Lightweight Diffusion Layer from the k^{th} root of the MDS Matrix

Souvik Kolay[1], Debdeep Mukhopadhyay[1]

[1]Dept. of Computer Science and Engineering
Indian Institute of Technology Kharagpur, India
{*souvik1809,debdeep.mukhopadhyay*}*@gmail.com*

Abstract. The Maximum Distance Separable (MDS) mapping, used in cryptography deploys complex Galois field multiplications, which consume lots of area in hardware, making it a costly primitive for lightweight cryptography. Recently in lightweight hash function: PHOTON, a matrix denoted as 'Serial', which required less area for multiplication, has been multiplied 4 times to achieve a lightweight MDS mapping. But no efficient method has been proposed so far to synthesize such a serial matrix or to find the required number of repetitive multiplications needed to be performed for a given MDS mapping. In this paper, first we provide an generic algorithm to find out a low-cost matrix, which can be multiplied k times to obtain a given MDS mapping. Further, we optimize the algorithm for using in cryptography and show an explicit case study on the MDS mapping of the hash function PHOTON to obtain the 'Serial'. The work also presents quite a few results which may be interesting for lightweight implementation.

Keywords: MDS Matrix, k^{th} Root of a Matrix, Lightweight Diffusion Layer

1 Introduction

With several resource constrained devices requiring support for cryptographic algorithms, the need for lightweight cryptography is imperative. Several block ciphers, like Kasumi[19], mCrypton[24], HIGHT[20], DESL and DESXL[23], CLEFIA[34], Present[8], Puffin[11], MIBS[22], KATAN[9], Klein[14], TWINE[36], LED[16], Piccolo [33] etc and hash functions like PHOTON[33], SPONGENT[7], Quark [3] etc. have been developed specifically to ensure that sufficient security is provided at the cost of less area, less power etc. Designing lightweight ciphers brings several challenges to cryptographers: as classical cryptography depends on resource intensive mathematical operations which cannot be directly be adopted for lightweight platforms directly [27]. Recently another thread of research in the domain of lightweight ciphers have started to evolve: to design lightweight crypto-primitives which can be used to design a family of secured but efficient solutions. One of the fundamental properties of ciphers are their diffusion quality, which is often achieved by error-correcting codes, popularly called as Maximum Distance Separable (MDS) codes. The use of MDS matrix in cryptography to provide perfect diffusion was proposed in [37]. Galois fields arithmetic in characteristic 2, denoted as $\mathbb{GF}(2^n)$ is used extensively for realizing these MDS mappings. Typically

in the ciphers, MDS matrices are multiplied in $\mathbb{GF}(2^n)$ to provide diffusion or mix the input bits to this transformation. Famous examples of such applications are block ciphers like AES [13], Twofish [32], and Camellia [2]. Popular stream ciphers like Mugi [39] and hash functions like WHIRLPOOL [4] also employ such mappings. However in the lightweight literature of ciphers, the initial constructions like HIGHT[20], mCrypton[24], PRESENT [8] do not employ such mappings. Though the use of MDS matrices for diffusion provides good security, but it is costly for hardware implementation. For example, the most compact implementation of AES [26] consumes almost 11% of the total GE only for diffusion layer, which is a $\mathbb{GF}(2^8)$ multiplication of an MDS matrix. Due to this problem, MDS matrices were not so popular initially for compact implementations and in particular for lightweight cryptography. However in the more recent constructions like the block cipher Klein, MIBS, PICCOLO, LED and the hash function PHOTON the use of MDS mappings can be observed. In Klein, MIBS and PICCOLO a circulant MDS mapping is used and the MDS mapping is defined in a smaller $\mathbb{GF}(2^4)$ field instead of $\mathbb{GF}(2^8)$, while in the block cipher LED and hash function PHOTON the MDS mapping has been realized by iterating a different matrix, which is easy to implement. While in the former case, using of the smaller field does not provide the same security guarantees and requires re-evaluation, in the later construction no algorithm or constructive methodology is provided to find out such lightweight matrices which can be iterated to generate the MDS mappings.

Many recent works try to explore new MDS matrices with different features. There are few works where authors construct MDS matrices from Companion matrices [17], Vandermonde matrices [30] and Cauchy matrices [40] which have good cryptographic property and can be implemented efficiently in hardware. In a recent work [31], authors show that it is possible to find some MDS matrices, which can be realized using a recursive Feistel network. In the design of lightweight hash function: PHOTON [15], an instance of MDS matrix is shown which can be obtained from a lightweight matrix, defined as $Serial(1, 2, 1, 4)$ (refer Section 2). These serial matrices [15] can be implemented efficiently in hardware due to its structure. For compact diffusion layer, authors in [15] suggest repetitive multiplications of the serial matrix to get an MDS matrix. Hence, the MDS matrix is denoted as $M = Serial(z_0, \ldots, z_{d-1})^k$, where k denotes, number of times $Serial(z_0, \ldots, z_{d-1})$ needs to be multiplied. But, by picking any random serial matrix and repetitively multiplying will not necessarily produce an MDS matrix with good cryptographic properties. To search for such a serial matrix authors in [15] used an exhaustive search technique using MAGMA and picked the most compact candidate, such that $Serial(z_0, \ldots, z_{d-1})^k$ is an MDS matrix. To the best of our knowledge no efficient method has been proposed so far to obtain such a serial matrix from a given MDS matrix.

In this paper, we investigate the possibility of the existence of a lightweight matrix which can be iterated to realize a MDS mapping using a generalized methodology to compute the k^{th} root of the given MDS mapping. The paper is organized as follows. Some preliminaries on coding theory, linear algebra and finite fields are discussed in section 2. Section 3 presents an algorithm for finding k^{th} root of a matrix, whose elements are in Galois field. In section 4 we show the usefulness of the algorithm in cryptography, with an example of MDS mapping used in the hash function: PHOTON.

Further, we show some interesting results obtained using our algorithm. Finally, we summarize the work done and conclude in section 7.

2 Preliminaries

In this section we present a preliminary overview on the theory of linear block codes, linear algebra and finite field, which will be useful to understand the subsequent sections.

2.1 Preliminaries on Linear Block Code

In cryptography, diffusion is an important property which makes the data bits depend on one another. Due to certain properties of MDS matrix, it is used in cryptography to guarantee a perfect diffusion. In this section, we provide some preliminaries of linear block code for the clarification of the use of MDS matrix in cryptography. For more details on coding theory the readers are referred to [25,29]

Definition 1. Linear Code: *A block code of length n and 2^k codewords is called a* linear (n, k) *code if and only if its 2^k codewords form a k-dimensional subspace of the vector space of all the n-tuples over the field $\mathbb{GF}(2)$.*

Definition 2. Binary Block Code: *A binary block code is linear iff the modulo-2 sum of two codewords is also a codeword.*

Definition 3. (n, k, d) Code: *The* Hamming distance d *of a system of codewords determines the number of errors that can be detected and corrected. If $(n-k)$ check bits are appended to k information bits to get a distance-d code, it is called (n, k, d) code[25].*

Definition 4. MDS Code: *For a linear (n, k, d) code over any field, $d \leq n - k + 1$. Codes with $d = n - k + 1$ are called Maximum Distance Separable (MDS) codes.*

Definition 5. MDS Matrix: *An $m \times n$ matrix over a finite field K is an MDS matrix if it is the transformation matrix of an MDS code. In other words, an (n, k, d) code with generator matrix $G = [I|A]$, where A is a $k \times (n - k)$ matrix is MDS iff every square submatrix (formed from any i rows and any i columns) from $i = 1, 2, \ldots, min(k, n-k)$ of A is non-singular.*

2.2 Preliminaries on Linear Algebra in Galois Field

As the work concentrate on the finding the k^{th} root of a matrix, where the elements of the matrix is in Galois fields, in this section, some basic concepts of linear algebra and Galois field has been discussed. For more details the readers are referred to [18,21]

Definition 6. Galois field: *A field with finite number of elements is said to be* Galois field *or* finite field*. Following are some properties of* Galois field:

- The number of elements in a field is called the order of the field.

– The order of a Galois field is always a prime or power of a prime. Galois fields or finite fields are therefore represented by $\mathbb{GF}(q)$, where $q = p^m$, p is a prime number and m is a positive integer.
– Any element, k of $\mathbb{GF}(q)$ can be represented by a polynomial, where the coefficients of the polynomial are elements of $\mathbb{GF}(p)$ and maximum degree of the polynomial is m. For example, the polynomial representation of 35 in $\mathbb{GF}(3^8)$ is : $1 \times 3^3 + 0 \times 3^2 + 2 \times 3^1 + 2 \times 3^0 \Rightarrow x^3 + 2x + 2$.
– Addition or multiplication of two elements in finite field is done by the polynomial addition and polynomial multiplications respectively and the coefficients of the polynomials follow the arithmetic of $\mathbb{GF}(p)$.
– If $p = 2$, the field is often called as binary field.

Definition 7. Subfield: *A subset $\mathcal{S}$ of a field $\mathcal{F}$ is said to be a* subfield *of $\mathcal{F}$, if the elements of $\mathcal{S}$ satisfy the five* field properties[1].

Definition 8. Extension Field: *A field $\mathcal{K}$ is said to be an extension of a field $\mathcal{F}$, denoted by $\mathcal{K}/\mathcal{F}$ if $\mathcal{F}$ is a subfield of $\mathcal{K}$. $\mathbb{GF}(k)/\mathbb{GF}(q)$ represents that $\mathbb{GF}(k)$ is an extension field of $\mathbb{GF}(q)$.*

Definition 9. Characteristic Polynomial: *If $\mathcal{M}_n$ is a matrix of order $n \times n$, then the* characteristic polynomial *of the matrix is defined as follows:*

$$det(\mathcal{M}_n - \mathcal{I}_n \cdot x)$$

where $\mathcal{I}_n$ denotes the identity matrix of order $n \times n$.

Definition 10. Eigenvalues and Eigenvectors: *The roots of the* characteristic equation *are known as* eigenvalues *of the matrix. For each eigenvalue λ, there is a* eigenvector *$\mathcal{X}$ which satisfies the following equation:*

$$(\mathcal{M}_n - \mathcal{I}_n\lambda) \times \mathcal{X} = 0 \tag{1}$$

Definition 11. Diagonal Matrix: *A* diagonal matrix*, $\mathcal{D}_n$ is a square matrix of order $n \times n$ of the form:*

$$\mathcal{D}_n = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$$

where λ_i is the eigenvalue of the matrix. A diagonal matrix $\mathcal{D}_n$ is often denoted as $diag(\lambda_1, \lambda_2, \cdots, \lambda_n)$

Theorem 1. *A matrix is* diagonalizable *if and only if all the eigenvalues of the matrix are distinct.*

[1]For both addition and multiplication operations, associativity, commutativity, distributivity holds and the identity and inverse element exists

Theorem 2. *Let $\mathcal{D}_n$ be a diagonal matrix, then the following relation holds for any k:*

$$\mathcal{D}_n^k = (diag(\lambda_1, \lambda_2, \cdots, \lambda_n))^k = diag(\lambda_1^k, \lambda_2^k, \cdots, \lambda_n^k)$$

Theorem 3. Eigen Decomposition: *Let $\mathcal{P}$ be a matrix of eigenvectors of a given square matrix $\mathcal{A}$ and $\mathcal{D}$ be a diagonal matrix with the corresponding eigenvalues on the diagonal. Then, if $\mathcal{P}$ is a square matrix, $\mathcal{A}$ can be written as the follow decomposition, known as Eigen Decomposition:*

$$\mathcal{A} = \mathcal{P} \times \mathcal{D} \times \mathcal{P}^{-1}$$

Theorem 4. *Let $\mathcal{A} = \mathcal{P} \times \mathcal{D} \times \mathcal{P}^{-1}$, then the following relation exists for any k:*

$$\mathcal{A}^k = \mathcal{P} \times \mathcal{D}^k \times \mathcal{P}^{-1}$$

Here are some special types of matrices, which have been used in the subsequent sections.

Definition 12. Companion Matrix: *Let $p(t) = c_0 + c_1 t + \cdots + c_{n-1}t^{n-1} + t^n$ be a polynomial with coefficients over an arbitrary field. Then the matrix*

$$\mathcal{C}(p) = \begin{pmatrix} 0 & 0 & \dots & 0 & -c_0 \\ 1 & 0 & \dots & 0 & -c_1 \\ 0 & 1 & \dots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -c_{n-1} \end{pmatrix}$$

is called the companion matrix of the polynomial $p(t)$ since its characteristic polynomial is $p(t)$.

Definition 13. Serial: *$Serial(z_0, \dots, z_{d-1})$ is defined as follows:*

$$Serial(z_0, \dots, z_{d-1}) = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \\ z_0 & z_1 & z_2 & \dots & z_{d-2} & z_{d-1} \end{pmatrix}$$

where $z_0, z_1, z_2, \dots, \in \mathbb{F}_{2^n}$, for some n. It may be noted that Serial is the transpose of the companion matrix.

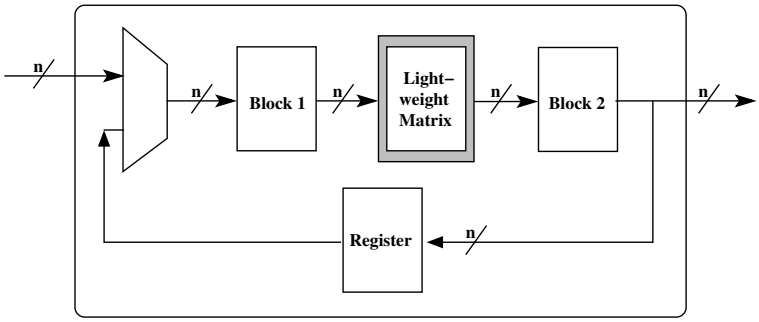
2.3 Lightweight matrix

In this section, first we discuss the alternative options to implement an MDS matrix. Then we suggest some properties by which we can say whether the matrix is lightweight or not, i.e whether the matrix can be implemented in a compact way.
Typically, a lightweight solution can be achieved using the standard hardware serialization techniques. In these cases, the size of the datapath is splitted, so that smaller

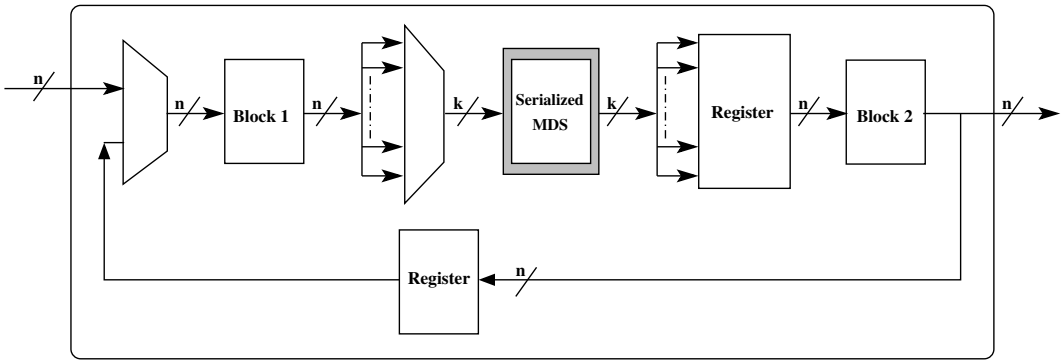
amount of data is processed in each clock cycle. Thus, a smaller hardware can be used repetitively in some clock cycles to process the whole data. Now, in case of any crypto-algorithm there are also other operations along with the MDS mapping. Let 'Block 1' and 'Block 2' denotes the hardware for the operations need to be performed before and after the MDS mapping respectively. The other alternative is to follow the approach adopted by the designers of PHOTON. In this case, without splitting the data path a serial matrix, which can be implemented efficiently in hardware, is multiplied in some clock cycles to obtain the desired MDS matrix. **Figure 2** shows the basic structural difference between architecture of two approaches: (a) Using a lightweight matrix and (b) Using serialized MDS matrix. In standard serialization techniques, (see **figure 2(b)**), a multiplexer is required before the MDS mapping for splitting the datapath and a register is required after the MDS mapping to hold and accumulate the processed data. 'Seralized MDS' denotes the smaller hardware, which can process the splitted datapath in one clock cycle. The additional register requirement may be overcome using the techniques mentioned in [33]. While, in case of 'Lightweight matrix' based implementation, the lightweight matrix of same datapath width is multiplied in each clock cycles. As the size of the datapath remains the same, no extra multiplexer or register is required (see **figure 2(b)**).

Now, suppose there are two matrix $\mathcal{S}$ and $\mathcal{L}$. $\mathcal{S}$ is a serial matrix and $\mathcal{S}^k = \mathcal{M}_{\mathcal{S}}$, while $\mathcal{L}$ is not a serial matrix, but requires same hardware resource as $\mathcal{S}$ and $\mathcal{L}^k = \mathcal{M}_{\mathcal{L}}$. Both $\mathcal{M}_{\mathcal{S}}$ and $\mathcal{M}_{\mathcal{L}}$ are MDS matrices with desired cryptographic properties. In this case, if we want to implement both the two matrices in hardware within k clock cycle, both will require the exact same resource. But if we want to apply the traditional lightweight implementation techniques, to obtain more compact implementation, these two cases may not produce the same result. For example, suppose we want to split the data path in 4 parts, i.e we will compute one byte in each clock cycle. So, we only require the hardware to process each byte in a clock cycle and 4 clock cycle is required to complete the multiplication of a single instance. Now for $\mathcal{S}$, we will require the exact same hardware resources, what is required for the normal implementation (non-serialized). This is because of the fact that by definition serialize matrix carries all the weight in last row. So, the implementation of first three rows will be just a wiring, no gates are required for that, but for the last row, we require the exact same hardware, what was required for the normal implementation. In contrary, for $\mathcal{L}$ the weights are distributed in all the rows. So, the hardware to implement each rows can be shared. In this case, the hardware requirement can be less that or equal to the normal implementation.

From these observation, and keeping the need of hardware serialization for lightweight cryptography in mind, we define a non-serialize matrix to be lightweight, whose each row can be implemented in hardware at around one fourth hardware cost of a serialize matrix.



(a) Using Lightweight Matrix



(b) Using Serialized MDS matrix

Fig. 1. Lightweight Implementation Techniques for MDS Mapping

3 Finding k^{th} root of a matrix with elements in $\mathbb{GF}(p^m)$

In this section, we discuss the algorithm to find out the the k^{th} roots of a given matrix, where the elements of the matrix are in finite field or Galois field. Algorithm 1 provides the steps in brief. Currently the algorithm works for diagonalizable matrix.

Let $q = p^m$, where p is a prime number and m is an integer. $\mathcal{M}_n(q)$ denotes an $n \times n$ matrix with elements in $\mathbb{GF}(q)$. The characteristic polynomial of the matrix $\mathcal{M}_n(q)$ is defined as $\mathcal{C}_{\mathcal{M}_n}(x) = det(\mathcal{M}_n(q) - \mathcal{I}_n \cdot x)$, where $\mathcal{I}_n$ is the identity matrix of order $n \times n$ and $det(\mathcal{A})$ denotes the determinant of matrix $\mathcal{A}$. $\mathcal{C}_{\mathcal{M}_n}(x)$ is a polynomial of degree n and all the operations is done according to $\mathbb{GF}(q)$ arithmetic. The roots of the polynomial $\mathcal{C}_{\mathcal{M}_n}(x)$ are the eigenvalues of the matrix $\mathcal{M}_n(q)$.

The roots of the polynomial can be found by factoring the polynomial $\mathcal{C}_{\mathcal{M}_n}(x)$. Berlekamp's algorithm [6] and Cantor-Zassenhaus algorithm [10] are two most popular probabilistic algorithm for factoring polynomial over finite fields. Berlekamp's algorithm is faster but it has higher space complexity compare to Cantor-Zassenhaus. On the other hand, Ben-Or's algorithm[5] is slightly faster than Cantor-Zassenhaus, without using extra space complexity. For this reason, we choose Ben-Or's algorithm for factorization of the characteristic polynomial. Note that there exists Victor Shoup's

deterministic algorithm[35] for polynomial factorization in finite field but it has a slow performance compared to the other techniques. The output returned by any of these algorithms will be as follows:

$$\mathcal{C}_{\mathcal{M}_n}(x) = f_1^{e_1}(x).f_2^{e_2}(x).f_3^{e_2}(x)\cdots f_s^{e_s}(x)$$

where $f_i(x)$, is an irreducible polynomial of $\mathbb{GF}(q)$ and $\sum e_i = n$, $1 \leq i \leq s$.

- If $f_i(x)$ is a polynomial of degree 1, then there is a root in $\mathbb{GF}(q)$, which is the additive inverse of the constant of the polynomial $f_i(x)$.
- Else the roots are in the extension field $\mathbb{GF}(q^{d_i})/\mathbb{GF}(q)$, w.r.t the primitive polynomial $f_i(x)$, where d_i is the degree of the polynomial. x is the trivial root of $f_i(x)$ as x satisfies the equation $f_i(x) = 0$. Other roots are $(x^q)^j$, where $1 \leq j \leq (d_i - 1)$. In $\mathbb{GF}(q^{d_i})$, x is the polynomial representation of q. Hence, the roots of $f_i(x)$ are q,$(q^q)^1, (q^q)^2 \cdots (q^q)^{d_i-1}$. These roots can be computed using any of the exponentiation by squaring and multiplication method.

Subsequently, finding the roots of all $f_i(x), 1 \leq i \leq s$, the eigenvalues of the matrix $\mathcal{M}_n(q)$ can be computed. Eigen vector corresponding to different eigenvalues can be computed using the equation 1.

Let the eigenvalues be: $\lambda_1, \lambda_2, \cdots, \lambda_n$ and the eigenvectors corresponding to them are $\mathcal{X}_1, \mathcal{X}_2, \cdots, \mathcal{X}_n$ respectively, where $\mathcal{X}_i = \{x_{1i}, x_{2i}, \cdots, x_{ni}\}$. Now, according to Eigen Decomposition Theorem, $\mathcal{M}_n(q)$ can be written as follows:

$$\mathcal{M}_n(q) = \mathcal{P}_n(q^m) \times \mathcal{D}_n(q^m) \times \mathcal{P}_n^{-1}(q^m)$$

where m =maximum degree of the factors of $\mathcal{C}_{\mathcal{M}_n}(x)$, $\mathcal{P}_n^{-1}(q^m)$ is the inverse of $\mathcal{P}_n(q^m)$ and

$$\mathcal{P}_n(q^m) = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix} \qquad \mathcal{D}_n(q^m) = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$$

Finally, from the theorem 4 $\mathcal{M}_n^k(q)$ can be computed as follows:

$$\begin{aligned} \mathcal{M}_n^k(q) &= (\mathcal{P}_n(q^m) \times \mathcal{D}_n(q^m) \times \mathcal{P}_n^{-1}(q^m))^k \\ &= \mathcal{P}_n(q^m) \times \mathcal{D}_n^k(q^m) \times \mathcal{P}_n^{-1}(q^m) \\ &= \mathcal{P}_n(q^m) \times (diag(\lambda_1, \lambda_2, \cdots, \lambda_n))^k \times \mathcal{P}_n^{-1}(q^m) \\ &= \mathcal{P}_n(q^m) \times diag(\lambda_1^k, \lambda_2^k, \cdots, \lambda_n^k) \times \mathcal{P}_n^{-1}(q^m) \end{aligned} \tag{2}$$

when $0 < k < 1$, equation 2 returns the k^{th} root of the matrix $\mathcal{M}_n^k(q)$.

3.1 Complexity of the Algorithm

The complexity of the algorithm can be analyzed as follows:

Algorithm 1: Finding the k^{th} root of the Matrix $\mathcal{M}_n(q)$

Input: MDS Matrix: $\mathcal{M}_n(q)$, k

Output: k^{th} root of the Matrix $\mathcal{M}_n(q)$: $\mathcal{M}_n^{\frac{1}{k}}(q)$

/* Compute the characteristic polynomial ($\mathcal{C}_{\mathcal{M}_n}(x)$) */

$\mathcal{C}_{\mathcal{M}_n}(x) = det(\mathcal{M}_n(q) - \mathcal{I}_n \cdot x)$

/* Find the factors of $\mathcal{C}_{\mathcal{M}_n}(x)$) using Ben-Or's algorithm */

$\mathcal{C}_{\mathcal{M}_n}(x) = f_1^{e_1}(x).f_2^{e_2}(x).f_3^{e_2}(x) \cdots f_s^{e_s}(x)$

/* Find the roots of the factors in the extension field of $\mathbb{GF}(q)$ using the procedure mentioned in section 3 */

$\mathcal{C}_{\mathcal{M}_n}(x) = (x - \lambda_1).(x - \lambda_2) \cdots (x - \lambda_n)$

/* λ_i is the eigenvalue of $\mathcal{M}_n(q)$ in $\mathbb{GF}(q^m)$, where m is the maximum degree of the factor */

$\{\lambda_1, \lambda_2 \cdots \lambda_n\} =$ Eigenvalues $[\mathcal{C}_{\mathcal{M}_n}(x)]$

/* $\mathcal{X}_i$ is the eigenvector w.r.t the eigenvalue λ_i */

$\mathcal{X}_i = \{x_{1i}, x_{2i}, \cdots, x_{ni}\}^T = Eigenvector[\mathcal{C}_{\mathcal{M}_n}(x), \lambda_i]$

/* Construct $\mathcal{P}_n(q^m)$ and $\mathcal{D}_n(q^m)$ */

$$\mathcal{P}_n(q^m) = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix} \quad \mathcal{D}_n(q^m) = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$$

/* Compute the inverse of $\mathcal{P}_n(q^m)$ */

$$\mathcal{P}_n^{-1}(q^m) = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix}^{-1}$$

/* Compute the $\mathcal{D}_n^{\frac{1}{k}}(q^m)$ */

$$\mathcal{D}_n^{\frac{1}{k}}(q^m) = \begin{pmatrix} \lambda_1^{\frac{1}{k}} & 0 & 0 & 0 \\ 0 & \lambda_2^{\frac{1}{k}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n^{\frac{1}{k}} \end{pmatrix}$$

$\mathcal{M}_n^{\frac{1}{k}}(q^m) = \mathcal{P}_n(q^m) \times \mathcal{D}_n^{\frac{1}{k}}(q^m) \times \mathcal{P}_n^{-1}(q^m)$

1. Characteristic polynomial can be computed in $O(n^3)$ using Hessenberg algorithm [12].
2. Factorization of polynomial using Ben-Or's algorithm required expected time complexity of $O(n^2 \,.\, log\, n \,.\, log\, log\, n \,.\, log\, q)$ which is less than $O(n^3)$ if $q = O(n)$.
3. Computation of the eigenvalues requires $O(log\, q)$ multiplications using the exponentiation by squaring method.
4. Eigen vectors can be obtained using the Gaussian-Elimination method, which requires complexity of $O(n^3)$ in finite field [38].
5. Matrix multiplication using school book techniques can be done in $O(n^3)$.
6. Matrix inversion using Guass-Jordan elimination [1] can also be done in $O(n^3)$.

Hence, the overall complexity of the algorithm is $O(n^3)$.

4 Optimization of the algorithm for Cryptography

Further this generic algorithm for finding the k^{th} root of any matrix whose elements are Galois field, can be optimized for cryptography. Because, in cryptography 4×4 MDS matrices are used for diffusion, where each element of the matrix is represented in $\mathbb{GF}(2^8)$. For finding k^{th} root of the MDS matrix, all the computations should be done in $\mathbb{GF}(2^8)$ and its extension field $\mathbb{GF}(2^{8m})/\mathbb{GF}(2^8)$, where $2 \leq m \leq 4$. For binary field ($\mathbb{GF}(2^8)$) both addition and subtraction is exclusive OR($\oplus$) and multiplication is bitwise AND($\cdot$).

Let $q = 2^8$ and $\mathcal{M}_4(q)$ denotes an 4×4 matrix with the elements in $\mathbb{GF}(q)$. The characteristic polynomial of the matrix $\mathcal{M}_4(q)$, $\mathcal{C}_{\mathcal{M}_4}(x) = det(\mathcal{M}_4(q) \oplus \mathcal{I}_4 \cdot x)$ can be factored using any of the algorithm discussed in section 3.

- If a factor is of degree 1, then there is a root in $\mathbb{GF}(2^8)$, which is the constant part of the factor.
- Else the roots are 256, 256^{256}, 256^{512} $\cdots 256^{256\times(d_i-1)}$ in $\mathbb{GF}(2^{8\times d_i})/\mathbb{GF}(2^8)$, where d_i is the degree of the factor. All the d_i roots of the factor can be computed using 8 squarings and $(d_i - 2)$ multiplications. As, d_i can be at most 4, so, all the roots can be found by only 8 squarings and $(d_i - 2)$ multiplications.

Let $\mathcal{X}_i$ be the eigenvector corresponding to the eigenvalue λ_i of the matrix $\mathcal{M}_4(q)$, where

$$\mathcal{M}_4(q) = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \qquad \mathcal{X}_i = \begin{pmatrix} x_{1i} \\ x_{2i} \\ x_{3i} \\ x_{4i} \end{pmatrix}$$

Now, from equation 1

$$\begin{pmatrix} m_{11} \oplus \lambda_i & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} \oplus \lambda_i & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} \oplus \lambda_i & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \oplus \lambda_i \end{pmatrix} \times \begin{pmatrix} x_{1i} \\ x_{2i} \\ x_{3i} \\ x_{4i} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$(m_{11} \oplus \lambda_i).x_{1i} \oplus m_{12}.x_{2i} \oplus m_{13}.x_{3i} \oplus m_{14}.x_{1i} = 0 \tag{3}$$

$$m_{21}.x_{1i} \oplus (m_{22} \oplus \lambda_i).x_{2i} \oplus m_{23}.x_{3i} \oplus m_{24}.x_{1i} = 0 \tag{4}$$

$$m_{31}.x_{1i} \oplus m_{32}.x_{2i} \oplus (m_{33} \oplus \lambda_i).x_{3i} \oplus m_{34}.x_{1i} = 0 \tag{5}$$

$$m_{41}.x_{1i} \oplus m_{42}.x_{2i} \oplus m_{43}.x_{3i} \oplus (m_{44} \oplus \lambda_i).x_{1i} = 0 \tag{6}$$

By the definition of eigenvalues, λ_i, $det(\mathcal{M}_4(q) - \mathcal{I}_4 \cdot \lambda_i) = 0$. So, there must be non-trivial solution for equation 3, 4, 5 and 6. A non-trivial solution can be found using the Gaussian-Elimination method[38].
Further we can use the Eigen decomposition theorem to represent the matrix in the following form:

$$\mathcal{M}_4(q) = \mathcal{P}_4(q^m) \times \mathcal{D}_4(q^m) \times \mathcal{P}_4^{-1}(q^m)$$

Finally, using equation 2, $\mathcal{M}_4^k(q)$ can be obtained. Note that if all the elements of $\mathcal{M}_4^k(q)$, where $0 < k < 1$ are in $\mathbb{GF}(2^8)$, then k^{th} root of the matrix $\mathcal{M}_4(q)$ exists in $\mathbb{GF}(2^8)$, else the k^{th} root is in the extension field of $\mathbb{GF}(2^8)$.

The above algorithm can be used to determine the k^{th} root of a given MDS mapping. In the absence of such an algorithm an exhaustive search needs to be performed to iterate a chosen lightweight matrix, and compute at which power it becomes MDS. However, such a method may be infeasible because of the large number of possible choices for lightweight matrices. Thus the designers of PHOTON, restrict their choice to only Serial matrices as lightweight. In the following example, we show the application of the above theory to determine the 4^{th} root of the MDS matrix of PHOTON. One may note that the choice of $k = 4$ is assumed to be known. However, one may vary that depending on the allowed no of clock cycles which is typically small, as mentioned in section 5.

Example 1. Let $\mathcal{M}_4(q)$ represent the MDS matrix of PHOTON and the Serial matrix $\mathcal{S}_4(q) = Serial(1, 2, 1, 4)$ be such that $\mathcal{S}_4^K(q) = \mathcal{M}_4(q)$. The objective of this exercise is thus to compute $\mathcal{S}_4(q)$ given $\mathcal{M}_4(q)$.

$$\mathcal{M}_4(q) = \begin{pmatrix} 1 & 2 & 1 & 4 \\ 4 & 9 & 6 & 17 \\ 17 & 38 & 24 & 66 \\ 66 & 149 & 100 & 11 \end{pmatrix} \qquad \Rrightarrow \qquad \mathcal{S}_4(q) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 4 \end{pmatrix}$$

$\Rrightarrow$ The characteristic polynomial of $\mathcal{M}_4(q)$ is:

$$\begin{aligned} \mathcal{C}_{\mathcal{M}_4}(x) &= det(\mathcal{M}_4(q) \oplus \mathcal{I}_4 \cdot x) \\ &= x^4 \oplus 27x^3 \oplus x^2 \oplus 16x \oplus 1 \\ &= (x \oplus 37)(x \oplus 217)(x^2 \oplus 231x \oplus 30) \end{aligned}$$

Hence, 37 and 217 are two eigenvalues, which are in $\mathbb{GF}(2^8)$ and the other two eigenvalues are in $\mathbb{GF}(2^8)^2$. As explained in section 4, the roots of $x^2 \oplus 231x \oplus 30 = 0$ are 256 and $256^{256} = 487$, which are in $\mathbb{GF}(2^8)^2$. So, the four eigenvalues are 37, 217, 256, 487.

Gaussian-Elimination method has been used to find out the following eigenvectors corresponding to the eigenvalues.

Eigenvalues	Eigenvectors
37	$\{232, 228, 94, 1\}$
217	$\{181, 133, 146, 1\}$
256	$\{11003, 43997, 53119, 1\}$
487	$\{10920, 43960, 53169, 1\}$

$$\mathcal{P}_4(q^2) = \begin{pmatrix} 232 & 181 & 11003 & 10920 \\ 228 & 133 & 43997 & 43960 \\ 94 & 146 & 53119 & 53169 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad \mathcal{P}_4^{-1}(q^2) = \begin{pmatrix} 234 & 62 & 81 & 38 \\ 43 & 218 & 243 & 199 \\ 48831 & 60741 & 54807 & 28627 \\ 48766 & 60883 & 54965 & 28467 \end{pmatrix}$$

$$\mathcal{D}_4(q^2) = \begin{pmatrix} 37 & 0 & 0 & 0 \\ 0 & 217 & 0 & 0 \\ 0 & 0 & 256 & 0 \\ 0 & 0 & 0 & 487 \end{pmatrix} \qquad \mathcal{D}_4^{\frac{1}{4}}(q^2) = \begin{pmatrix} 97 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 43041 & 0 \\ 0 & 0 & 0 & 43126 \end{pmatrix}$$

Hence from equation 2,

$$\begin{aligned} \mathcal{M}_4^{\frac{1}{4}}(q) &= \mathcal{P}_4(q^2) \times \mathcal{D}_4^{\frac{1}{4}}(q^2) \times \mathcal{P}_4^{-1}(q^2) \\ &= \begin{pmatrix} 232 & 181 & 11003 & 10920 \\ 228 & 133 & 43997 & 43960 \\ 94 & 146 & 53119 & 53169 \\ 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 97 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 43041 & 0 \\ 0 & 0 & 0 & 43126 \end{pmatrix} \\ &\qquad \times \begin{pmatrix} 234 & 62 & 81 & 38 \\ 43 & 218 & 243 & 199 \\ 48831 & 60741 & 54807 & 28627 \\ 48766 & 60883 & 54965 & 28467 \end{pmatrix} \\ &= \begin{pmatrix} 228 & 133 & 43997 & 43960 \\ 94 & 146 & 53119 & 53169 \\ 1 & 1 & 1 & 1 \\ 97 & 50 & 43041 & 43126 \end{pmatrix} \times \begin{pmatrix} 234 & 62 & 81 & 38 \\ 43 & 218 & 243 & 199 \\ 48831 & 60741 & 54807 & 28627 \\ 48766 & 60883 & 54965 & 28467 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 4 \end{pmatrix} = \mathcal{S}_4(q) \end{aligned}$$

Thus, using the steps of the algorithm 1, we can get the Serial matrix, $S_4(q)$ from the actual MDS matrix used in PHOTON.

So far, we have seen the way to compute $M_4^{\frac{1}{k}}(q)$, where is $M_4(q)$ denotes the general MDS matrix used in cryptography. Now, we need to guess the possible values of k, for which $S_4(q) = M_4^{\frac{1}{k}}(q)$ is a lightweight matrix.

5 Guessing the *k* and finding the lightweight matrix

In this section, first we compare the advantages of our work with the alternative options available for lightweight implementations and then propose some heuristics to guess the possible values of k, so that the advantages can be maximized.

We are trying to find a lightweight matrix $\mathcal{S}_4(q)$(which can be a serial matrix or not), such that $\mathcal{S}_4^k(q)$ is an MDS matrix $\mathcal{M}_4(q)$.

Alternately a lightweight solution can be achieved using the standard hardware serialization techniques. In these cases, the size of the datapath is splitted, so that smaller amount of data is being processed in each clock cycle. Thus, a smaller hardware can be used repetitively in some clock cycles to process the whole data.

5.1 Comparison between two paradigms

For any crypto-algorithm there are also other operations along with the MDS mapping. Let 'Block 1' and 'Block 2' denote the hardware for the operations need to be

performed before and after the MDS mapping respectively. **Figure 2** shows the basic structural difference between architecture of two approaches: (a) Using a lightweight matrix (defined later in section 5.2) and (b) Using serialized MDS matrix[2]. To compare these two approaches first we fix the metrics for area and throughput.

Metric for Area: As all the computations should be done in $\mathbb{GF}(2^8)$, so the entire MDS matrix multiplication can be performed using only XORs. Due to this reason, we have chosen the XORs count of the circuit to measure the area requirement.

Metric for Throughput: The maximum frequency for lightweight devices is generally very low (around 100 KHz) [8] because higher frequency requires more power, which is not available for these resource constrained devices. Thus, we can assume that whatever may be the delay of the critical path of the diffusion layer, it will be still faster enough to support the maximum frequency of these lightweight devices. Hence, the number of clock cycle required plays the most important role on the throughput. For this reason, we consider the number of clock cycles as a measure of the throughput.

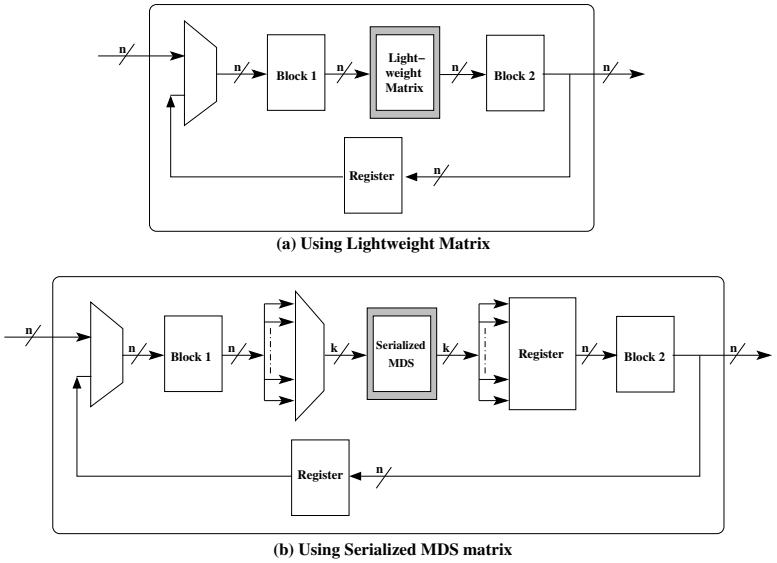


Fig. 2. Lightweight Implementation Techniques for MDS Mapping

– In standard serialization techniques, (see **figure 2(b)**), a multiplexer is required before the MDS mapping for splitting the datapath and a register is required after the MDS mapping to hold and accumulate the processed data. 'Serialized MDS'

[2]This is not the 'Serial' matrix, but a standard hardware serialization technique adopted to perform the MDS matrix multiplication

denotes the smaller hardware, which can process the splitted datapath in one clock cycle. The additional register requirement may be overcome using the techniques mentioned in [33].
– In case of 'Lightweight matrix' based implementation, the lightweight matrix of same datapath width is multiplied in each clock cycles. As the size of the datapath remains the same, no extra multiplexer or register is required (see **figure 2(b)**).

5.2 Area Requirement for a Matrix to be 'Lightweight'

Let the number of XORs required for $\mathcal{S}_4(q)$ be $\#\mathcal{XOR}_{\mathcal{S}_4(q)}$ and the number of XORs required for $\mathcal{M}_4(q)$ be $\#\mathcal{XOR}_{\mathcal{M}_4(q)}$. Now, if we try to serialize the implementation of $\mathcal{M}_4(q)$ 'using serialized MDS matrix', which requires k clock cycles and area requirement $\mathcal{A}_{Serialize(\mathcal{M}_4(q))}$, then ideally the following holds:

$$\mathcal{A}_{Serialize(\mathcal{M}_4(q),\,k)} = Area\left(\frac{\#\mathcal{XOR}_{\mathcal{M}_4(q)}}{k}\right) + Area\left((k:1)\text{ MUX of width}\frac{n}{k}\right) \quad (7)$$

where n is the actual bit width of the datapath.
But using $\mathcal{S}_4(q)$, k times to get $\mathcal{M}_4(q)$ have the following advantages:

– This does not need additional MUX, because we are not modifying the size of the datawidth, i.e the same size matrix is being multiplied each time with the column vectors.
– The estimation provided in equation 7 is only a lower bound. In actual scenario, this can be violated significantly because it is not always possible to use all the shared hardware(XORs) in each of the clock cycle, which also leads to some extra MUX in the design. Note that the different rows in MDS mapping can not be implemented without extra MUX. Whereas in case of lightweight matrix based implementation, as the matrix to be multiplied is always the same, all the XORs are being used in each of the clock cycle. Hence, we do not need any extra MUX.

Hence a matrix, obtained from the algorithm 1 can only be considered as a lightweight matrix, if it takes lesser area than its serialized implementation i.e if $\#\mathcal{XOR}_{\mathcal{S}_4(q)}$ equals to or nearly equals to the $\mathcal{A}_{Serialize(\mathcal{M}_4(q),\,k)}$.

5.3 Choosing the proper value of k

Keeping these facts in mind, we recommend to pick k, such that gate equivalent of $\#\mathcal{XOR}_{\mathcal{S}_4(q)}$ equals to or nearly equals to the $\mathcal{A}_{Serialize(\mathcal{M}_4(q),\,k)}$. Generally, for standard lightweight implementation k should be small else the number of clock cycle required will be high. For our experiments, we keep k in the range $[2, 6]$. Hence, a lightweight matrix $\mathcal{S}_4(q)$ can be obtained from a given MDS matrix $\mathcal{M}_4(q)$ using the following steps:

1. Compute the number of XORs($\#\mathcal{XOR}_{\mathcal{M}_4(q)}$) required for the hardware implementation of the MDS matrix $\mathcal{M}_4(q)$.
2. Initialize k as 2
3. Apply algorithm 1 to compute $\mathcal{S}_4(q) = \mathcal{M}_4^{\frac{1}{k}}(q)$

4. Compute the XOR requirement, $\#\mathcal{XOR}_{\mathcal{S}_4}$ for the matrix $\mathcal{S}_4(q)$
5. Estimate the area requirement $\mathcal{A}_{Serialize(\mathcal{M}_4(q),\,k)}$ for the standard hardware serialization techniques using equation.
6. If Area($\#\mathcal{XOR}_{\mathcal{S}_4(q)}$) is less than $\mathcal{A}_{Serialize(\mathcal{M}_4(q),\,k)}$, add the matrix $\mathcal{S}_4(q)$ as a probable lightweight solution.
7. Increment the value of k by one and go to step 3 until k equals to 6
8. If there are more than one lightweight matrix, by the repetitive multiplication of which, the original matrix can be obtained, then choose the one which is best suited in terms of metrics for area and throughput.

6 Results

Though the proposed algorithm can also be used to find out a 'Serial Matrix' (if exists) from a given MDS matrix. One can point out that the output of the algorithm will not always produce a 'Serial' matrix but it is actually an advantage of the algorithm.

Any 4×4 'Serial' matrix can never produce an MDS matrix if it is multiplied less than four times.

Let $\mathcal{S}$ be a 'Serial' matrix, then the following holds.

$$\mathcal{S} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ x_1 & x_2 & x_3 & x_4 \end{pmatrix} \quad \mathcal{S}^2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \end{pmatrix} \quad \mathcal{S}^3 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \end{pmatrix}$$

Now any $4 * 4$ matrix must have all non-zero element to be an MDS matrix[17] (necessary but not sufficient condition). Hence $\mathcal{S}$, $\mathcal{S}^2$ and $\mathcal{S}^3$ cannot be an MDS. Whereas using the proposed algorithm, we can find a lightweight matrix which can be multiplied less than four times to get an MDS matrix.

We picked some number of random 4×4 MDS matrices whose elements are in $\mathbb{GF}(2^8)$ and used the proposed algorithm to find the kth root of the matrix. Table 1 shows some of the interesting results obtained. It may also be noted that serialized hardware is a popular technique for lightweight cipher implementation. However the implementation becomes efficient in the context of the MDS mapping when the matrix has the rows with similar computation overheads. However it can be observed that the Serial matrix has only one row with the entire computation whereas the other rows do not have any computation. Hence a serialized approach does not provide any benefit in that case. This motivates further the search for other lightweight matrices which can be iterated to obtain an MDS mapping. The second example in Table 1 illustrates this point for PHOTON, where our algorithm in addition to the Serial matrix provides a 7^th root lightweight matrix (which is not a Serial matrix) but because of its uniform distribution is more amenable to hardware serialization.

7 Conclusion

In this paper, we proposed an efficient technique to compute k^{th} root of a diagonalizable matrix, whose elements are in Galois field within the complexity of $O(n^3)$. After having

Table 1. Some interesting MDS matrices

MDS Matrix	$k^{th} root$ **of the Matrix**	k	**Comments**
$\begin{pmatrix} 14 & 3 & 11 & 35 \\ 2 & 2 & 1 & 3 \\ 13 & 1 & 11 & 40 \\ 26 & 6 & 21 & 71 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 2 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 3 \\ 1 & 0 & 1 & 4 \end{pmatrix}$	3	This is an example where we can get an matrix, almost as lightweight as a 'Serial' matrix but can produce an MDS matrix in only 3 multiplications. Obtaining an MDS matrix by 3 times multiplication is not possible for any 'Serial' matrix.
$\begin{pmatrix} 1 & 2 & 1 & 4 \\ 4 & 9 & 6 & 17 \\ 17 & 38 & 24 & 66 \\ 66 & 149 & 100 & 11 \end{pmatrix}$	$\begin{pmatrix} 255 & 204 & 172 & 250 \\ 250 & 16 & 54 & 105 \\ 105 & 40 & 121 & 137 \\ 137 & 96 & 161 & 107 \end{pmatrix}$	7	This is the MDS matrix used in PHOTON. The interesting property of this MDS matrix is that it does not have any root for $k = 5, 6$, but it has a 7^{th} root. This 7^{th} root of the matrix can be used as an alternative to standard hardware serialization technique.
$\begin{pmatrix} 4 & 2 & 1 & 1 \\ 8 & 6 & 5 & 6 \\ 11 & 14 & 10 & 6 \\ 2 & 2 & 15 & 11 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 4 & 2 & 1 & 1 \end{pmatrix}$	4	This is the MDS matrix used in the diffusion layer of the lightweight block cipher LED. So far this is the only algorithm other than PHOTON to use a 'Serial' matrix for diffusion layer.

the k^{th} root of the matrix, say $\mathcal{S}$, we can estimate the hardware requirement for $\mathcal{S}$. We also suggest some heuristics to choose the value of the k, so that the hardware requirement of $\mathcal{S}$ is less. Further, we provide an explicit case study for the hash function PHOTON and show the reduction in hardware due to the proposed architecture. The paper also presents few results which may be interesting for the implementation of lightweight diffusion layer. . We believe that using the proposed algorithm one can find quite a few lightweight matrix, which will be suitable for lightweight cryptography.

References

1. S. C. Althoen and R. McLaughlin. Gauss-jordan reduction: a brief history. *Am. Math. Monthly*, 94(2):130–142, Feb. 1987.
2. K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, A. M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. Camellia: A 128-bit block cipher suitable for multiple platforms - design and analysis, 2000.
3. J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A lightweight hash. *J. Cryptology*, 26(2):313–339, 2013.
4. P. S. L. M. Barreto and V. Rijmen. Whirlpool. In H. C. A. van Tilborg and S. Jajodia, editors, *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 1384–1385. Springer, 2011.
5. M. Ben-Or. Probabilistic algorithms in finite fields. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:394–398, 1981.
6. E. R. Berlekamp. Factoring polynomials over finite fields. In *Bell System Technical Journal*, page 18531859, 1967.
7. A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede. Spongent: The design space of lightweight cryptographic hashing. *IEEE Trans. Computers*, 62(10):2041–2053, 2013.
8. 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, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.
9. C. D. Cannière, O. Dunkelman, and M. Knezevic. Katan and ktantan - a family of small and efficient hardware-oriented block ciphers. In C. Clavier and K. Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.
10. D. G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. In *Mathematics of Computation, Vol 36*. American Mathematical Society.
11. H. Cheng, H. M. Heys, and C. Wang. Puffin: A novel compact block cipher targeted to embedded digital systems. In *DSD*, pages 383–390, 2008.
12. H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 1993.
13. J. Daemen and V. Rijmen. The block cipher rijndael. In Quisquater and Schneier [28], pages 277–284.
14. Z. Gong, S. Nikova, and Y. W. Law. Klein: A new family of lightweight block ciphers. In *RFIDSec*, pages 1–18, 2011.
15. J. Guo, T. Peyrin, and A. Poschmann. The photon family of lightweight hash functions. In P. Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2011.
16. J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The led block cipher. In *CHES*, pages 326–341, 2011.
17. K. C. Gupta and I. G. Ray. On constructions of mds matrices from companion matrices for lightweight cryptography. In A. Cuzzocrea, C. Kittl, D. E. Simos, E. Weippl, L. Xu, A. Cuzzocrea, C. Kittl, D. E. Simos, E. Weippl, and L. Xu, editors, *CD-ARES Workshops*, volume 8128 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2013.
18. I. N. Herstein. *Topics in Algebra*. Wiley, June 20, 1975.
19. V. T. Hoang and P. Rogaway. Design principles of the kasumi block cipher. *IACR Cryptology ePrint Archive*, 2010:301, 2010.
20. D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee. Hight: A new block cipher suitable for low-resource device. In *CHES*, pages 46–59, 2006.

21. R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, February 23, 1990.
22. M. Izadi, B. Sadeghiyan, S. S. Sadeghian, and H. A. Khanooki. Mibs: A new lightweight block cipher. In *CANS*, pages 334–348, 2009.
23. G. Leander, C. Paar, A. Poschmann, and K. Schramm. New lightweight des variants. In *FSE*, pages 196–210, 2007.
24. C. H. Lim and T. Korkishko. mcrypton - a lightweight block cipher for security of low-cost rfid tags and sensors. In *WISA*, pages 243–258, 2005.
25. F. J. MacWilliams. *The Theory of Error-Correcting Codes*. North-Holland Mathematical Library, 1977.
26. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the limits: A very compact and a threshold implementation of aes. In K. G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.
27. A. Y. Poschmann. *LIGHTWEIGHT CRYPTOGRAPHY*. PhD thesis, February, 2009.
28. J.-J. Quisquater and B. 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*. Springer, 2000.
29. T. R. N. Rao and E. Fujiwara. *Error-Control Coding for Computer Systems*. Prentice Hall series in computer engineering, January 1989.
30. M. Sajadieh, M. Dakhilalian, H. Mala, and B. Omoomi. On construction of involutory mds matrices from vandermonde matrices in gf(2^q). *Des. Codes Cryptography*, 64(3):287–308, 2012.
31. M. Sajadieh, M. Dakhilalian, H. Mala, and P. Sepehrdad. Recursive diffusion layers for block ciphers and hash functions. In A. Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 385–401. Springer, 2012.
32. B. Schneier and D. Whiting. Twofish on smart cards. In Quisquater and Schneier [28], pages 265–276.
33. K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: An ultra-lightweight blockcipher. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 342–357. Springer, 2011.
34. T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata. The 128-bit blockcipher clefia (extended abstract). In *FSE*, pages 181–195, 2007.
35. V. Shoup. Smoothness and factoring polynomials over finite fields. In *Math. Comp*, pages 398–406, 1996.
36. T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi. Twine: A lightweight, versatile block cipher. In *ECRYPT Workshop on Lightweight Cryptography - November 2011*, volume 2011, pages 148–169, 2011.
37. S. Vaudenay. On the need for multipermutations: Cryptanalysis of md4 and safer. In B. Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 1994.
38. F. R. W. *Linear Least Squares Computations*. Marcel Dekker, 1988.
39. D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, and B. Preneel. A new keystream generator mugi. *IEICE Transactions*, 87-A(1):37–45, 2004.
40. A. M. Youssef, S. Mister, and S. E. Tavares. On the design of linear transformations for substitution permutation encryption networks. In *School of Computer Science, Carleton University*, pages 40–48, 1997.