New Subquadratic Algorithms for Constructing Lightweight Hadamard MDS Matrices

Tianshuo Cong[a], Ximing Fu[b,c,*], Xuting Zhou[e], Yuli Zou[d] and Haining Fan[e]

[a] *Institute for Advanced Study, Tsinghua University, Beijing, China*
[b] *The Chinese University of Hong Kong, Shenzhen, Shenzhen, China*
[c] *University of Science and Technology of China, Hefei, China*
[d] *Alibaba Local Life Service Lab, Shanghai, China*
[e] *Department of Computer Science and Technology, Tsinghua University, Beijing, China*

ARTICLE INFO

Keywords:
Lightweight cryptography
MDS matrix
Hadamard matrix
Involution
Subquadratic matrix-vector product

ABSTRACT

Maximum Distance Separable (MDS) Matrix plays a crucial role in designing cryptosystems. In this paper we mainly talk about constructing lightweight Hadamard MDS matrices based on subquadratic multipliers over GF(2^4). We firstly propose subquadratic Hadamard matrix-vector product formulae (HMVP), and provide two new XOR count metrics. To the best of our knowledge, subquadratic multipliers have not been used to construct MDS matrices. Furthermore, combined with HMVP formulae we design a construction algorithm to find lightweight Hadamard MDS matrices under our XOR count metric. Applying our algorithms, we successfully find MDS matrices with the state-of-the-art fewest XOR counts for 4×4 and 8×8 involutory and non-involutory MDS matrices. Experiment results show that our candidates save up to 40.63% and 10.34% XOR gates for 8×8 and 4×4 matrices over GF(2^4) respectively.

1. Introduction

With the rapid development and application of source restricted equipment like Radio Frequency Identification (RFID), lightweight ciphers have attracted great attention, such as stream ciphers `Trivium` [13], `Grain v1` [22], `MICKEY 2.0` [1] and block ciphers `LED` [19], `CLEFIA` [36], etc. Confusion layer and diffusion layer play crucial roles in designing symmetric ciphers in terms of security and efficiency. The branch number is the primary factor for the performance of diffusion layers. Maximum Distance Separable (MDS) matrix has the biggest branch number and hence provides the best resistance to differential and linear attacks. As a consequence, lightweight MDS matrices have been applied in diffusion layers to provide better security and hardware performances: Block cipher `Shark` [32] firstly utilizes MDS matrix; `AES` [11] uses a 4×4 matrix over GF(2^8), which occupies a wealth of hardware area resources; 4×4 matrix in `LED` is hardware-friendly. MDS matrices are also widely used as diffusion layers in other block ciphers such as `CLEFIA`, `FOX` [25], `KHAZAD` [2], `ANUBIS` [3], `Square` [10], `Twofish` [35], `Joltik` [23]; some stream ciphers [12] and hash functions [17, 18, 29].

The method of constructing MDS matrices is mainly considered from two perspectives: structure and involution. Two networks are widely used in block ciphers, subtitution-permutation networks (SPNs) and Feistel structure. Matrix with lightweight inverse matrix could be used in SPNs and hardware-friendly involutory MDS matrices could be used in Feistel structures because of the identical process of Feistel's encryption and decryption. The diffusion layers of `AES`, `Grøstl`, `WHIRLPOOL` use circulant matrices. The diffusion layer of `KHAZAD` uses Hadamard matrix. In [9], authors propose compact Cauchy matrices which have the fewest different entries, they prove that all compact Cauchy matrices could be improved into self-inverse ones. In [33], the application of Toeplitz matrices in lightweight diffusion layers was discussed. The authors prove that Toeplitz matrices could not be both MDS and involutory, they give the result of 4×4 involutory MDS matrix with small number of XOR operations. It is proved in [37] that there are equivalent classes of various MDS matrices, for example there are 30 equivalent classes of 8×8 Hadamard matrices. In [24], the authors propose a tool named LIGHTER to produce optimized implementations of small functions in lightweight cryptographic designs. They find 4×4 and 8×8 involutory and non-involutory MDS matrices over GF(2^4) and GF(2^8) with fewer XOR counts.

The GF(2^c) multiplication can be represented as a matrix-vector product and MDS matrices usually appear in matrix-vector multiplication, so the method to design multiplier can also be adapted to design MDS matrices. A typical hardware evaluation of the given MDS matrix is the total number of 2-input XOR gates cost because an addition operation over GF(2) could be realized by a 2-input Exclusive Or (XOR) gate, and addition operation over GF(2^c) can be realized by c 2-input XOR gates with one XOR gate delay. Minimizing the number of XOR gates needed in matrix-vector multiplication has always been a concern [28, 5]. Bit parallel GF(2^c) multipliers could be classified into two categories: quadratic and subquadratic multipliers [15]. Quadratic multipliers are built on the straightforward computation and they have high space complexity [30, 31, 34, 21]. Subquadratic multiplication algorithms could be utilized to design low space complexity GF(2^c) multipliers for large c [4, 16, 6, 26]. In this

*Corresponding author

`cts17@mails.tsinghua.edu.cn` (T. Cong); `fuximimg@cuhk.edu.cn` (X. Fu); `zhouxt19@mails.tsinghua.edu.cn` (X. Zhou); `zyuli1027@gmail.com` (Y. Zou); `fhn@tsinghua.edu.cn` (H. Fan)
ORCID(s):

work, we design lightweight MDS matrices using the subquadratic Toeplitz matrix-vector product formulae in [14] (TMVP).

1.1. Our Contributions

The goal in this work is to construct lightweight involutory and non-involutory MDS matrices by using subquadratic Hadamard matrix-vector product formulae. To the best of our knowledge, this approach has not been used to design MDS matrices with low XOR count before.

Considering lightweightness, we mainly consider MDS matrices over small fields. We construct four kinds of MDS matrices, which are involutory and non-involutory 4×4 and 8×8 Hadamard matrices over $GF(2^4)$. We propose two new XOR count metrics and use the subquadratic algorithms to construct lightweight MDS matrices over $GF(2^4)$ defined by three irreducible polynomials under our new metrics. Comparison with the best results of previous work is shown in Table 1. The known lower bounds are from [24] (FSE2017), which are the benchmarks in our experiment. **Inv.** with the check mark means involutions.

Table 1
Summary table of the MDS matrices

Dimension	Inv.	XOR count	[24]	Comparison
4×4	✓	58	63	7.94%
8×8	✓	282	424	33.49%
4×4		52	58	10.34%
8×8		228	384	40.63%

The rest of this paper is organised as follows. Section 2 briefly introduces some basic concepts such as background knowledge of MDS matrices and subquadratic TMVP formulae. Section 3 begins by laying out the applications of HMVP to Hadamard matrices, and introduces the searching process of involutions and non-involutions. Section 4 presents the experiment parameters and searching results of the algorithm. Meanwhile we compare our MDS matrices with the previous results to show the efficiency. Finally, Section 5 concludes this paper.

2. Preliminaries

In this section, some basic concepts and subquadratic HMVP formulae will be introduced. The elements of the Hadamard matrix in our work all belong to the finite field $GF(2^c)$ generated by degree-c irreducible polynomial $p(x)$, which is denoted as $GF(2^c)/p(x)$. We use hexadecimal form to denote $p(x)$ and the elements in matrices.

2.1. Branch Number

The branch number is a key index for inspecting the performance of diffusion layers. It could quantify the avalanche effect.

Definition 1 (Hamming weight). *The hamming weight $w_h(\mathbf{x})$ is the number of non-zero components of the vector $\mathbf{x}$.*

Definition 2 (Branch number). *For a linear invertible mapping $\theta : [GF(2^m)]^n \to [GF(2^m)]^n$, the branch number is*

$$\mathcal{B}(\theta) = \min_{\mathbf{x}\neq0} (w_h(\mathbf{x}) + w_h(\theta(\mathbf{x}))). \quad (1)$$

Definition 3 (Best diffusion). *The diffusion is denoted as the best diffusion when the branch number*

$$\mathcal{B}(\theta) = n + 1. \quad (2)$$

If the mapping θ is a matrix M, then M is called an MDS matrix.

When the input $\mathbf{x}$'s hamming weight is equal to 1, and $w_h(\theta(\mathbf{x})) \leq n$, the diffusion matrix's maximum branch number is $n + 1$. Reference [11] gives a property to find MDS matrix: A matrix M is MDS if and only if every square submatrix of M is nonsingular.

2.2. Hadamard Matrix

Hadamard matrix is a specially structured matrix. Let $H_{k,k} = (h_{i,j})$ be a $k \times k$ Hadamard matrix, where $k = 2^r, r = 1, 2, \cdots, (h_{i,j})$ is the (i, j)-th element and $0 \leq i, j \leq k-1$, then $h_{i,j} = h_{i\oplus j}$ holds, $\oplus$ denotes the bit-wise XOR, $h_0, h_1, \cdots, h_{k-1}$ are the elements in the first row. Hence, the elements in $H_{k,k}$ are determined by its first row, so the Hadamard matrix H can also be denoted by $\mathsf{had}(h_0, \cdots, h_{k-1})$. An example of 4×4 Hadamard matrix $\mathsf{had}(h_0, h_1, h_2, h_3)$ is shown as

$$H_{4,4} = \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_1 & h_0 & h_3 & h_2 \\ h_2 & h_3 & h_0 & h_1 \\ h_3 & h_2 & h_1 & h_0 \end{pmatrix}.$$

Definition 4 (Involutory Matrix [20]). *A $k\times k$ square matrix H is called an involutory matrix if $H^2 = I_{k,k}$, where $I_{k,k}$ is the $k \times k$ identity matrix.*

The advantage in applying involutory MDS matrices is its free inverse implementation.

2.3. Subquadratic TMVP Formulae

A matrix is denoted as a Toeplitz matrix if the elements on the line parallel to the main diagonal are constant. An example of 4×4 Toeplitz matrix is

$$T = \begin{pmatrix} t_0 & t_1 & t_2 & t_3 \\ t_4 & t_0 & t_1 & t_2 \\ t_5 & t_4 & t_0 & t_1 \\ t_6 & t_5 & t_4 & t_0 \end{pmatrix}.$$

Definition 5 (Subquadratic TMVP formulae [14]). *Let T be a $k \times k$ Toeplitz matrix and V be a column vector of dimension k. The Toeplitz matrix-vector product TV could be computed by*

$$TV = \begin{pmatrix} T_1 & T_0 \\ T_2 & T_1 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \end{pmatrix} = \begin{pmatrix} P_0 + P_2 \\ P_1 + \underline{P_2} \end{pmatrix}, \quad (3)$$

where

$$\begin{cases} P_0 = (T_0 + T_1)V_1 \\ P_1 = (T_1 + T_2)V_0 \\ P_2 = T_1(V_0 + V_1) \end{cases} .$$

Here T_0, T_1, T_2 are the sub-matrices of T with size $(k/2)\times(k/2)$. V_0, V_1 are the column sub-vectors of V with size $(k/2)\times 1$. P_0, P_1, P_2 are three TMVPs of dimension $k/2$.

Straightforward computation of TV requires 4 TMVPs of size $k/2$, while (3) saves 1 TMVP (underlined $\underline{P_2}$).

2.4. XOR Count

In this paper we use the number of XOR gates needed (XOR count) for implementing the MDS matrix-vector multiplication to measure the lightweightness of a given MDS matrix. XOR counts was first proposed in [27] and has been widely adopted to quantify the hardware implementation area cost [37, 24].

For a $k \times k$ Hadamard matrix H, there are 2 metrics of XOR count:

- The XOR count needed for directly computing the multiplication of H with a vector is called direct-XOR-count, denoted by $\mathcal{D}(H)$. $\mathcal{D}(H)$ is an overestimation of the hardware implementation cost.
- For computing the matrix-vector product, some repeated computation (XOR gates) can be reused. The direct-XOR-count minus the reused XOR count is denoted as $\mathcal{S}(H)$ [24]. However, minimizing $\mathcal{S}(H)$ is shown to be NP-hard [7]. We find the 2-input repeated XOR gates by hand.

Consider the XOR count of a given $k \times k$ Hadamard matrix. Let $\mathcal{D}(h_i)$ denote the XOR count of the element h_i. If we directly calculate matrix-vector multiplication, we will cost k^2 multiplications and $k(k-1)$ additions. Therefore the direct-XOR-count over $\mathrm{GF}(2^c)/p(x)$ could be calculated as

$$\mathcal{D}(H_{k,k}) = k \times \sum_{i=0}^{k-1} \mathcal{D}(h_i) + k \times (k-1) \times c.$$

Take matrix $E =$ `had(0x1,0x2,0x8,0xa)` and vector $V = (v_0, v_1, v_2, v_3)^\top$ over $\mathrm{GF}(2^4)/$`0x13` as an example. Firstly, we should calculate the XOR count of each element in the matrix. Let $v_k = \sum_{i=0}^{3} v_k^i x^i$ for $0 \le k \le 3$. For element `0xa`, `0xa` $\cdot\, v_3$ is equal to

$$\begin{aligned} \texttt{0xa} \cdot v_3 =& (x^3 + x) \times (v_3^3 x^3 + v_3^2 x^2 + v_3^1 x + v_3^0) \\ =& (v_3^0 v_3^2 v_3^3)x^3 + (v_3^1 v_3^2 v_3^3)x^2 + (v_3^0 v_3^1 v_3^2 v_3^3)x + (v_3^1 v_3^3). \end{aligned}$$

As there is no AND operation, we omit the XOR mark $\oplus$ in the expression without ambiguity. For example, $v_3^0 v_3^2$ here denotes $v_3^0 \oplus v_3^2$. So $\mathcal{D}(10) = 2 + 2 + 3 + 1 = 8$. The XOR counts of the other elements in the matrix over all three finite fields $\mathrm{GF}(2^4))/p(x)$ can be computed similarly and are shown in Table 2. We acquire $\mathcal{D}(h_0) = 0$, $\mathcal{D}(h_1) = 1$, $\mathcal{D}(h_2) = 3$, $\mathcal{D}(h_3) = 8$, and hence

$$\mathcal{D}(E) = 4 \times (0 + 1 + 3 + 8) + 48 = 96.$$

Table 2
XOR Counts of Elements over $\mathrm{GF}(2^4)$

Elements	$\mathrm{GF}(2^4)/$`0x13`	$\mathrm{GF}(2^4)/$`0x1f`	$\mathrm{GF}(2^4)/$`0x19`
`0x1`	0	0	0
`0x2`	1	3	1
`0x3`	5	5	3
`0x4`	2	3	3
`0x5`	6	5	5
`0x6`	5	6	2
`0x7`	9	6	6
`0x8`	3	3	6
`0x9`	1	5	8
`0xa`	8	6	5
`0xb`	6	6	9
`0xc`	5	6	1
`0xd`	3	6	5
`0xe`	8	5	6
`0xf`	6	3	8

The expression of the straightforward matrix-vector product is shown in (4) and $r_k = \sum_{i=0}^{3} r_k^i x^i$, $0 \le k \le 3$, it requires 96 XOR gates. The 2-bit combination underlined are the shared 2-input XOR gates, e.g., $v_2^0 v_2^3$ appears both in r_0^3 and r_1^3. There are 18 shared XOR gates, such that

$$\mathcal{S}(E) = 96 - 18 = 78.$$

$$\begin{aligned} r_0 =& [v_2^0 v_2^3 v_0^3 v_1^2 v_3^0 v_3^2 v_3^3]x^3 + [v_0^2 v_1^1 v_2^2 v_2^3 v_3^2 v_3^3 v_3^1]x^2 + \\ & [v_1^0 v_1^3 v_2^1 v_2^2 v_3^1 v_3^2 v_3^0 v_0^1 v_3^3]x + [v_0^0 v_1^3 v_2^1 v_3^1 v_3^3], \\ r_1 =& [v_3^0 v_3^3 \underline{v_2^0 v_2^3} v_0^2 v_1^3 v_2^2]x^3 + [(\underline{v_2^2 v_2^3})(\underline{v_3^2 v_3^3}) v_0^1 v_1^2 v_2^1]x^2 + \\ & [v_1^1 v_2^0 (\underline{v_0^0 v_0^3})(\underline{v_2^1 v_2^2})(\underline{v_3^1 v_3^2}) v_2^3]x + [v_2^3 v_3^1 v_0^3 v_1^0 v_2^1], \\ r_2 =& [v_0^0 v_0^3 v_1^2 v_1^3 v_1^0 v_2^3 v_3^2]x^3 + [(\underline{v_0^2 v_1^1})(\underline{v_1^2 v_1^3}) v_0^3 v_2^2 v_3^1]x^2 + \\ & [v_0^1 v_0^2 (\underline{v_1^0 v_1^3})(\underline{v_3^0 v_3^3}) v_1^1 v_1^2 v_2^1]x + [v_1^3 v_3^3 \underline{v_1^1 v_2^0} v_0^1], \\ r_3 =& [\underline{v_0^0 v_0^3} v_0^2 v_1^0 v_2^2 \underline{v_1^3 v_3^3}]x^3 + [\underline{v_0^1 v_0^2} v_0^3 v_2^1 v_3^2 \underline{v_1^2 v_1^3} v_0^3 v_2^1 v_3^2]x^2 + \\ & [\underline{v_0^1 v_0^2} v_0^3 v_1^1 \underline{v_2^3 v_3^1} v_0^0 v_1^2 v_2^0]x + [\underline{v_0^3 v_1^1} v_0^1 v_2^3 v_3^0]. \end{aligned} \tag{4}$$

3. Construction Algorithms

In this section, we will introduce our construction algorithms in detail. We firstly introduce the subquadratic Hadamard matrix-vector formulae and its application to 4×4 and 8×8 Hadamard matrices, followed by the construction algorithms of involutory and non-involutory MDS matrices. Involutory and non-involutory matrices are called involutions and non-involutions respectively for short.

3.1. Subquadratic HMVP Formulae

In this part, we show that the Hadamard matrices are applicable for TMVP formulae.

Lemma 1. *A $k \times k$ Hadamard matrix H could be denoted as*

$$H = \begin{pmatrix} H_1 & H_0 \\ H_0 & H_1 \end{pmatrix},$$

where H_0 and H_1 are $(k/2)\times(k/2)$ Hadamard sub-matrices.

Proof 1. *Denote*

$$H = \begin{pmatrix} H_0 & H_1 \\ H_2 & H_3 \end{pmatrix},$$

where H_0, H_1, H_2 and H_3 are $(k/2) \times (k/2)$ sub-matrices. Denote $H_0 = (h_{i,j})$, $0 \le i, j < k/2$. Then $H_1 = (h_{i,j+(k/2)})$, $H_2 = (h_{i+(k/2),j})$, $H_3 = (h_{i+(k/2),j+(k/2)})$. The elements of H_0, H_1, H_2 and H_3 satisfy $h_{i,j} = h_{i\oplus j}$, so they are Hadamard matrices. Meanwhile we have

$$i \oplus j = (i + (k/2)) \oplus (j + (k/2)),$$

and

$$i \oplus (j + (k/2)) = (i + (k/2) \oplus j),$$

as a result,

$$H_0 = H_3,$$
$$H_1 = H_2.$$

Lemma 2. *Given a $k \times k$ Hadamard matrix H of the form*

$$H = \begin{pmatrix} H_1 & H_0 \\ H_0 & H_1 \end{pmatrix},$$

where H_0 and H_1 are $(k/2)\times(k/2)$ sub-matrices, then $H_0 + H_1$ is a Hadamard matrix.

Proof 2. *Denote $H_1 = (h_{i,j})$, $H_0 = (h_{i,q})$, $H_{01} = H_0 + H_1 = \mathsf{had}(\theta_0, \cdots, \theta_{k/2-1}) = (\theta_{i,j})$, where $0 \le i, j \le k/2 - 1$, $q = j + k/2$. We know that $h_{i,j} = h_{i\oplus j}$, $h_{i,q} = h_{i\oplus q}$, and $\theta_{i\oplus j} = h_{i\oplus j} + h_{i\oplus q}$, $\theta_{i,j} = h_{i,j} + h_{i,q}$, so $\theta_{i\oplus j} = \theta_{i,j}$ and H_{01} is a Hadamard matrix.*

Definition 6 (Subquadratic HMVP Formulae). *Based on Lemma 1 and Lemma 2, we can compute the subquadratic Hamamard matrix-vector product (HMVP) of H and V by*

$$HV = \begin{pmatrix} H_1 & H_0 \\ H_0 & H_1 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \end{pmatrix} = \begin{pmatrix} P_0 + P_2 \\ P_1 + \underline{P_2} \end{pmatrix},$$

where

$$\begin{cases} P_0 = (H_0 + H_1)V_1 \\ P_1 = (H_1 + H_0)V_0 \\ P_2 = H_1(V_0 + V_1) \end{cases}.$$

Here, P_0, P_1 and P_2 are HMVPs of dimension $k/2$.

There are $2k - 1$ independent elements which are in the first row and first column of a Toeplitz matrix while there are k independent of a Hadamard matrix. Hence, constructing Hadamard matrices needs smaller search space than Toeplitz matrices.

3.2. Application of HMVP to 4×4 Hadamard Matrices

We firstly apply the HMVP formulae on 4×4 Hadamard matrices. According to the structure of Hadamard matrices, $H_{4,4}$ is in the form of

$$H_{4,4} = \mathsf{had}(h_0, h_1, h_2, h_3) = \begin{pmatrix} H_1 & H_0 \\ H_0 & H_1 \end{pmatrix},$$

where

$$H_1 = \begin{pmatrix} h_0 & h_1 \\ h_1 & h_0 \end{pmatrix}, \quad H_0 = \begin{pmatrix} h_2 & h_3 \\ h_3 & h_2 \end{pmatrix}.$$

The vector V is in the form of

$$V = \begin{pmatrix} V_0 \\ V_1 \end{pmatrix},$$

where

$$V_0 = \begin{pmatrix} v_0 \\ v_1 \end{pmatrix}, \quad V_1 = \begin{pmatrix} v_2 \\ v_3 \end{pmatrix}.$$

By applying the HMVP formulae, the three corresponding HMVPs could be computed by

$$P_0 = (H_0 + H_1)V_1 = \begin{pmatrix} h_{02}v_2 + h_{13}v_3 \\ h_{13}v_2 + h_{02}v_3 \end{pmatrix},$$
$$P_1 = (H_1 + H_0)V_0 = \begin{pmatrix} h_{02}v_0 + h_{13}v_1 \\ h_{13}v_0 + h_{02}v_1 \end{pmatrix},$$
$$P_2 = H_1(V_0 + V_1) = \begin{pmatrix} h_0v_{02} + h_1v_{13} \\ h_1v_{02} + h_0v_{13} \end{pmatrix}.$$

Finally the matrix-vector product $R_4 = H_{4,4}V$ could be computed by

$$R_4 = \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} P_0 + P_2 \\ P_1 + \underline{P_2} \end{pmatrix}$$
$$= \begin{pmatrix} h_{02}v_2 + h_{13}v_3 + h_0v_{02} + h_1v_{13} \\ h_{13}v_2 + h_{02}v_3 + h_1v_{02} + h_0v_{13} \\ h_{02}v_0 + h_{13}v_1 + \underline{h_0v_{02} + h_1v_{13}} \\ h_{13}v_0 + h_{02}v_1 + \underline{h_1v_{02} + h_0v_{13}} \end{pmatrix}.$$

where h_{ij} denotes $h_i + h_j$ and v_{ij} denotes $v_i + v_j$.

According to above computation, when we directly compute the matrix-vector multiplication on 4×4 matrix, there needs 12 addition operations and 16 multiplication operations over the finite field.

In order to implement the multiplication of $H_{4,4}$ with a vector, $H_0 + H_1$ can be computed in advance. $V_0 + V_1$ costs 2 addition operations. By applying the HMVP formulae, 10 addition operations and 12 multiplication operations are needed in R_4, so the total number of addition operations is $10 + 2 = 12$. 12 Multiplication operations all appear in R_4. In summary, we economize 4 multiplication operations, which helps us to reduce the space complexity in hardware implementation.

Here we give a new observation of HMVP on XOR count and define two new metrics:

- The number of XOR gates by directly applying HMVP formulae is denoted as $\mathcal{H}(H)$.
- The number of XOR gates $\mathcal{H}(H)$ minus the repeated XOR gates is denoted as $\mathcal{HS}(H)$.

From the expression of R_4 we can obtain the XOR count of 4×4 Hadamard matrix by applying the HMVP formulae over GF(2^c):

$$\mathcal{H}(H_{4,4}) = 4(\mathcal{D}(h_{02}) + \mathcal{D}(h_{13})) + 2(\mathcal{D}(h_0) + \mathcal{D}(h_1)) + 12c. \tag{5}$$

Look back to the example $E = \mathsf{had}(\texttt{0x1,0x2,0x8,0xa})$, if we apply the HMVP formulae on E, $h_{02} = \texttt{0x1} \oplus \texttt{0x8} = \texttt{0x9}$, $h_{13} = \texttt{0x2} \oplus \texttt{0xa} = \texttt{0x8}$, so $\mathcal{D}(h_{02}) = 1$, $\mathcal{D}(h_{13}) = 3$. We could compute

$$\mathcal{H}(E) = 4\times(1+3) + 2\times(0+1) + 12\times 4 = 66.$$

The implementation of HMVP based matrix-vector product is shown in Section A.1. There are 8 shared XOR gates (underlined) totally, and hence

$$\mathcal{HS}(E) = 66 - 8 = 58.$$

3.3. Application of HMVP to 8×8 Hadamard Matrices

In this section, we apply the HMVP to 8×8 Hadamard matrices.

According to Lemma 1, $H_{8,8}$ is in the form of

$$H_{8,8} = \mathsf{had}(h_0,\dots,h_7) = \begin{pmatrix} H_1 & H_0 \\ H_0 & H_1 \end{pmatrix},$$

where $H_1 = \mathsf{had}(h_0,h_1,h_2,h_3)$ and $H_0 = \mathsf{had}(h_4,h_5,h_6,h_7)$. The vector V is in the form of

$$V = \begin{pmatrix} V_0 \\ V_1 \end{pmatrix},$$

where $V_0 = (v_0,v_1,v_2,v_3)^\top$ and $V_1 = (v_4,v_5,v_6,v_7)^\top$.

By applying the HMVP formulae, the three corresponding HMVPs of $H_{8,8}V$, i.e., P_0, P_1 and P_2, need to be computed. Firstly, P_0 can be computed by

$$P_0 = (H_0 + H_1)V_1 = \begin{pmatrix} h_{04} & h_{15} & h_{26} & h_{37} \\ h_{15} & h_{04} & h_{37} & h_{26} \\ h_{26} & h_{37} & h_{04} & h_{15} \\ h_{37} & h_{26} & h_{15} & h_{04} \end{pmatrix}\begin{pmatrix} v_4 \\ v_5 \\ v_6 \\ v_7 \end{pmatrix}.$$

According to Lemma 2, $H_0 + H_1$ is also a Hadamard matrix. And hence, the HMVP formulae can be applied for computing the matrix-vector product $P_0 = (H_0 + H_1)V_1$. Rewrite $H_0 + H_1$ as

$$H_0 + H_1 = \begin{pmatrix} H_1^{P_0} & H_0^{P_0} \\ H_0^{P_0} & H_1^{P_0} \end{pmatrix},$$

where $H_0^{P_0} = \mathsf{had}(h_{26}, h_{37})$, $H_1^{P_0} = \mathsf{had}(h_{04}, h_{15})$. Let V_1 be

$$V_1 = \begin{pmatrix} V_0^{P_0} \\ V_1^{P_0} \end{pmatrix},$$

where $V_0^{P_0} = (v_4, v_5)^\top$, $V_1^{P_0} = (v_6, v_7)^\top$. Then P_0 can be computed by

$$P_0 = \begin{pmatrix} H_1^{P_0} & H_0^{P_0} \\ H_0^{P_0} & H_1^{P_0} \end{pmatrix}\begin{pmatrix} V_0^{P_0} \\ V_1^{P_0} \end{pmatrix}$$

Denote three corresponding HMVPs of P_0 as A_0, A_1, A_2, which can be computed according to (3), i.e.,

$$A_0 = (H_0^{P_0} + H_1^{P_0})V_1^{P_0} = \begin{pmatrix} h_{0246}v_6 + h_{1357}v_7 \\ h_{1357}v_6 + h_{0246}v_7 \end{pmatrix},$$

$$A_1 = (H_1^{P_0} + H_0^{P_0})V_0^{P_0} = \begin{pmatrix} h_{0246}v_4 + h_{1357}v_5 \\ h_{1357}v_4 + h_{0246}v_5 \end{pmatrix},$$

$$A_2 = H_1^{P_0}(V_0^{P_0} + V_1^{P_0}) = \begin{pmatrix} h_{04}v_{46} + h_{15}v_{57} \\ h_{15}v_{46} + h_{04}v_{57} \end{pmatrix}.$$

Then, P_0 can be computed accordingly by

$$P_0 = \begin{pmatrix} A_0 + A_2 \\ A_1 + \underline{A_2} \end{pmatrix} = \begin{pmatrix} h_{0246}v_6 + h_{1357}v_7 + h_{04}v_{46} + h_{15}v_{57} \\ h_{1357}v_6 + h_{0246}v_7 + h_{15}v_{46} + h_{04}v_{57} \\ h_{0246}v_4 + h_{1357}v_5 + \underline{h_{04}v_{46} + h_{15}v_{57}} \\ h_{1357}v_4 + h_{0246}v_5 + \underline{h_{15}v_{46} + h_{04}v_{57}} \end{pmatrix}.$$

By similar derivation process, $B_0 = (H_0^{P_1} + H_1^{P_1})V_1^{P_1}$, $B_1 = (H_1^{P_1} + H_0^{P_1})V_0^{P_1}$, $B_2 = H_1^{P_1}(V_0^{P_1} + V_1^{P_1})$, $C_0 = (H_0^{P_2} + H_1^{P_2})V_1^{P_2}$, $C_1 = (H_1^{P_2} + H_0^{P_2})V_0^{P_2}$, $C_2 = H_1^{P_2}(V_0^{P_2} + V_1^{P_2})$, the other two HMVPs of $H_{8,8}V$, i.e., P_1 and P_2 are

$$P_1 = \begin{pmatrix} B_0 + B_2 \\ B_1 + \underline{B_2} \end{pmatrix} = \begin{pmatrix} h_{0246}v_2 + h_{1357}v_3 + h_{04}v_{02} + h_{15}v_{13} \\ h_{1357}v_2 + h_{0246}v_3 + h_{15}v_{02} + h_{04}v_{13} \\ h_{0246}v_0 + h_{1357}v_1 + \underline{h_{04}v_{02} + h_{15}v_{13}} \\ h_{1357}v_0 + h_{0246}v_1 + \underline{h_{15}v_{02} + h_{04}v_{13}} \end{pmatrix},$$

and

$$P_2 = \begin{pmatrix} C_0 + C_2 \\ C_1 + \underline{C_2} \end{pmatrix} = \begin{pmatrix} h_{02}v_{26} + h_{13}v_{37} + h_0v_{0246} + h_1v_{1357} \\ h_{13}v_{26} + h_{02}v_{37} + h_1v_{0246} + h_0v_{1357} \\ h_{02}v_{04} + h_{13}v_{15} + \underline{h_0v_{0246} + h_1v_{1357}} \\ h_{13}v_{04} + h_{02}v_{15} + \underline{h_1v_{0246} + h_0v_{1357}} \end{pmatrix}.$$

Finally the matrix-vector product $R_8 = H_{8,8}V$ is

$$R_8 = \begin{pmatrix} P_0 + P_2 \\ P_1 + \underline{P_2} \end{pmatrix} = \begin{pmatrix} A_0 + A_2 + C_0 + C_2 \\ A_1 + \underline{A_2} + C_1 + \underline{C_2} \\ B_0 + B_2 + \underline{C_0 + C_2} \\ B_1 + \underline{B_2} + \underline{C_1 + C_2} \end{pmatrix}.$$

A direct calculation of 8×8 HMVP needs 56 addition operations and 64 multiplication operations. By using the above HMVP formulae, R_8 consists of $P_0 + P_2$ and $P_1 + \underline{P_2}$, P_0, P_1, P_2 each cost 10 addition operations, $\underline{P_2}$ costs free. And P_0, P_1, P_2 are vectors with 4×1, so $P_0 + P_2$ and $P_1 + \underline{P_2}$ each cost 4 addition operations. 10 addition operations are needed in v_{02}, v_{04}, v_{13}, v_{15}, v_{26}, v_{46}, v_{37}, v_{57}, v_{0246}, v_{1357}. So the total number of addition operations is $10 \times 3 + 4 \times 2 + 10 = 48$. Meanwhile, in R_8, P_0, P_1, P_2 each cost 12 multiplication operations, so R_8 needs 36 multiplication operations totally. In summary, we economize 8 addition operations and 28 multiplication operations.

Here, by counting the frequency of elements in R_8 over $\mathrm{GF}(2^c)$ we generalize

$$\begin{aligned}\mathcal{H}(H_{8,8}) =& 8(\mathcal{D}(h_{0246}) + \mathcal{D}(h_{1357})) + 2(\mathcal{D}(h_0) + \mathcal{D}(h_1)) \\ &+ 4(\mathcal{D}(h_{04}) + \mathcal{D}(h_{15}) + \mathcal{D}(h_{02}) + \mathcal{D}(h_{13})) + 48c.\end{aligned} \tag{6}$$

3.4. Searching for Involutory MDS Matrices

If a Hadamard matrix $H_{k,k}$ is an involutory MDS matrix over $\mathrm{GF}(2^c)/p(x)$, the following three constraints hold [8]:

- $h_i \neq h_j$ for $i \neq j$, where $0 \leq i, j \leq k - 1$;
- $h_i \neq 0$, where $0 \leq i \leq k - 1$;
- $\sum_{i=0}^{k-1} h_i = 1$.

We use these involutory constraints to reduce the search space of the Hadamard MDS matrices.

In this part we introduce a bottom-up construction strategy for involutory MDS matrices. Combined with the HMVP formulae, we identify an exhausted search strategy according to (5) and (6).

3.4.1. *Construct* 4×4 *Involutory MDS Matrices*

According to (5), which illustrates the XOR count of 4×4 Hadamard matrices by applying HMVP, the XOR count can be minimized if $\mathcal{D}(h_{02}) + \mathcal{D}(h_{13})$ and $\mathcal{D}(h_0) + \mathcal{D}(h_1)$ can be minimized. So we can combine lightweight elements into a group, and assign these elements to h_{02}, h_0, h_1.

Firstly, we compute the XOR count of each possible element and choose s most lightweight elements as the candidate set $\mathcal{G}$, so the search space over $\mathrm{GF}(2^c)$ is determined by s. As there is no zero element in a Hadamard matrix, $s \leq 2^c - 1$. To construct a 4×4 involutory Hadamard matrix, we search h_0, h_1, h_{02} from $\mathcal{G}$. Due to the constraint $h_0 + h_1 + h_2 + h_3 = 1$ for involution, $h_{13} = h_{02} + 1$, and hence $h_3 = h_{13} + h_1 = h_{02} + h_1 + 1$. Combined with $h_2 = h_{02} + h_0$, all elements of the Hadamard matrix can be determined. The whole construction algorithm is shown in Algorithm 1.

Algorithm 1 Construct 4×4 Involutory MDS Matrices.

Input: The finite field $\mathrm{GF}(2^c)/p(x)$, and the candidate $\mathcal{G}$ with the corresponding size s.

Output: A series of 4×4 involutions over $\mathrm{GF}(2^c)/p(x)$.

```
1: for each $h_0, h_1, h_{02}$ from $\mathcal{G}$ do
2:     $h_2 \leftarrow h_{02} + h_0$
3:     $h_{13} \leftarrow h_{02} + 1$
4:     $h_3 \leftarrow h_{13} + h_1$
5:     $H_{4,4} \leftarrow \mathsf{had}(h_0, h_1, h_2, h_3)$
6:     if $H_{4,4}$ is an involutory MDS matrix then
7:         Save $H_{4,4}$ and calculate $\mathcal{D}(H_{4,4})$ and $\mathcal{H}(H_{4,4})$.
8:     end if
9: end for
```

The complexity of Algorithm 1 is

$$T_1 = O(s^3).$$

If we search all matrices over $\mathrm{GF}(2^4)$, the total space is $15^3 = 3,375 \approx 2^{11.7}$ matrices. And for finite field $\mathrm{GF}(2^8)/p(x)$, there are 255 non-zeros elements, the search space is $255^3 = 16,581,375$. The whole search space of 4×4 involutory matrices is under the computing ability of a single PC.

3.4.2. *Construct* 8×8 *Involutory MDS Matrices*

Continue to use the similar method, according to (6), which illustrates the XOR count of 8×8 HMVP, the XOR count can be minimized if $\mathcal{D}(h_{0246}) + \mathcal{D}(h_{1357})$, $\mathcal{D}(h_0) + \mathcal{D}(h_1)$ and $\mathcal{D}(h_{04}) + \mathcal{D}(h_{15}) + \mathcal{D}(h_{02}) + \mathcal{D}(h_{13})$ can be minimized.

Due to the constraint $\sum_{i=0}^{7} h_i = 1$, $h_{1357} = h_{0246} + 1$, we should search only h_0, h_1, h_{02}, h_{13}, h_{04}, h_{15}, h_{0246} from $\mathcal{G}$, and all elements of $H_{8,8}$ can be determined as $h_2 = h_{02} + h_0$, $h_3 = h_{13} + h_1$, $h_4 = h_{04} + h_0$, $h_5 = h_{15} + h_1$, $h_6 = h_{0246} + h_{02} + h_4$, $h_7 = h_{1357} + h_{13} + h_5 = h_{0246} + 1 + h_{13} + h_5$. The whole construction algorithm is shown in Algorithm 2.

The complexity of Algorithm 2 is

$$T_2 = O(s^7).$$

For finite field $\mathrm{GF}(2^4)$, the whole search space is $15^7 = 170,859,375$. For finite field $\mathrm{GF}(2^8)$, the whole search space is $255^7 = 70,110,209,207,109,375$.

In order to reduce the search space, s can be smaller, i.e., $\mathcal{G}$ can be initialized with the top s most lightweight elements, which will be detailed in Sec. 4.

3.5. Searching for Non-involutory MDS Matrices

For 4×4 non-involutions, we should search h_0, h_1, h_{02}, h_{13} from $\mathcal{G}$. Elements h_2 and h_3 can be determined as follows: $h_2 = h_{02} + h_0$, and $h_3 = h_{13} + h_1$. The whole construction algorithm of constructing 4×4 non-involutory MDS matrices is similar with Algorithm 1. The complexity is $O(s^4)$.

Algorithm 2 Construct 8×8 Involutory MDS Matrices.

Input: The finite field $GF(2^c)/p(x)$, and the candidate $\mathcal{G}$ with the corresponding size s.

Output: A series of 8×8 involutions over $GF(2^c)/p(x)$.

```
1: for each $h_0, h_1, h_{02}, h_{04}, h_{13}, h_{15}, h_{0246}$ from $\mathcal{G}$ do
2:     $h_2 \leftarrow h_{02} + h_0$
3:     $h_3 \leftarrow h_{13} + h_1$
4:     $h_4 \leftarrow h_{04} + h_0$
5:     $h_5 \leftarrow h_{15} + h_1$
6:     $h_6 \leftarrow h_{0246} + h_0 + h_2 + h_4$
7:     $h_{1357} \leftarrow h_{0246} + 1$
8:     $h_7 \leftarrow h_{1357} + h_1 + h_3 + h_5$
9:     $H_{8,8} \leftarrow \mathsf{had}(h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7)$
10:    if $H_{8,8}$ is an involutory MDS matrix then
11:        Save $H_{8,8}$ and calculate $\mathcal{D}(H_{8,8})$ and $\mathcal{H}(H_{8,8})$.
12:    end if
13: end for
```

For 8×8 non-involutions, we should search h_0, h_1, h_{02}, h_{13}, h_{04}, h_{15}, h_{0264}, h_{1357} from $\mathcal{G}$. And all elements can be determined as $h_2 = h_{02} + h_0$, $h_3 = h_{13} + h_1$, $h_4 = h_{04} + h_0$, $h_5 = h_{15} + h_1$, $h_6 = h_{0246} + h_{02} + h_4$, $h_7 = h_{1357} + h_{13} + h_5$. The whole construction algorithm is similar as Algorithm 2. The complexity is $O(s^8)$.

4. Experiment results

We construct MDS matrices over all three finite fields ($GF(2^4)/$`0x13`, $GF(2^4)/$`0x1f`, $GF(2^4)/$`0x19`).

According to the analysis in Sec. 3.4.1 and Sec. 3.5, the total search space for 4×4 involutory and non-involutory matrices are s^3 and s^4 respectively, such that the full search needs $15^3 \approx 2^{11.7}$ and $15^4 \approx 2^{15.6}$, so we set $s = 15$ for 4×4 matrices and search full space.

For 8×8 matrices, the total search space are s^7 and s^8 for 8×8 involutory and non-involutory matrices, according to Sec. 3.4.2 and Sec. 3.5 respectively. In order to reduce the search space, we fix $h_0 = 1$, $h_1 = 2$, such that the search space for involutory and non-involutory matrices are s^5 and s^6 respectively. For $s = 15$, the search space is $15^5 \approx 2^{19.53}$ and $15^6 \approx 2^{23.44}$, which is within the computing ability of a single PC. There are lightweight non-involutions while there is no such involution with this setting. We fix $s = 10$ for constructing 8×8 involutions to select an appropriate value for h_{0246}. By heuristic attempt, finally we fix $h_{0246} = 10$ and go through h_0 and h_1. In order to further reduce the search space, the size s is reduced to 10, i.e., $\mathcal{G}$ contains the top 10 most lightweight elements. The values of h_{0246} and h_{1357} are specified, such that the search space for constructing 8×8 involutions is $10^6 \approx 2^{19.93}$.

All experimental parameters and results are shown in Table 3. **Num.** is used to record the number of matrices and **Max.** and **Min.** are the maximum and minimum XOR count $\mathcal{H}(H)$ respectively. The detailed information of the matrices are as follows.

- 4×4 **involutions:** We find 1512 MDS involutory MDS matrices, of which the smallest $\mathcal{H}(H)$ is 66. There are 8 MDS matrices with XOR count 66 by applying the HMVP formulae, of which 4 matrices are over $GF(2^4)/$`0x13` and 4 matrices are over $GF(2^4)/$`0x19`.

  The matrices over $GF(2^4)/$`0x13` are

  ```
  had(0x1,0x2,0x9,0xb), had(0x1,0x2,0x8,0xa),
  had(0x2,0x1,0xa,0x8), had(0x2,0x1,0xb,0x9).
  ```

  The matrices over $GF(2^4)/$`0x19` are

  ```
  had(0x1,0xc,0x3,0xf), had(0x1,0xc,0x2,0xe),
  had(0xc,0x1,0xe,0x2), had(0xc,0x1,0xf,0x3).
  ```

  We choose `had(0x1,0x2,0x8,0xa)` over $GF(2^4)/$`0x13` as the 4×4 involutory candidate I_4, and hence we can obtain $\mathcal{HS}(I_4) = 58$ according to the implementation in Section A.1.

- 4×4 **non-involutions:** We find 21168 non-involutory MDS matrices, of which the smallest $\mathcal{H}(H)$ is 56. There are 16 MDS matrices with $\mathcal{H}(H)$ 56 by applying the HMVP formulae, of which 8 matrices are over $GF(2^4)/$`0x13` and 8 matrices are over $GF(2^4)/$`0x19`.

  The matrices over $GF(2^4)/$`0x13` are

  ```
  had(0x1,0x4,0x3,0x5), had(0x1,0x4,0x8,0x5),
  had(0x2,0x9,0x3,0xb), had(0x2,0x9,0xb,0x8),
  had(0x4,0x1,0x5,0x3), had(0x4,0x1,0x5,0x8),
  had(0x9,0x2,0x8,0xb), had(0x9,0x2,0xb,0x3).
  ```

  The matrices over $GF(2^4)/$`0x19` are

  ```
  had(0x1,0x6,0x3,0x7), had(0x1,0x6,0xd,0x7),
  had(0x2,0xc,0x3,0xe), had(0x2,0xc,0xe,0xd),
  had(0x6,0x1,0x7,0x3), had(0x6,0x1,0x7,0xd),
  had(0xc,0x2,0xd,0xe), had(0xc,0x2,0xe,0x3).
  ```

  We choose `had(0x1,0x4,0x3,0x5)` over $GF(2^4)/$`0x13` as the 4×4 non-involutory MDS candidate N_4, where we find 4 shared XOR gates, such that $\mathcal{HS}(N_4) = 56-4 = 52$. The detailed matrix-vector product is shown in Section A.2.

- 8×8 **involutions:** We totally find 22 8×8 involutions. There are 2 matrices over $GF(2^4)/$`0x19` cost 344 XOR gates, which is the smallest $\mathcal{H}(H)$ of them. They are:

  ```
  had(0xc,0x5,0xd,0x6,0x8,0x7,0x3,0xf),
  had(0xc,0x5,0x8,0x7,0xd,0x6,0x3,0xf).
  ```

  We choose `had(0xc,0x5,0x8,0x7,0xd,0x6,0x3,0xf)` as the 8×8 involutory candidate I_8. The corresponding matrix-vector product is shown in Section A.3. We find 62 shared XOR gates, such that $\mathcal{HS}(I_8) = 344 - 62 = 282$.

Table 3
Input parameters and experiment results

	Parameter			GF(2^4)/`0x13`			GF(2^4)/`0x1f`			GF(2^4)/`0x19`		
Type	k	c	s	**Num.**	**Max.**	**Min.**	**Num.**	**Max.**	**Min.**	**Num.**	**Max.**	**Min.**
Involution	4	4	15	1512	138	66	1512	120	86	1512	138	66
Non-involution	4	4	15	21168	148	56	21168	120	66	21168	148	56
Involution	8	4	10	4	496	362	12	390	376	6	376	344
Non-involution	8	4	15	96	410	258	96	382	290	96	406	266

Table 4
Comparisons of involutory MDS matrices

Field	k	$\mathcal{D}/\mathcal{H}$	$\mathcal{S}/\mathcal{HS}$	**Inv.**	**Ref.**
GF(2^4)/`0x13`	4	72	68	✓	[23]
GF(2^4)/`0x13`	4	64	64	✓	[33]
GF(2^4)/`0x13`	4	68	63	✓	[24]
GF(2^4)/`0x13`	4	66	**58**	✓	I_4
GF(2^4)/`0x19`	4	58	58		[33]
GF(2^4)/`0x13`	4	58	58		[24]
GF(2^4)/`0x19`	4	56	**52**		N_4
GF(2^4)/`0x13`	8	512	424	✓	[37],[24]
GF(2^4)/`0x19`	8	344	**282**	✓	I_8
GF(2^4)/`0x13`	8	512	408		[29]
GF(2^4)/`0x13`	8	448	392		[29]
GF(2^4)/`0x13`	8	432	384		[24]
GF(2^4)/`0x13`	8	258	**228**		N_8

- 8×8 **non-involutions:** We find 96 8×8 non-involutions. 2 matrices over GF(2^4)/`0x13` have smallest XOR count $\mathcal{H}(H) = 258$:

  `had(0x1,0x2,0x5,0xe,0x7,0x3,0xa,0xd),`

  `had(0x1,0x2,0x7,0x3,0x5,0xe,0xa,0xd).`

  We choose `had(0x1,0x2,0x5,0xe,0x7,0x3,0xa,0xd)` as the 8×8 non-involutory candidate N_8. The corresponding matrix-vector product is shown in Section A.4. We find 30 shared XOR gates, such that $\mathcal{HS}(N_8) = 258 - 30 = 228$.

We compare our results with the previous work in Table 4. All the candidates we choose are more lightweight than the previous MDS matrices.

5. Conclusion

In this paper we propose efficient algorithms for constructing lightweight involutory and non-involutory MDS matrices. We find 4×4 and 8×8 lightweight MDS matrices with the fewest XOR counts over GF(2^4) until now. In this paper, we focus on the construction of lightweight MDS Hadamard matrices over GF(2^4). Our algorithms can be extended to construct MDS matrices over bigger finite field such as GF(2^8), which will be our future work.

Acknowledgement

This work was supported by the National Key Research and Development Program of China (No. 2018YFA0704701).

A. Matrix-vector product of the candidates

A.1. Matrix-vector product of I_4

$$P_0 = \begin{pmatrix} (v_2^0 v_3^0 v_3^3)x^3 + (v_2^3 v_3^3 v_3^2)x^2 + (v_2^2 v_3^2 v_3^1)x + (v_2^1 v_3^1 v_2^0) \\ (\underline{v_2^0 v_3^0} v_2^3)x^3 + (\underline{v_2^3 v_3^3} v_2^2)x^2 + (\underline{v_2^2 v_3^2} v_2^1)x + (\underline{v_3^1 v_2^1} v_3^0) \end{pmatrix},$$

$$P_1 = \begin{pmatrix} (v_0^0 v_1^0 v_1^3)x^3 + (v_0^3 v_1^3 v_1^2)x^2 + (v_0^2 v_1^2 v_1^1)x + (v_0^1 v_1^1 v_0^0) \\ (\underline{v_0^0 v_1^0} v_0^3)x^3 + (\underline{v_0^3 v_1^3} v_0^2)x^2 + (\underline{v_0^2 v_1^2} v_0^1)x + (\underline{v_0^1 v_1^1} v_1^0) \end{pmatrix},$$

$$P_2 = \begin{pmatrix} (v_{02}^3 v_{13}^2)x^3 + (v_{02}^2 v_{13}^1)x^2 + (v_{02}^1 v_{13}^0 v_{13}^3)x + (v_{02}^0 v_{13}^3) \\ (v_{02}^2 v_{13}^3)x^3 + (v_{02}^1 v_{13}^2)x^2 + (v_{02}^0 v_{02}^3 v_{13}^1)x + (v_{02}^3 v_{13}^0) \end{pmatrix}.$$

A.2. Matrix-vector product of N_4

$$P_0 = \begin{pmatrix} (v_2^2 v_3^3)x^3 + (v_2^1 v_3^2)x^2 + (v_2^0 v_2^3 v_3^1)x + (v_2^3 v_3^0) \\ (v_2^3 v_3^2)x^3 + (v_2^2 v_3^1)x^2 + (v_2^1 v_3^0 v_3^3)x + (v_2^0 v_3^3) \end{pmatrix},$$

$$P_1 = \begin{pmatrix} (v_0^2 v_1^3)x^3 + (v_0^1 v_1^2)x^2 + (v_0^0 v_0^3 v_1^1)x + (v_0^3 v_1^0) \\ (v_0^3 v_1^2)x^3 + (v_0^2 v_1^1)x^2 + (v_0^1 v_1^0 v_1^3)x + (v_0^0 v_1^3) \end{pmatrix},$$

$$P_2 = \begin{pmatrix} (v_{02}^3 v_{13}^1)x^3 + (v_{02}^2 v_{13}^0 v_{13}^3)x^2 + (v_{02}^1 v_{13}^3 v_{13}^2)x + (v_{02}^0 v_{13}^2) \\ (\underline{v_{02}^1 v_{13}^3})x^3 + (\underline{v_{02}^0 v_{13}^2} v_{02}^3)x^2 + (v_{02}^2 \underline{v_{02}^3 v_{13}^1})x + (\underline{v_{02}^2 v_{13}^0}) \end{pmatrix}.$$

A.3. Matrix-vector product of I_8

$$A_0 = \begin{pmatrix} (v_6^0 v_6^1 v_7^0 v_7^1 v_7^3)x^3 + (\underline{v_6^1 v_7^1} v_6^3 v_7^2 v_7^3)x^2 + \\ (\underline{v_6^0 v_7^0} v_6^2 \underline{v_6^3 v_7^3} v_7^1 v_7^2)x + (\underline{v_6^1 v_7^1} v_7^0 \underline{v_6^2 v_7^2}) \\ (\underline{v_6^0 v_7^0} v_6^3 \underline{v_6^1 v_7^1})x^3 + (\underline{v_6^1 v_7^1} v_6^2 \underline{v_6^3 v_7^3})x^2 + \\ (\underline{v_6^0 v_7^0} v_6^1 (\underline{v_6^2 v_7^2})(\underline{v_6^3 v_7^3}))x + (\underline{v_6^1 v_7^1} v_6^0 \underline{v_6^2 v_7^2}) \end{pmatrix},$$

$$A_1 = \begin{pmatrix} (v_4^0v_4^1v_5^0v_5^1v_5^3)x^3 + (v_4^1v_4^3v_5^1v_5^2v_5^3)x^2+ \\ (v_4^0v_4^2v_4^3\underline{v_5^0v_5^1}v_5^2v_5^3)x + (v_4^1(\underline{v_5^0v_5^1})(\underline{v_4^2v_5^2})) \\ (\underline{v_4^0v_4^1}v_4^3\underline{v_5^0v_5^1})x^3 + (v_4^1\underline{v_4^2v_4^3}v_5^1v_5^3)x^2+ \\ (\underline{v_4^0v_4^1}v_5^0\underline{v_4^2v_4^3}v_5^2v_5^3)x + (\underline{v_4^0v_4^1}v_5^1\underline{v_4^2v_5^2}) \end{pmatrix},$$

$$A_2 = \begin{pmatrix} (v_{46}^3v_{57}^2)x^3+(v_{46}^2v_{57}^1v_{57}^2)x^2+(v_{46}^1v_{57}^0v_{57}^1)x+(v_{46}^0v_{57}^0v_{57}^3) \\ (v_{46}^2v_{57}^3)x^3+(v_{46}^1\underline{v_{46}^2v_{57}^2})x^2+(v_{46}^0\underline{v_{46}^1v_{57}^1})x+(\underline{v_{46}^0v_{57}^0}v_{46}^3) \end{pmatrix},$$

$$B_0 = \begin{pmatrix} (v_2^0v_2^1v_3^0v_3^1v_3^3)x^3 + (v_2^1v_2^3\underline{v_3^1v_3^3}v_3^2)x^2+ \\ (v_2^0v_2^2v_2^3v_3^0\underline{v_3^1v_3^3}v_3^2)x + (v_2^1v_2^2\underline{v_3^0v_3^2}v_3^1) \\ (\underline{v_2^0v_2^1}v_2^3v_3^0v_3^1)x^3 + (v_2^1(\underline{v_2^2v_2^3})(\underline{v_3^1v_3^3}))x^2+ \\ ((\underline{v_2^0v_2^1})(\underline{v_2^2v_2^3})(\underline{v_3^0v_3^2})v_3^3)x + (\underline{v_2^0v_2^1}v_3^2\underline{v_2^2v_3^1}) \end{pmatrix},$$

$$B_1 = \begin{pmatrix} (v_0^0v_0^1v_1^0v_1^1v_1^3)x^3 + (v_0^1v_0^3\underline{v_1^1v_1^3}v_1^2)x^2+ \\ (v_0^0v_0^2v_0^3v_1^0\underline{v_1^1v_1^3}v_1^2)x + (v_0^1v_0^2v_1^1\underline{v_1^0v_1^2}) \\ (\underline{v_0^0v_0^1}v_0^3v_1^0v_1^1)x^3 + (\underline{v_0^2v_0^3}v_0^1\underline{v_1^1v_1^3})x^2+ \\ ((\underline{v_0^0v_0^1})(\underline{v_0^2v_0^3})v_1^3\underline{v_1^0v_1^2})x + (\underline{v_0^0v_0^1}v_1^2\underline{v_0^2v_1^1}) \end{pmatrix},$$

$$B_2 = \begin{pmatrix} (v_{02}^3v_{13}^2)x^3 + (v_{02}^2v_{13}^1v_{13}^2)x^2+ \\ (v_{02}^1v_{13}^0v_{13}^1)x + (v_{02}^0v_{13}^0v_{13}^3) \\ (v_{02}^2v_{13}^3)x^3 + (v_{02}^1\underline{v_{02}^2v_{13}^2})x^2+ \\ (v_{02}^0\underline{v_{02}^1v_{13}^1})x + (\underline{v_{02}^0v_{13}^0}v_{02}^3) \end{pmatrix},$$

$$C_0 = \begin{pmatrix} (v_{26}^1v_{26}^2v_{26}^3v_{37}^2v_{37}^3)x^3 + (v_{26}^0v_{37}^1)x^2+ \\ (v_{26}^3v_{37}^0)x + (\underline{v_{26}^2v_{26}^3}v_{37}^3) \\ (\underline{v_{26}^2v_{26}^3}v_{37}^1\underline{v_{37}^2v_{37}^3})x^3 + (v_{26}^1v_{37}^0)x^2+ \\ (v_{26}^0v_{37}^3)x + (v_{26}^3\underline{v_{37}^2v_{37}^3}) \end{pmatrix},$$

$$C_1 = \begin{pmatrix} (v_{04}^1v_{04}^2v_{04}^3v_{15}^2v_{15}^3)x^3 + (v_{04}^0v_{15}^1)x^2+ \\ (v_{04}^3v_{15}^0)x + (\underline{v_{04}^2v_{04}^3}v_{15}^3) \\ (\underline{v_{04}^2v_{04}^3}v_{15}^1\underline{v_{15}^2v_{15}^3})x^3 + (v_{04}^1v_{15}^0)x^2+ \\ (v_{04}^0v_{15}^3)x + (v_{04}^3\underline{v_{15}^2v_{15}^3}) \end{pmatrix},$$

$$C_2 = \begin{pmatrix} (v_{0246}^0v_{1357}^1v_{1357}^2)x^3 + (v_{0246}^0v_{0246}^3v_{1357}^0v_{1357}^2)x^2+ \\ (v_{0246}^2v_{1357}^1v_{1357}^3)x + (v_{0246}^1\underline{v_{1357}^0v_{1357}^2}v_{1357}^3) \\ (v_{0246}^1v_{0246}^2v_{1357}^0)x^3 + (v_{0246}^0v_{0246}^2v_{1357}^0v_{1357}^3)x^2+ \\ (v_{0246}^1v_{0246}^3v_{1357}^2)x + (\underline{v_{0246}^0v_{0246}^2}v_{0246}^3v_{1357}^1) \end{pmatrix}.$$

A.4. Matrix-vector product of N_8

$$A_0 = \begin{pmatrix} (v_6^0v_7^2)x^3 + (v_6^3v_7^1)x^2 + (v_6^2v_7^0v_7^3)x + (v_6^0v_6^1v_7^3) \\ (\underline{v_6^2v_7^0})x^3 + (\underline{v_6^1v_7^3})x^2 + (\underline{v_6^0v_7^2}v_6^3)x + (\underline{v_6^3v_7^1}v_7^0) \end{pmatrix},$$

$$A_1 = \begin{pmatrix} (v_4^0v_5^2)x^3 + (v_4^3v_5^1)x^2 + (v_4^2v_5^0v_5^3)x + (v_4^0v_4^1v_5^3) \\ (\underline{v_4^2v_5^0})x^3 + (\underline{v_4^1v_5^3})x^2 + (\underline{v_4^0v_5^2}v_4^3)x + (\underline{v_4^3v_5^1}v_5^0) \end{pmatrix},$$

$$A_2 = \begin{pmatrix} (v_{46}^1v_{46}^2v_{57}^3)x^3 + (v_{46}^0v_{46}^1v_{46}^3v_{57}^2)x^2+ \\ (v_{46}^0v_{46}^2v_{57}^1)x + (v_{46}^2v_{46}^3v_{57}^0) \\ (\underline{v_{46}^3v_{57}^2}v_{57}^1)x^3 + (\underline{v_{46}^2v_{57}^3}v_{57}^0v_{57}^1)x^2+ \\ (\underline{v_{46}^1v_{57}^2}v_{57}^0)x + (\underline{v_{46}^0v_{57}^2}v_{57}^3) \end{pmatrix},$$

$$B_0 = \begin{pmatrix} (v_2^0v_3^2)x^3 + (v_2^3v_3^1)x^2 + (v_2^2v_3^0v_3^3)x + (v_2^0v_2^1v_3^3) \\ (\underline{v_2^2v_3^0})x^3 + (\underline{v_2^1v_3^3})x^2 + (\underline{v_2^0v_3^2}v_2^3)x + (\underline{v_2^3v_3^1}v_3^0) \end{pmatrix},$$

$$B_1 = \begin{pmatrix} (v_0^0v_1^2)x^3 + (v_0^3v_1^1)x^2 + (v_0^2v_1^0v_1^3)x + (v_0^0v_0^1v_1^3) \\ (\underline{v_0^2v_1^0})x^3 + (\underline{v_0^1v_1^3})x^2 + (\underline{v_0^0v_1^2}v_0^3)x + (\underline{v_0^3v_1^1}v_1^0) \end{pmatrix},$$

$$B_2 = \begin{pmatrix} (v_{02}^1v_{02}^2v_{13}^3)x^3 + (v_{02}^0v_{02}^1v_{02}^3v_{13}^2)x^2+ \\ (v_{02}^0v_{02}^2v_{13}^1)x + (v_{02}^2v_{02}^3v_{13}^0) \\ (v_{02}^3v_{13}^1v_{13}^2)x^3 + (\underline{v_{02}^2v_{13}^1}v_{13}^0v_{13}^3)x^2+ \\ (\underline{v_{02}^1v_{13}^2}v_{13}^0)x + (v_{02}^0v_{13}^2v_{13}^3) \end{pmatrix},$$

$$C_0 = \begin{pmatrix} (v_{26}^1v_{37}^0v_{37}^1v_{37}^3)x^3 + (v_{26}^0v_{26}^3v_{37}^0v_{37}^2)x^2+ \\ (v_{26}^2v_{26}^3v_{37}^1v_{37}^3)x + (v_{26}^2v_{37}^1v_{37}^2) \\ (v_{26}^0v_{26}^3\underline{v_{26}^1v_{37}^1})x^3 + (v_{26}^2\underline{v_{26}^0v_{37}^0}v_{37}^3)x^2+ \\ (v_{26}^1\underline{v_{26}^3v_{37}^3}v_{37}^2)x + (v_{26}^1\underline{v_{26}^2v_{37}^2}) \end{pmatrix},$$

$$C_1 = \begin{pmatrix} (v_{04}^1v_{15}^0v_{15}^1v_{15}^3)x^3 + (v_{04}^0v_{04}^3v_{15}^0v_{15}^2)x^2+ \\ (v_{04}^2v_{04}^3v_{15}^1v_{15}^3)x + (v_{04}^2v_{15}^1v_{15}^2) \\ (v_{04}^0v_{04}^3\underline{v_{04}^1v_{15}^1})x^3 + (v_{04}^2\underline{v_{04}^0v_{15}^0}v_{15}^3)x^2+ \\ (v_{04}^1\underline{v_{04}^3v_{15}^3}v_{15}^2)x + (v_{04}^1\underline{v_{04}^2v_{15}^2}) \end{pmatrix},$$

$$C_2 = \begin{pmatrix} (v_{0246}^3 v_{1357}^2)x^3 + (v_{0246}^2 v_{1357}^1)x^2 + \\ \quad (v_{0246}^1 v_{1357}^0 v_{1357}^3)x + (v_{0246}^0 v_{1357}^3) \\ (v_{0246}^2 v_{1357}^3)x^3 + (v_{0246}^1 v_{1357}^2)x^2 + \\ \quad (v_{0246}^0 v_{0246}^3 v_{1357}^1)x + (v_{0246}^3 v_{1357}^0) \end{pmatrix}.$$

References

[1] Steve Babbage and Matthew Dodd. The stream cipher MICKEY 2.0. *ECRYPT Stream Cipher*, 2006.

[2] PSLM Barreto and Vincent Rijmen. The Khazad legacy-level block cipher. *Primitive submitted to NESSIE*, 97, 2000.

[3] PSLM Barreto and Vincent Rijmen. The Anubis Block Cipher. *Submission to the NESSIE Project*, 2000.

[4] Gérard M Baudet, Franco P Preparata, and Jean E Vuillemin. Area? time optimal vlsi circuits for convolution. *IEEE transactions on computers*, (7):684–688, 1983.

[5] Christof Beierle, Thorsten Kranz, and Gregor Leander. Lightweight multiplication in GF(2^n) with applications to MDS matrices. *international cryptology conference*, 2016:625–653, 2016.

[6] Daniel J Bernstein. Multidigit multiplication for mathematicians. *Advances in Applied Mathematics*, pages 1–19, 2001.

[7] Joan Boyar, Philip Matthews, and René Peralta. On the shortest linear straight-line program for computing linear forms. In *International Symposium on Mathematical Foundations of Computer Science*, pages 168–179. Springer, 2008.

[8] Ting Cui and Chenhui Jin. Construction of involution cauchy-hadamard type MDS matrices. *Journal of Electronics Information & Technology*, 32:500–503, 02 2010.

[9] Ting Cui, Chenhui Jin, and Zhiyin Kong. On compact cauchy matrices for substitution-permutation networks. *IEEE Transactions on Computers*, 64(7):2098–2102, 2014.

[10] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher Square. In *International Workshop on Fast Software Encryption*, 1997.

[11] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[12] Watanabe Dai, Soichi Furuya, Hirotaka Yoshida, Kazuo Takaragi, and Bart Preneel. *A New Keystream Generator MUGI*. 2002.

[13] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In *International Conference on Information Security*, pages 171–186. Springer, 2006.

[14] Haining Fan and M Anwar Hasan. A new approach to subquadratic space complexity parallel multipliers for extended binary fields. *IEEE Transactions on Computers*, 56(2):224–233, 2007.

[15] Haining Fan and M Anwar Hasan. A survey of some recent bit-parallel GF (2n) multipliers. *Finite Fields and Their Applications*, 32:5–43, 2015.

[16] Martin Fürer and Kurt Mehlhorn. At 2-optimal Galois field multiplier for vlsi. In *Aegean Workshop on Computing*, pages 217–225. Springer, 1986.

[17] Praveen Gauravaram, Lars R Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S Thomsen. Grøstl-a SHA-3 candidate. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.

[18] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In *Annual Cryptology Conference*, pages 222–239. Springer, 2011.

[19] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J B Robshaw. The LED block cipher. *cryptographic hardware and embedded systems*, 2012:326–341, 2011.

[20] Kishan Chand Gupta and Indranil Ghosh Ray. On constructions of circulant MDS matrices for lightweight cryptography. In *International Conference on Information Security Practice and Experience*, pages 564–576. Springer, 2014.

[21] Alper Halbutogullari and Çetin K Koç. Mastrovito multiplier for general irreducible polynomials. *IEEE Transactions on Computers*, 49(5):503–518, 2000.

[22] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *IJWMC*, 2(1):86–93, 2007.

[23] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Joltik v1. 3. *CAESAR Round*, 2, 2015.

[24] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing implementations of lightweight building blocks. *IACR Transactions on Symmetric Cryptology*, pages 130–168, 2017.

[25] Pascal Junod and Serge Vaudenay. Fox: a new family of block ciphers. *Lecture Notes in Computer Science*, 3357:131–146, 2004.

[26] Anatolii Karatsuba. Multiplication of multidigit numbers on automata. In *Soviet physics doklady*, volume 7, pages 595–596, 1963.

[27] Khoongming Khoo, Thomas Peyrin, Axel Y Poschmann, and Huihui Yap. Foam: searching for hardware-optimal spn structures and components with a fair comparison. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 433–450. Springer, 2014.

[28] Lukas Kölsch. Xor-counts and lightweight multiplication with fixed elements in binary finite fields. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 285–312. Springer, 2019.

[29] Paulo S L M and Vincent Rijmen. The whirlpool hashing function. *the web conference*, 2003.

[30] Christophe Negre. Quadrinomial modular arithmetic using modified polynomial basis. In *International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II*, volume 1, pages 550–555. IEEE, 2005.

[31] Arash Reyhani-Masoleh and M Anwarul Hasan. A new construction of massey-omura parallel multiplier over GF (2^m). *IEEE Transactions on Computers*, 51(5):511–520, 2002.

[32] Vincent Rijmen, Joan Daemen, Bart Preneel, Antoon Bosselaers, and Erik De Win. The cipher shark. In *International Workshop on Fast Software Encryption*, 1996.

[33] Sumanta Sarkar and Habeeb Syed. Lightweight diffusion layer: Importance of toeplitz matrices. *IACR Transactions on Symmetric Cryptology*, pages 95–113, 2016.

[34] Erkay Savaš, Alexandre F Tenca, and Cetin K Koç. A scalable and unified multiplier architecture for finite fields GF (p) and GF (2^m). In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 277–292. Springer, 2000.

[35] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. The Twofish encryption algorithm: a 128-bit block cipher. 1999.

[36] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA. In *International Conference on Fast Software Encryption*, 2007.

[37] Siang Meng Sim, Khoongming Khoo, Frederique E Oggier, and Thomas Peyrin. Lightweight MDS involution matrices. *fast software encryption*, 2015:471–493, 2015.