x & 0b_x & 08_x & 06_x \\ 06_x & 08_x & 0b_x & 07_x & 01_x & 03_x & 04_x & 05_x \\ 08_x & 06_x & 07_x & 0b_x & 03_x & 01_x & 05_x & 04_x \\ 0b_x & 07_x & 06_x & 08_x & 04_x & 05_x & 01_x & 03_x \\ 07_x & 0b_x & 08_x & 06_x & 05_x & 04_x & 03_x & 01_x \end{bmatrix} \tag{1.7}$$

Matrix (1.8) is Hadamard and involutory. It is used in the ANUBIS [2] cipher.

$$\begin{bmatrix} 01_x & 02_x & 04_x & 06_x \\ 02_x & 01_x & 06_x & 04_x \\ 04_x & 06_x & 01_x & 02_x \\ 06_x & 04_x & 02_x & 01_x \end{bmatrix} \tag{1.8}$$

Still regarding the ANUBIS cipher, while (1.8) is used as its linear transformation layer, (1.9) is used in the key extraction. It is a Vandermonde construction. When $N = 4$, it is an MDS matrix.

$$\begin{bmatrix} 01_x & 01_x & 01_x & ... & 01_x \\ 01_x & 02_x & 02_x^2 & ... & 02_x^{N-1} \\ 01_x & 06_x & 06_x^2 & ... & 06_x^{N-1} \\ 01_x & 08_x & 08_x^2 & ... & 06_x^{N-1} \end{bmatrix} \tag{1.9}$$

Matrix (1.10) and its inverse (1.11) are used in the Rijndael [11] cipher, which was selected to become AES. They are circulant. We show the hexadecimal notation and the corresponding polynomials to enphasize that, albeit stored as integers in cryptographic software implementation, all matrix elements are actually polynomials in a Finite Field. This applies not only to the Rijndael cipher's matrices but to all matrices listed in this work.

$$\begin{bmatrix} 02_x & 03_x & 01_x & 01_x \\ 01_x & 02_x & 03_x & 01_x \\ 01_x & 01_x & 02_x & 03_x \\ 03_x & 01_x & 01_x & 02_x \end{bmatrix} = \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1x & \end{bmatrix} \tag{1.10}$$

$$\begin{bmatrix} 0e_x & 0b_x & 0d_x & 09_x \\ 09_x & 0e_x & 0b_x & 0d_x \\ 0d_x & 09_x & 0e_x & 0b_x \\ 0b_x & 0d_x & 09_x & 0e_x \end{bmatrix} = \begin{bmatrix} x^3+x^2+x & x^3+x+1 & x^3+x^2+1 & x^3+1 \\ x^3+1 & x^3+x^2+x & x^3+x+1 & x^3+x^2+1 \\ x^3+x^2+1 & x^3+1 & x^3+x^2+x & x^3+x+1 \\ x^3+x+1 & x^3+x^2+1 & x^3+1 & x^3+x^2+x \end{bmatrix} \tag{1.11}$$

Matrix (1.12) is used in the BKSQ [10] cipher. It is circulant.

$$\begin{bmatrix} 03_x & 02_x & 02_x \\ 02_x & 03_x & 02_x \\ 02_x & 02_x & 03_x \end{bmatrix} \tag{1.12}$$

1.5 Computing xtime and xor of the matrices

One can compute the costs manually, however, the following following C code can also be used for this purpose, considering that polynomials in GF(2^8) are stored in integers (a set bit means coefficient equal to 1, a zero bit means coefficient equal to 0).

For e.g SHARK and SQUARE, the field order is equal to 8, therefore `ORDER` must be set to 8 and `DEGREE_LIMIT_MASK` must be set to x^8.

```
#define DEGREE_LIMIT_MASK 0x100
#define ORDER 8
```

The following function obtains the amount of **xtime** required to multiply by the polynomial.

```
unsigned int poly_xtime_cost(unsigned int poly) {
        unsigned int degree_mask = DEGREE_LIMIT_MASK;
        unsigned int degree = ORDER;
        while ((poly & degree_mask) == 0) {
                degree_mask >>= 1;
                degree--;
        }
        return degree;
}
```

The following function obtains the amount of **xor** required to multiply by the polynomial.

```
unsigned int poly_xor_cost(unsigned int poly) {
        unsigned int mask = 1;
        unsigned int set_bits = 0;
        unsigned int current_bit = 0;
        while (current_bit <= ORDER) {
                set_bits += ((poly & mask) != 0);
                mask <<= 1;
                current_bit++;
        }
        return set_bits - 1;
}
```

And, to compute the **xtime** and **xor** costs of a matrix, we must sum the costs of each row, which is accomplished by the following functions. Note that for e.g SHARK `DIM` must be set to 8, for SQUARE, to 4, and so forth.

```c
#define DIM 8

unsigned int matrix_xtime_cost(unsigned int mat [DIM][DIM]) {
        unsigned int total_cost = 0;
        for (int row = 0; row < DIM; row++) {
                unsigned int row_cost = 0;
                for (int col = 0; col < DIM; col++) {
                        row_cost += poly_xtime_cost(mat[row][col]);
                }
                printf("Row %d costs %d xtime\n", row, row_cost);
                total_cost += row_cost;
        }
        printf("The full matrix costs %d xtime\n", total_cost);
        return total_cost;
}

unsigned int matrix_xor_cost(unsigned int mat[DIM][DIM]) {
        unsigned int total_cost = 0;
        for (int row = 0; row < DIM; row++) {
                unsigned int row_cost = DIM - 1; //sum elements
                for (int col = 0; col < DIM; col++) {
                        row_cost += poly_xor_cost(mat[row][col]);
                }
                printf("Row %d costs %d xor\n", row, row_cost);
                total_cost += row_cost;
        }
        printf("The full matrix costs %d xor\n", total_cost);
        return total_cost;
}
```

1.5.1 SQUARE manual calculation example

The computational cost for matrix (1.2), used in the SQUARE cipher, was explained in Section 1.3.4.

For matrix (1.5), used in SQUARE's decryption process, each row contains elements from $\{0e_x, 09_x, 0d_x, 0b_x\}$.

$$0e_x = 00001110_2 = x^3 + x^2 + x \text{ requires 3 } \mathbf{xtime} \text{ and 2 } \mathbf{xor}$$

$$09_x = 00001001_2 = x^3 + 1 \text{ requires 3 } \mathbf{xtime} \text{ and 1 } \mathbf{xor}$$

$$0d_x = 00001101_2 = x^3 + x^2 + 1 \text{ requires 3 } \mathbf{xtime} \text{ and 2 } \mathbf{xor}$$

$$0b_x = 00001011_2 = x^3 + x + 1 \text{ requires 3 } \mathbf{xtime} \text{ and 2 } \mathbf{xor}$$

There are 3 **xor** to add the intermediate row multiplication results, totalizing 12 **xtime** and 10 **xor** per row. There are 4 rows, hence 48 **xtime** and 40 **xor**.

1.6 Conclusions

yet to be written

Bibliography

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

[2] Paulo S. L. M. Barreto and Vincent Rijmen. The ANUBIS block cipher. In *First NESSIE Workshop, Heverlee, Belgium*, 2000.

[3] Paulo S. L. M. Barreto and Vincent Rijmen. The KHAZAD Legacy-Level block cipher. In *First NESSIE Workshop, Heverlee, Belgium*, 2000.

[4] PSLM Barreto and M Simplicio. Curupira, a block cipher for constrained platforms. *Anais do 25o Simpsio Brasileiro de Redes de Computadores e Sistemas Distribudos-SBRC*, 1:61–74, 2007.

[5] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 313–314. Springer, 2013.

[6] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A new block cipher proposal. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 1998.

[7] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[8] Toshiba Corporation. Specification of Hierocrypt-3. In *First NESSIE Workshop, Heverlee, Belgium*, 2000.

[9] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher SQUARE. In Eli Biham, editor, *Fast Software Encryption, 4th Interna-*

tional Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997.

[10] Joan Daemen and Vincent Rijmen. The block cipher BKSQ. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Smart Card Research and Applications, This International Conference, CARDIS '98, Louvain-la-Neuve, Belgium, September 14-16, 1998, Proceedings*, volume 1820 of *Lecture Notes in Computer Science*, pages 236–245. Springer, 1998.

[11] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[12] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl - a SHA-3 candidate. In Helena Handschuh, Stefan Lucks, Bart Preneel, and Phillip Rogaway, editors, *Symmetric Cryptography, 11.01. - 16.01.2009*, volume 09031 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009.

[13] X Lai and J Massey. The idea block cipher. *NESSIE Block Cipher Submissions*, 2000.

[14] S. Lang. *Linear Algebra*. Springer Undergraduate Texts in Mathematics and Technology. Springer, 1987.

[15] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 2nd edition, 1978.

[16] A.M. Masuda and D. Panario. *Tópicos de corpos finitos com aplicações em criptografia e teoria de códigos*. Publicações matemáticas. IMPA, 2007.

[17] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., USA, 1st edition, 1996.

[18] G.L. Mullen and D. Panario. *Handbook of Finite Fields*. Discrete Mathematics and Its Applications. CRC Press, 2013.

[19] Vincent Rijmen, Joan Daemen, Bart Preneel, Antoon Bosselaers, and Erik De Win. The cipher SHARK. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 99–111. Springer, 1996.

[20] A. M. Youssef, S. Mister, and S. E. Tavares. On the design of linear transformations for substitution permutation encryption networks. In C. Adams and M. Just, editors, *Selected Areas in Cryptography, 11th International Workshop, SAC 1997, Ottawa, Canada, August 11 - 12, 1997, Revised Selected Papers*, pages 40–48. Springer, 1997.