Lightweight MDS Involution Matrices

Siang Meng Sim[1], Khoongming Khoo[2], Frédérique Oggier[1], and Thomas Peyrin[1]

[1] Nanyang Technological University, Singapore
[2] DSO National Laboratories, Singapore

ssim011@e.ntu.edu.sg, kkhoongm@dso.org.sg, frederique@ntu.edu.sg, thomas.peyrin@ntu.edu.sg

Abstract. In this article, we provide new methods to look for lightweight MDS matrices, and in particular involutory ones. By proving many new properties and equivalence classes for various MDS matrices constructions such as circulant, Hadamard, Cauchy and Hadamard-Cauchy, we exhibit new search algorithms that greatly reduce the search space and make lightweight MDS matrices of rather high dimension possible to find. We also explain why the choice of the irreducible polynomial might have a significant impact on the lightweightness, and in contrary to the classical belief, we show that the Hamming weight has no direct impact. Even though we focused our studies on involutory MDS matrices, we also obtained results for non-involutory MDS matrices. Overall, using Hadamard or Hadamard-Cauchy constructions, we provide the (involutory or non-involutory) MDS matrices with the least possible XOR gates for the classical dimensions 4×4, 8×8, 16×16 and 32×32 in $GF(2^4)$ and $GF(2^8)$. Compared to the best known matrices, some of our new candidates save up to 50% on the amount of XOR gates required for an hardware implementation. Finally, our work indicates that involutory MDS matrices are really interesting building blocks for designers as they can be implemented with almost the same number of XOR gates as non-involutory MDS matrices, the latter being usually non-lightweight when the inverse matrix is required.

Key words: lightweight cryptography, Hadamard matrix, Cauchy matrix, involution, MDS.

1 Introduction

Most symmetric key primitives, like block ciphers, stream ciphers or hash functions, are usually based on various components that provide confusion and diffusion. Both concepts are very important for the overall security and efficiency of the cryptographic scheme and extensive studies have been conducted to find the best possible building blocks. The goal of diffusion is basically to spread the internal dependencies as much as possible. Several designs use a weak yet fast diffusion layer based on simple XOR, addition and shifting operation, but another trend is to rely on strong linear diffusion matrices, like Maximal Distance Separable (MDS) matrices. A typical example is the `AES` cipher [17], which uses a 4×4 matrix in $GF(2^8)$ to provide diffusion among a vector of 4 bytes. These mathematical objects ensure the designers a perfect diffusion (the underlying linear code meets the Singleton bound), but can be quite heavy to implement. Software performances are usually not so much impacted as memory is not really constrained and table-based implementations directly incorporate the field multiplications in the stored values. However, hardware implementations will usually suffer from an important area requirement due to the Galois field multiplications. The impact will also be visible on the efficiency of software bitslice implementations which basically mimic the hardware computations flow.

Good hardware efficiency has became a major design trend in cryptography, due to the increasing importance of ubiquitous computing. Many lightweight algorithms have recently been proposed, notably block ciphers [12, 14, 19, 9] and hash functions [4, 18, 11]. The choice of MDS matrices played an important role in the reduction of the area required to provide a certain amount of security. Along with `PHOTON` hash function [18] was proposed a new type of MDS matrix that can be computed in a serial or recursive manner. This construction greatly reduces the temporary memory (and thus the hardware area) usually required for the computation of the matrix. Such matrices were later used in `LED` [19] block cipher, or `PRIMATEs` [1] authenticated encryption scheme, and were further studied and generalized in subsequent articles [28, 32, 3, 2, 10]. Even though these serial matrices provide a good way to save area, this naturally comes at the expense of an increased number of cycles to apply the matrix. In general, they are not well suited for round-based or low-latency implementations.

Another interesting property for an MDS matrix to save area is to be involutory. Indeed, in most use cases, encryption and decryption implementations are required and the inverse of the MDS matrix will have to be implemented as well (except for constructions like Feistel networks, where the inverse of the internal function is not needed for decryption). For example, the MDS matrix of `AES` is quite lightweight for encryption, but not really for decryption[3]. More generally, it is a valuable advantage that one can use *exactly* the same diffusion matrix for encryption and decryption. Some ciphers like `ANUBIS` [5], `KHAZAD` [6], `ICEBERG` [31] or `PRINCE` [13] even pushed the involution idea a bit further by defining a round function that is almost entirely composed of involution operations, and where the non-involution property of the cipher is mainly provided by the key schedule.

There are several ways to build a MDS matrix [33, 23, 25, 29, 20, 15], a common method being to use a circulant construction, like for the `AES` block cipher [17] or the `WHIRLPOOL` hash function [8]. The obvious benefit of a circulant matrix for hardware implementations is that all of its rows are similar (up to a right shift), and one can trivially reuse the multiplication circuit to save implementation costs. However, it has been proven in [22] that circulant matrices of order 4 cannot be simultaneously MDS and involutory. And very recently

[3] The serial matrix construction proposed in [18, 19] allows an efficient inverse computation if the first coefficient is equal to 1. However, we recall that serial matrices are not well suited for round-based or low-latency implementations.

Gupta *et al.* [21] proved that circulant MDS involutory matrices do not exist. Finding lightweight matrices that are both MDS and involutory is not an easy task and this topic has attracted attention recently. In [29], the authors consider Vandermonde or Hadamard matrices, while in [33, 20, 15] Cauchy matrices were used. Even if these constructions allow to build involutory MDS matrices for big matrix dimensions, it is difficult to find the most lightweight candidates as the search space can become really big.

Our contributions. In this article, we propose a new method to search for lightweight MDS matrices, with an important focus on involutory ones. After having recalled the formula to compute the XOR count, i.e. the amount of XORs required to evaluate one row of the matrix, we show in Section 2 that the choice of the irreducible polynomial is important and can have a significant impact on the efficiency, as remarked in [24]. In particular, we show that the best choice is not necessarily a low Hamming weight polynomial as widely believed, but instead one that has a high standard deviation regarding its XOR count. Then, in Section 3, we recall some constructions to obtain (involutory) MDS matrices: circulant, Hadamard, Cauchy and Cauchy-Hadamard. In particular, we prove new properties for some of these constructions, which will later help us to find good matrices. In Section 4 we prove the existence of equivalent classes for Hadamard matrices and involutory Hadamard-Cauchy matrices and we use these considerations to conceive improved search algorithms of lightweight (involutory) MDS matrices. In Section 5, we quickly describe these new algorithms, providing all the details for lightweight involutory MDS matrices in Appendix B and for lightweight non-involutory MDS matrices in Appendix C. Our methods can also be relaxed and applied to the search of lightweight non-involutory MDS matrices. These algorithms are significant because they are feasible exhaustive search while the search space of the algorithms described in [20, 15] is too big to be exhausted[4]. Our algorithms guarantee that the matrices found are the lightest according to our metric.

Overall, using Hadamard or Hadamard-Cauchy constructions, we provide the smallest known (involutory or non-involutory) MDS matrices for the classical dimensions 4×4, 8×8, 16×16 and 32×32 in $\mathrm{GF}(2^4)$ and $\mathrm{GF}(2^8)$. All our results are summarized and commented in Section 6. Surprisingly, it seems that involutory MDS matrices are not much more expensive than non-involutory MDS ones, the former providing the great advantage of a free inverse implementation as well. We recall that in this article we are not considering serial matrices, as their evaluation either requires many clock cycles (for serial implementations) or an important area (for round-based implementations).

Due to space constraints, all proofs are given in the Appendix D.

[4] The huge search space issue can be reduced if one could search intelligently only among lightweight matrix candidates. However, this is not possible with algorithms from [20, 15] since the matrix coefficients are known only at the end of the matrix generation, and thus one cannot limit the search to lightweight candidates only.

Notations and preliminaries. We denote by $\mathrm{GF}(2^r)$ the finite field with 2^r elements, $r \geq 1$. This field is isomorphic to polynomials in $\mathrm{GF}(2)[X]$ modulo an irreducible polynomial $p(X)$ of degree r, meaning that every field element can be seen as a polynomial $\alpha(X)$ with coefficients in $\mathrm{GF}(2)$ and of degree $r-1$: $\alpha(X) = \sum_{i=0}^{r-1} b_i X^i$, $b_i \in \mathrm{GF}(2)$, $0 \leq i \leq r-1$. The polynomial $\alpha(X)$ can also naturally be viewed as an r-bit string $(b_{r-1}, b_{r-2}, ..., b_0)$. In the rest of the article, an element α in $\mathrm{GF}(2^r)$ will be seen either as the polynomial $\alpha(X)$, or the r-bit string represented in a hexadecimal representation, which will be prefixed with 0x. For example, in $\mathrm{GF}(2^8)$, the 8-bit string 00101010 corresponds to the polynomial $X^5 + X^3 + X$, written 0x2a in hexadecimal.

The addition operation on $\mathrm{GF}(2^r)$ is simply defined as a bitwise XOR on the coefficients of the polynomial representation of the elements, and does not depend on the choice of the irreducible polynomial $p(X)$. However, for multiplication, one needs to specify the irreducible polynomial $p(X)$ of degree r. We denote this field as $\mathrm{GF}(2^r)/p(X)$, where $p(X)$ can be given in hexadecimal representation[5]. The multiplication of two elements is then the modulo $p(X)$ reduction of the product of the polynomial representations of the two elements.

Finally, we denote by $M[i, j]$ the (i, j) entry of the matrix M, we start the counting from 0, that is $M[0, 0]$ is the entry corresponding to the first row and first column.

2 Analyzing XOR count according to different finite fields

In this section, we explain the XOR count that we will use as a measure to evaluate the lightweightness of a given matrix. Then, we will analyze the XOR count distribution depending on the finite field and irreducible polynomial considered. Although it is known that finite fields of the same size are isomorphic to each other and it is believed that the security of MDS matrices is not impacted by this choice, looking at the XOR count is a new aspect of finite fields that remains unexplored in cryptography.

2.1 The XOR count

It is to note that the XOR count is an easy-to-manipulate and simplified metric, but MDS coefficients have often been chosen to lower XOR count, e.g. by having low Hamming weight. As shown in [24], low XOR count is strongly correlated minimization of hardware area.

Later in this article, we will study the hardware efficiency of some diffusion matrices and we will search among huge sets of candidates. One of the goals

[5] This should not be confused with the explicit construction of finite fields, which is commonly denoted as $\mathrm{GF}(2^r)[X]/(P)$, where (P) is an ideal generated by irreducible polynomial P.

will therefore be to minimize the area required to implement these lightweight matrices, and since they will be implemented with XOR gates (the diffusion layer is linear), we need a way to easily evaluate how many XORs will be required to implement them. We explain our method in this subsection.

In general, it is known that low Hamming weight generally requires lesser hardware resource in implementations, and this is the usual choice criteria for picking a matrix. For example, the coefficients of the `AES` MDS matrix are 1, 2 and 3, in a hope that this will ensure a lightweight implementation. However, it was shown in [24] that while this heuristic is true in general, it is not always the case. Due to some reduction effects, and depending on the irreducible polynomial defining the computation field, some coefficients with not-so-low Hamming weight might be implemented with very few XORs.

Definition 1 *The XOR count of an element α in the field $\mathrm{GF}(2^r)/p(X)$ is the number of XORs required to implement the multiplication of α with an arbitrary β over $\mathrm{GF}(2^r)/p(X)$.*

For example, let us explain how we compute the XOR count of $\alpha = 3$ over $\mathrm{GF}(2^4)/\mathtt{0x13}$ and $\mathrm{GF}(2^4)/\mathtt{0x19}$. Let (b_3, b_2, b_1, b_0) be the binary representation of an arbitrary element β in the field. For $\mathrm{GF}(2^4)/\mathtt{0x13}$, we have:

$$(0,0,1,1)\cdot(b_3,b_2,b_1,b_0) = (b_2,b_1,b_0\oplus b_3,b_3)\oplus(b_3,b_2,b_1,b_0) = (b_2\oplus b_3, b_1\oplus b_2, b_0\oplus b_1\oplus b_3, b_0\oplus b_3),$$

which corresponds to 5 XORs[6]. For $\mathrm{GF}(2^4)/\mathtt{0x19}$, we have:

$$(0,0,1,1)\cdot(b_3,b_2,b_1,b_0) = (b_2\oplus b_3,b_1,b_0,b_3)\oplus(b_3,b_2,b_1,b_0) = (b_2, b_1\oplus b_2, b_0\oplus b_1, b_0\oplus b_3),$$

which corresponds to 3 XORs. One can observe that XOR count is different depending on the finite field defined by the irreducible polynomial.

In order to calculate the number of XORs required to implement an entire row of a matrix, we can use the following formula given in [24]:

$$\text{XOR count for one row of } M = (\gamma_1, \gamma_2, ..., \gamma_k) + (n-1)\cdot r, \qquad (1)$$

where γ_i is the XOR count of the i-th entry in the row of M, n being the number of nonzero elements in the row and r the dimension of the finite field.

For example, the first row of the `AES` diffusion matrix being $(1, 1, 2, 3)$ over the field $\mathrm{GF}(2^8)/\mathtt{0x11b}$, the XOR count for the first row is $(0+0+3+11)+3\times 8 = 38$ XORs (the matrix being circulant, all rows are equivalent in terms of XOR count).

[6] We acknowledge that one can perform the multiplication with 4 XORs as $b_0 \oplus b_3$ appears twice. But that would require additional cycle and extra memory cost which completely outweighed the small saving on the XOR count.

2.2 XOR count for different finite fields

We programmed a tool that computes the XOR count for every nonzero element over $\mathrm{GF}(2^r)$ for $r = 2, \ldots, 8$ and for all possible irreducible polynomials (all the tables will be given in the full version of this article, we provide an extract in Appendix F). By analyzing the outputs of this tool, we could make two observations that are important to understand how the choice of the irreducible polynomial affects the XOR count. Before presenting our observations, we state some terminologies and properties related to reciprocal polynomials in finite fields.

Definition 2 *A reciprocal polynomial* $\frac{1}{p}(X)$ *of a polynomial* $p(X)$ *over* $\mathrm{GF}(2^r)$*, is a polynomial expressed as* $\frac{1}{p}(X) = X^r \cdot p(X^{-1})$*. A reciprocal finite field,* $\mathbf{K} = \mathrm{GF}(2^r)/\frac{1}{p}(X)$*, is a finite field defined by the reciprocal polynomial which defines* $\mathbf{F} = \mathrm{GF}(2^r)/p(X)$*.*

In other words, a reciprocal polynomial is a polynomial with the order of the coefficients reversed. For example, the reciprocal polynomial of $p(X) =$ `0x11b` in $\mathrm{GF}(2^8)$ is $\frac{1}{p}(X) = \mathtt{0x}\frac{1}{\mathtt{11b}} =$ `0x1b1`. It is also to be noted that the reciprocal polynomial of an irreducible polynomial is also irreducible.

The total XOR count. Our first new observation is that even if for an individual element of the field the choice of the irreducible polynomial has an impact on the XOR count, the total sum of the XOR count over all elements in the field is independent of this choice. We state this in the following theorem, the proof being provided in Appendix D.1.

Theorem 1 *The total XOR count for a field* $\mathrm{GF}(2^r)$ *is* $r \sum_{i=2}^{r} 2^{i-2}(i-1)$*, where* $r \geq 2$*.*

From Theorem 1, it seems that there is no clear implication that one irreducible polynomial is strictly better than another, as the mean XOR count is the same for any irreducible polynomial. However, the irreducible polynomials have different distribution of the XOR count among the field elements, that is quantified by the standard deviation. A high standard deviation implies that the distribution of XOR count is very different from the mean, thus there will be more elements with relatively lower/higher XOR count. In general, the order of the finite field is much larger than the order of the MDS matrix and since only a few elements of the field will be used in the MDS matrices, there is a better chance of finding an MDS matrix with lower XOR count.

Hence, our recommendation is to choose the irreducible polynomial with the highest standard deviation regarding the XOR count distribution. From previous example, in $\mathrm{GF}(2^4)$ (XOR count mean equals 4.25 for this field dimension), the irreducible polynomials `0x13` and `0x19` lead to a standard deviation of 2.68, while

0x1f leads to a standard deviation of 1.7075. Therefore, the two first polynomials seem to be a better choice. This observation will allow us to choose the best irreducible polynomial to start with during the searches. We refer to Appendix F for all the standard deviations according to the irreducible polynomial.

We note that the folklore belief was that in order to get lightweight implementations, one should use a low Hamming weight irreducible polynomial. The underlying idea is that with such a polynomial less XORs might be needed when the modular reduction has to be applied during a field multiplication. However, we have shown that this is not necessarily true. Yet, by looking at the data from Appendix F, we remark that the low Hamming weight irreducible polynomials usually have a high standard deviation, which actually validates the folklore belief. We conjecture that this heuristic will be less and less exact when we go to higher and higher order fields.

Matching XOR count. Our second new observation is that the XOR count distribution implied by a polynomial will be the same compared to the distribution of its reciprocal counterpart. We state this observation in the following theorem, the proof being provided in Appendix D.2.

Theorem 2 *There exists an isomorphic mapping from a primitive* $\alpha \in \mathrm{GF}(2^r)/p(X)$ *to another primitive* $\beta \in \mathrm{GF}(2^r)/\frac{1}{p}(X)$ *where the XOR count of* α^i *and* β^i *is equal for each* $i = \{1, 2, ..., 2^r - 1\}$.

In Appendix E, we listed all the primitive mapping from a finite field to its reciprocal finite field for all fields $\mathrm{GF}(2^r)$ with $r = 2, \ldots, 8$ and for all possible irreducible polynomials. We give an example to illustrate our theorem. For $\mathrm{GF}(2^4)$, there are three irreducible polynomials: 0x13, 0x19 and 0x1f and the XOR count for the elements are shown in Appendix F. From the binary representation we see that $0\mathrm{x}\frac{1}{13} = 0\mathrm{x}19$. Consider an isomorphic mapping $\phi : \mathrm{GF}(2^4)/0\mathrm{x}13 \rightarrow \mathrm{GF}(2^4)/0\mathrm{x}19$ defined as $\phi(2) = 12$, where 2 and 12 are the primitives for the respective finite fields. Table 2 of Appendix E shows that the order of the XOR count is the same.

We remark that for a self-reciprocal irreducible polynomial, for instance 0x1f in $\mathrm{GF}(2^4)$, there also exists an automorphism mapping from a primitive to another primitive with the same order of XOR count (see Appendix E).

Theorem 2 is useful for understanding that we do not need to consider $\mathrm{GF}(2^r)/\frac{1}{p}(X)$ when we are searching for lightweight matrices. As there exists an isomorphic mapping preserving the order of the XOR count, any MDS matrix over $\mathrm{GF}(2^r)/\frac{1}{p}(X)$ can be mapped to an MDS matrix over $\mathrm{GF}(2^r)/p(X)$ while preserving the XOR count. Therefore, it is redundant to search for lightweight MDS matrices over $\mathrm{GF}(2^r)/\frac{1}{p}(X)$ as the lightest MDS matrix can also be found in $\mathrm{GF}(2^r)/p(X)$. This will render our algorithms much more efficient: when using exhaustive search for low XOR count MDS over finite field defined by various irreducible polynomial, one can reduce the search space by almost a factor 2 as the reciprocal polynomials are redundant.

3 Types of MDS matrices and properties

In this section, we first recall a few properties of MDS matrices and we then explain various constructions of (involutory) MDS matrices that were used to generate lightweight candidates. Namely, we will study 4 types of diffusion matrices: circulant, Hadamard, Cauchy, and Hadamard-Cauchy. We recall that we do not consider serially computable matrices in this article, like the ones described in [18, 19, 28, 32, 3, 2], since they are not adapted to round-based implementations. As MDS matrices are widely studied and their properties are commonly known, their definition and properties are given in the Appendix A.

3.1 Circulant matrices

A common way to build an MDS matrix is to start from a circulant matrix, reason being that the probability of finding an MDS matrix would then be higher than a normal square matrix [16].

Definition 3 *A $k \times k$ matrix C is circulant when each row vector is rotated to the right relative to the preceding row vector by one element. The matrix is then fully defined by its first row.*

An interesting property of circulant matrices is that since each row differs from the previous row by a right shift, a user can just implement one row of the matrix multiplication in hardware and reuse the multiplication circuit for subsequent rows by just shifting the input. However in this paper, we will show in Section B.1 and C.1 that these matrices are not the best choice.

3.2 Hadamard matrices

Definition 4 ([20]) *A finite field Hadamard (or simply called Hadamard) matrix H is a $k \times k$ matrix, with $k = 2^s$, that can be represented by two other submatrices H_1 and H_2 which are also Hadamard matrices:*

$$H = \begin{pmatrix} H_1 & H_2 \\ H_2 & H_1 \end{pmatrix}.$$

Similarly to [20], in order to represent a Hadamard matrix we use notation $had(h_0, h_1, ..., h_{k-1})$ (with $h_i = H[0, i]$ standing for the entries of the first row of the matrix) where $H[i, j] = h_{i \oplus j}$ and $k = 2^s$. It is clear that a Hadamard matrix is bisymmetric. Indeed, if we define the left and right diagonal reflection transformations as $H_L = T_L(H)$ and $H_R = T_R(H)$ respectively, we have that $H_L[i, j] = H[j, i] = H[i, j]$ and $H_R[i, j] = H[k - 1 - i, k - 1 - j] = H[i, j]$ (the binary representation of $k - 1 = 2^s - 1$ is all 1, hence $k - 1 - i = (k - 1) \oplus i$).

Moreover, by doing the multiplication directly, it is known that if $H = had(h_0, h_1, ..., h_{k-1})$ is a Hadamard matrix, then $H \times H = c^2 \cdot I$, with $c^2 =$

$h_0^2 + h_1^2 + h_2^2 + ... + h_{k-1}^2$. In other words, the product of a Hadamard matrix with itself is a multiple of an identity matrix, where the multiple c^2 is the sum of the square of the elements from the first row.

A direct and crucial corollary to this fact is that a Hadamard matrix over $\mathrm{GF}(2^r)$ is involution if the sum of the elements of the first row is equal to 1. Now, it is important to note that if one deals with a Hadamard matrix for which the sum of the first row over $\mathrm{GF}(2^r)$ is nonzero, we can very simply make it involutory by dividing it with the sum of its first row.

We will use these considerations in Section B.2 to generate low dimension diffusion matrices (order 4 and 8) with an innovative exhaustive search over all the possible Hadamard matrices. We note that, similarity to a circulant matrix, an Hadamard matrix will have the interesting property that each row is a permutation of the first row, therefore allowing to reuse the multiplication circuit to save implementation costs.

3.3 Cauchy matrices

Definition 5 *A square Cauchy matrix, C, is a $k \times k$ matrix constructed with two disjoint sets of elements from* $\mathrm{GF}(2^r)$, $\{\alpha_0, \alpha_1, ..., \alpha_{k-1}\}$ *and* $\{\beta_0, \beta_1, ..., \beta_{k-1}\}$ *such that* $C[i,j] = \frac{1}{\alpha_i + \beta_j}$.

It is known that the determinant of a square Cauchy matrix, C, is given as

$$\det(C) = \frac{\prod_{0 \le i < j \le k-1} (\alpha_j - \alpha_i)(\beta_j - \beta_i)}{\prod_{0 \le i < j \le k-1} (\alpha_i + \alpha_j)}.$$

Since $\alpha_i \neq \alpha_j$, $\beta_i \neq \beta_j$ for all $i, j \in \{0, 1, ..., k-1\}$, a Cauchy matrix is nonsingular. Note that for a Cauchy matrix over $\mathrm{GF}(2^r)$, the subtraction is equivalent to addition as the finite field has characteristic 2. As the sets are disjoint, we have $\alpha_i \neq \beta_j$, thus all entries are well-defined and nonzero. In addition, any submatrix of a Cauchy matrix is also a Cauchy matrix as it is equivalent to constructing a smaller Cauchy matrix with subsets of the two disjoint sets. Therefore, by the first statement of Proposition 3, a Cauchy matrix is an MDS matrix.

3.4 Hadamard-Cauchy matrices

The innovative exhaustive search over Hadamard matrices from Section B.2 is sufficient to generate low dimension diffusion matrices (order 4 and 8). However, the computation for verifying the MDS property and the exhaustive search space grows exponentially. It eventually becomes impractical to search for higher dimension Hadamard matrices (order 16 or more). Therefore, we use the Hadamard-Cauchy matrix construction, proposed in [20] as an evolution of the involutory MDS Vandermonde matrices [28], that guarantees the matrix to be an involutory MDS matrix.

In [20], the authors proposed a $2^s \times 2^s$ matrix construction that combines both the characteristics of Hadamard and Cauchy matrices. Because it is a Cauchy matrix, a Hadamard-Cauchy matrix is an MDS matrix. And because it is a Hadamard matrix, it will be involutory when c^2 is equal to 1. Therefore, we can construct a Hadamard-Cauchy matrix and check if the sum of first row is equal to 1 and, if so, we have an MDS and involutory matrix. A detailed discussion on Hadamard-Cauchy matrices is given in Section B.3.

4 Equivalence classes of Hadamard-based matrices

Our methodology for finding lightweight MDS matrices is to perform an innovative exhaustive search and by eventually picking the matrix with the lowest XOR count. Naturally, the main problem to tackle is the huge search space. By exploiting the properties of Hadamard matrices, we found ways to group them in equivalent classes and significantly reduce the search space. In this section, we introduce the equivalence classes of Hadamard matrices and the equivalence classes of involutory Hadamard-Cauchy matrices. It is important to note that these two equivalence classes are rather different as they are defined by very different relations. We will later use these classes in Sections B.2, B.3, C.2 and C.3.

4.1 Equivalence classes of Hadamard matrices

It is known that a Hadamard matrix can be defined by its first row, and different permutation of the first row results in a different Hadamard matrix with possibly different branch number. In order to find a lightweight MDS involution matrix, it is necessary to have a set of k elements with relatively low XOR count that sum to 1 (to guarantee involution). Moreover, we need all coefficients in the first row to be different. Indeed, if the first row of an Hadamard matrix has 2 or more of the same element, say $H[0, i] = H[0, j]$, where $i, j \in \{0, 1, ..., k-1\}$, then in another row we have $H[i \oplus j, i] = H[i \oplus j, j]$. These 4 entries are the same and by Corollary 3, H is not MDS.

By permuting the entries we hope to find an MDS involution matrix. However, given k distinct nonzero elements, there are $k!$ ways to permute the first row of the Hadamard matrix, which can quickly become intractable. Therefore, we introduce a relation that relates certain permutations that lead to the same branch number.

Definition 6 *Let H and $H^{(\sigma)}$ be two Hadamard matrices with the same set of entries up to some permutation σ. We say that they are related, $H \sim H^{(\sigma)}$, if every pair of input vectors, $(v, v^{(\sigma)})$ with the same permutation σ, to H and $H^{(\sigma)}$ respectively, have the same set of elements in the output vectors.*

For example, let us consider the following three Hadamard matrices

$$H = \begin{pmatrix} w & x & y & z \\ x & w & z & y \\ y & z & w & x \\ z & y & x & w \end{pmatrix}, \qquad H^{(\sigma_1)} = \begin{pmatrix} y & z & w & x \\ z & y & x & w \\ w & x & y & z \\ x & w & z & y \end{pmatrix}, \qquad H^{(\sigma_2)} = \begin{pmatrix} w & x & z & y \\ x & w & y & z \\ z & y & w & x \\ y & z & x & w \end{pmatrix},$$

One can see that $H^{(\sigma_1)}$ is defined by the third row of H, i.e. the rows are shifted by two positions and $\sigma_1 = \{2, 3, 0, 1\}$. Let us consider an arbitrary input vector for H, say $v = (a, b, c, d)$. Then, if we apply the permutation to v, we obtain $v^{(\sigma_1)} = (c, d, a, b)$. We can observe that:

$$v \cdot H = (aw + bx + cy + dz, ax + bw + cz + dy, ay + bz + cw + dx, az + by + cx + dw),$$

$$v^{(\sigma_1)} \cdot H^{(\sigma_1)} = (cy + dz + aw + bx, cz + dy + ax + bw, cw + dx + ay + bz, cx + dw + az + by),$$

It is now easy to see that $v \cdot H = v^{(\sigma_1)} \cdot H^{(\sigma_1)}$. Hence, we say that $H \sim H^{(\sigma_1)}$. Similarily, with $\sigma_2 = \{0, 1, 3, 2\}$, we have $v^{(\sigma_2)} = (a, b, d, c)$ and:

$$v \cdot H = (aw + bx + cy + dz, ax + bw + cz + dy, ay + bz + cw + dx, az + by + cx + dw),$$

$$v^{(\sigma_2)} \cdot H^{(\sigma_2)} = (aw + bx + dz + cy, ax + bw + dy + cz, az + by + dw + cx, ay + bz + dx + cw),$$

and since $v \cdot H$ and $v^{(\sigma_2)} \cdot H^{(\sigma_2)}$ are the same up to the permutation σ_2, we can say that $H \sim H^{(\sigma_2)}$.

Definition 7 *An equivalence class of Hadamard matrices is a set of Hadamard matrices satisfying the equivalence relation* $\sim$*.*

Proposition 1 *Hadamard matrices in the same equivalence class have the same branch number.*

When searching for an MDS matrix, we can make use of this property to greatly reduce the search space: if one Hadamard matrix in an equivalence class is not MDS, then all other Hadamard matrices in the same equivalence class will not be MDS either. Therefore, it all boils down to analyzing how many and which permutation of the Hadamard matrices belongs to the same equivalence classes. Using the two previous examples σ_1 and σ_2 as building blocks, we generalize them and present two lemmas.

Lemma 1 *Given a Hadamard matrix H, any Hadamard matrix $H^{(\alpha)}$ defined by the $(\alpha + 1)$-th row of H, with $\alpha = 0, 1, 2, ..., k - 1$, is equivalent to H.*

Next, let us consider the other type of permutation. We can see in the example with σ_2 that up to the permutation applied to the Hadamard matrix, input and output vectors are the same. Let $H^{(\sigma)}$, $v^{(\sigma)}$ and $u^{(\sigma)}$ denote the permuted Hadamard matrix, the permuted input vector and its corresponding permuted output vector. We want the permutation to satisfy $u_{\sigma(j)} = u_j^{(\sigma)}$, where $j \in \{0, 1, ..., k - 1\}$. That is the permutation of the output vector of H is the same as the permuted output vector of $H^{(\sigma)}$. Using the definition of the Hadamard matrix, we can rewrite it as

$$\bigoplus_{i=0}^{k-1} v_i h_{i\oplus\sigma(j)} = \bigoplus_{i=0}^{k-1} v_i^{(\sigma)} H^{(\sigma)}[i,j].$$

Using the definition of the permutation and by the fact that it is one-to-one mapping, we can rearrange the XOR order of the terms on the left-hand side and we obtain

$$\bigoplus_{i=0}^{k-1} v_{\sigma(i)} h_{\sigma(i)\oplus\sigma(j)} = \bigoplus_{i=0}^{k-1} v_{\sigma(i)} h_{\sigma(i\oplus j)}.$$

Therefore, we need the permutation to be linear with respect to XOR: $\sigma(i\oplus j) = \sigma(i) \oplus \sigma(j)$. This proves our next lemma.

Lemma 2 *For any linear permutation σ (w.r.t. XOR), the two Hadamard matrices H and $H^{(\sigma)}$ are equivalent.*

We note that the permutations in Lemma 1 and 2 are disjoint, except for the identity permutation. This is because for the linear permutation σ, it always maps the identity to itself: $\sigma(0) = 0$. Thus, for any linear permutation, the first entry remains unchanged. On the other hand, when choosing another row of H as the first row, the first entry is always different.

With these two lemmas, we can now partition the family of Hadamard matrices into equivalence classes. For Lemma 1, we can easily see that the number of permutation is equal to the order of the Hadamard matrix. However, for Lemma 2 it is not so trivial. Therefore, we have the following lemma.

Lemma 3 *Given a set of 2^s nonzero elements, $S = \{\alpha_0, \alpha_1, ..., \alpha_{2^s-1}\}$, there are $\prod_{i=0}^{s-1}(2^s - 2^i)$ linear permutations w.r.t. XOR operation.*

Theorem 3 *Given a set of 2^s nonzero elements, $S = \{\alpha_0, \alpha_1, ..., \alpha_{2^s-1}\}$, there are $\frac{(2^s-1)!}{\prod_{i=0}^{s-1}(2^s-2^i)}$ equivalence classes of Hadamard matrices of order 2^s defined by the set of elements S.*

For convenience, we call the permutations in Lemma 1 and 2 the $\mathcal{H}$-permutations. The $\mathcal{H}$-permutations can be described as a sequence of the following types of permutations on the index of the entries:

1. choose $\alpha \in \{0, 1, ..., 2^s - 1\}$, define $\sigma(i) = i \oplus \alpha, \forall i = 0, 1, ..., 2^s - 1$, and
2. fix $\sigma(0) = 0$, in ascending order of the index i, choose the permutation if i is power of 2, otherwise it is defined by the linear permutation (w.r.t. XOR): $\sigma(i \oplus j) = \sigma(i) \oplus \sigma(j)$.

We remark that given a set of 4 nonzero elements, from Theorem 3 we see that there is only 1 equivalence class of Hadamard matrices. This implies that there is no need to permute the entries of the 4×4 Hadamard matrix in hope to find MDS matrix if one of the permutation is not MDS.

With the knowledge of equivalence classes of Hadamard matrices, what we need is an algorithm to pick one representative from each equivalence class and check if it is MDS. The idea is to exhaust all non-$\mathcal{H}$-permutations through selecting the entries in ascending index order. Since the entries in the first column of Hadamard matrix are distinct (otherwise the matrix is not MDS), it is sufficient for us to check the matrices with the first entry (index 0) being the smallest element. This is because for any other matrices with the first entry set as some other element, it is in the same equivalence class as some matrix $H^{(\alpha)}$ where the first entry of $(\alpha + 1)$-th row is the smallest element. For indexes that are powers of 2, select the smallest element from the remaining set. While for the other entries, one can pick any element from the remaining set.

For 8×8 Hadamard matrices for example, the first three entries, α_0, α_1 and α_2 are fixed to be the three smallest elements in ascending order. Next, by Lemma 2, α_3 should be defined by α_1 and α_2 in order to preserve the linear property, thus to "destroy" the linear property and obtain matrices from different equivalence classes, pick an element from the remaining set in ascending order as the fourth entry α_3. After which, α_4 is selected to be the smallest element among the remaining 4 elements and permute the remaining 3 elements to be α_5, α_6 and α_7 respectively. For each of these arrangement of entries, we check if it is MDS using the algorithm discussed in Section B.2. We terminate the algorithm prematurely once an MDS matrix is found, else we conclude that the given set of elements does not generate an MDS matrix.

It is clear that arranging the entries in this manner will not obtain two Hadamard matrices from the same equivalence class. But one may wonder if it actually does exhaust all the equivalence classes. The answer is yes: Theorem 3 shows that there is a total of 30 equivalence classes for 8×8 Hadamard matrices. On the other hand, from the algorithm described above, we have 5 choices for α_3 and we permute the remaining 3 elements for α_5, α_6 and α_7. Thus, there are 30 Hadamard matrices that we have to check.

4.2 Equivalence classes of involutory Hadamard-Cauchy matrices

Despite having a new technique to reduce the search space, the computation cost for checking the MDS property is still too huge when the order of the Hadamard matrix is larger than 8. Therefore, we use the Hadamard-Cauchy construction for order 16 and 32. Thanks to the Cauchy property, we are ensured that the matrix will be MDS. Hence, the only problem that remains is the huge search space of possible Hadamard-Cauchy matrices. To prevent confusion with Hadamard matrices, we denote Hadamard-Cauchy matrices with K.

First, we restate in Algorithm 1 the technique from [20] to build involutory MDS matrices, with some modifications on the notations for the variables. Although it is not explicitly stated, we can infer from Lemma 6,7 and Theorem 4 from [20] that all Hadamard-Cauchy matrices can be expressed as an output of Algorithm 1.

Algorithm 1 Construction of $2^s \times 2^s$ MDS matrix or involutory MDS matrix over $\mathrm{GF}(2^r)/p(X)$.

INPUT: an irreducible polynomial $p(X)$ of $\mathrm{GF}(2^r)$, integers s, r satisfying $s < r$ and $r > 1$, a boolean $B_{involutory}$.
OUTPUT: $2^s \times 2^s$ Hadamard-Cauchy matrix K, where K is involutory if $B_{involutory}$ is set `True`.

procedure CONSTRUCTH-C$(r, p(X), s, B_{involutory})$
  select s linearly independent elements $x_1, x_2, x_{2^2}, ..., x_{2^{s-1}}$ from $\mathrm{GF}(2^r)$ and construct S, the set of 2^s elements x_i,
    where $x_i = \bigoplus_{t=0}^{s-1} b_t x_{2^t}$ **for all** $i \in [0, 2^s - 1]$ (with $(b_{s-1}, b_{s-2}, ..., b_1, b_0)$ being the binary representation of i)
  select $z \in \mathrm{GF}(2^r) \setminus S$ and construct the set of 2^s elements y_i, where $y_i = z + x_i$ **for all** $i \in [0, 2^s - 1]$
  initialize an empty array ary_s of size 2^s
  if ($B_{involutory}$ == `False`) **then**
    $ary\_s[i] = \frac{1}{y_i}$ **for all** $i \in [0, 2^s - 1]$
  else
    $ary\_s[i] = \frac{1}{c \cdot y_i}$ **for all** $i \in [0, 2^s - 1]$, where $c = \bigoplus_{t=0}^{s-1} \frac{1}{z + x_t}$
  end if
  construct the $2^s \times 2^s$ matrix K, where $K[i, j] = ary\_s[i \oplus j]$
  return K
end procedure

Similarly to Hadamard matrices, we denote a Hadamard-Cauchy matrix by its first row of elements as $hc(h_0, h_1, ..., h_{2^s-1})$, with $h_i = K[0, i]$. To summarize the construction of a Hadamard-Cauchy matrix of order 2^s mentioned in Algorithm 1, we pick an ordered set of $s + 1$ linearly independent elements, we call it the basis. We use the first s elements to span an ordered set S of 2^s elements, and add the last element z to all the elements in S. Next, we take the inverse of each of the elements in this new set and we get the first row of the Hadamard-Cauchy matrix. Lastly, we generate the matrix based on the first row in the same manner as an Hadamard matrix.

For example, for an 8×8 Hadamard-Cauchy matrix over $\mathrm{GF}(2^4)$/`0x13`, say we choose $x_1 = 1, x_2 = 2, x_4 = 4$, we generate the set $S = \{0, 1, 2, 3, 4, 5, 6, 7\}$, choosing $z = 8$ and taking the inverses in the new set, we get a Hadamard-Cauchy matrix $K = hc(15, 2, 12, 5, 10, 4, 3, 8)$. To make it involution, we multiply each element by the inverse of the sum of the elements. However for this instance the sum is 1, hence K is already an involutory MDS matrix.

One of the main differences between the Hadamard and Hadamard-Cauchy matrices is the choice of entries. While we can choose all the entries for a

Hadamard matrix to be lightweight and permute them in search for an MDS candidate, the construction of Hadamard-Cauchy matrix makes it nontrivial to control its entries efficiently. Although in [20] the authors proposed a backward re-construction algorithm that finds a Hadamard-Cauchy matrix with some pre-decided lightweight entries, the number of entries that can be decided beforehand is very limited. For example, for a Hadamard-Cauchy matrix of order 16, the algorithm can only choose 5 lightweight entries, the weight of the other 11 entries is not controlled. The most direct way to find a lightweight Hadamard-Cauchy matrix is to apply Algorithm 1 repeatedly for all possible basis. We introduce now new equivalence classes that will help us to exhaust all possible Hadamard-Cauchy matrices with much lesser memory space and number of iterations.

Definition 8 *Let K_1 and K_2 be two Hadamard-Cauchy matrices, we say they are related, $K_1 \sim_{HC} K_2$, if one can be transformed to the other by either one or both operations on the first row of entries:*

1. *multiply by a nonzero scalar, and*
2. *$\mathcal{H}$-permutation of the entries.*

The crucial property of the construction is the independence of the elements in the basis, which is not affected by multiplying a nonzero scalar. Hence, we can convert any Hadamard-Cauchy matrix to an involutory Hadamard-Cauchy matrix by multiplying it with the inverse of the sum of the first row and vice versa. However, permutating the positions of the entries is the tricky part. Indeed, for the Hadamard-Cauchy matrices of order 8 or higher, some permutations destroy the Cauchy property, causing it to be non-MDS. Using our previous 8×8 example, suppose we swap the first two entries, $K' = hc(2, 15, 12, 5, 10, 4, 3, 8)$, it can be verified that it is not MDS. To understand why, we work backwards to find the basis corresponding to K'. Taking the inverse of the entries, we have $\{9, 8, 10, 11, 12, 13, 14, 15\}$. However, there is no basis that satisfies the 8 linear equations for the entries. Thus it is an invalid construction of Hadamard-Cauchy matrix. Therefore, we consider applying the $\mathcal{H}$-permutation on Hadamard-Cauchy matrix. Since it is also a Hadamard matrix, the $\mathcal{H}$-permutation preserves its branch number, thus it is still MDS. So we are left to show that a Hadamard-Cauchy matrix that undergoes $\mathcal{H}$-permutation is still a Hadamard-Cauchy matrix.

Lemma 4 *Given a $2^s \times 2^s$ involutory Hadamard-Cauchy matrix K, there are $2^s \cdot \prod_{i=0}^{s-1}(2^s - 2^i)$ involutory Hadamard-Cauchy matrices that are related to K by the $\mathcal{H}$-permutations of the entries of the first row.*

With that, we can define our equivalence classes of involutory Hadamard-Cauchy matrices.

Definition 9 *An equivalence class of involutory Hadamard-Cauchy matrices is a set of Hadamard-Cauchy matrices satisfying the equivalence relation $\sim_{HC}$.*

In order to count the number of equivalence classes of involutory Hadamard-Cauchy matrices, we use the same technique for proving Theorem 3. To do that, we need to know the total number of Hadamard-Cauchy matrices that can be constructed from the Algorithm 1 for a given finite field.

Lemma 5 *Given two natural numbers s and r, based on Algorithm 1, there are* $\prod_{i=0}^{s}(2^r - 2^i)$ *many* $2^s \times 2^s$ *Hadamard-Cauchy matrices over* $\mathrm{GF}(2^r)$.

Theorem 4 *Given two positive integers s and r, there are* $\prod_{i=0}^{s-1} \frac{2^{r-1}-2^i}{2^s-2^i}$ *equivalence classes of involutory Hadamard-Cauchy matrices of order* 2^s *over* $\mathrm{GF}(2^r)$.

In [15], the authors introduced the notion of compact Cauchy matrices which are defined as Cauchy matrices with exactly 2^s distinct elements. These matrices seem to include Cauchy matrices beyond the class of Hadamard-Cauchy matrices. However, it turns out that the equivalence classes of involutory Hadamard-Cauchy matrices can be extended to compact Cauchy matrices.

Corollary 1 *Any compact Cauchy matrices can be generated from some equivalence class of involutory Hadamard-Cauchy matrices.*

Note that since the permutation of the elements in S and $z + S$ only results in rearrangement of the entries of the compact Cauchy matrix, the XOR count is invariant from Hadamard-Cauchy matrix with the same set of entries.

5 Searching for involutory MDS and non-involutory MDS matrices

Due to space constraints, we have put respectively in Appendix B and C the new methods we have designed to look for the lightest possible involutory MDS and non-involutory MDS matrices.

More precisely, regarding involutory MDS matrices (see Appendix B), using the previous properties and equivalence classes given in Sections 3 and 4 for several matrix constructions, we have derived algorithms to search for the most lightweight candidate. First, we point out that the circulant construction can not lead to involutory MDS matrices, then we focus on the case of matrices of small dimension using the Hadamard construction. For bigger dimension, we add the Cauchy property to the Hadamard one in order to guarantee that the matrix will be MDS. We recall that, similarity to a circulant matrix, an Hadamard matrix will have the interesting property that each row is a permutation of the first row, therefore allowing to reuse the multiplication circuit to save implementation costs.

Regarding non-involutory MDS matrices (see Appendix C), we have extended the involutory MDS matrix search to include non-involutory candidates. For

Hadamard construction, we removed the constraint that the sum of the first row elements must be equal to 1. For the Hadamard-Cauchy, we multiply each equivalent classes by a non-zero scalar value. We note that the disadvantage of non-involutory MDS matrices is that their inverse may have a high computation cost. But if the inverse is not required (for example in the case of popular constructions such as a Feistel network, or a CTR encryption mode), non-involution matrices might be lighter than involutory matrices.

6 Results

We first emphasize that although in [20, 15] the authors proposed methods to construct lightweight matrices, the choice of the entries are limited as mentioned in Section 4.2. This is due to the nature of the Cauchy matrices where the inverse of the elements are used during the construction, which makes it non-trivial to search for lightweight Cauchy matrices[7]. However, using the concept of equivalence classes, we can exhaust all the matrices and pick the lightest-weight matrix.

We applied the algorithms of Section 5 to construct lightweight MDS involutions over $GF(2^8)$. We list them in the upper half of Table 1 and we can see that they are much lighter than known MDS involutions like the `KHAZAD` and `ANUBIS`, previous Hadamard-Cauchy matrices [6, 20] and compact Cauchy matrices [15]. In lower half of Table 1, we list the $GF(2^8)$ MDS matrices we found using the methods of Appendix C and show that they are lighter than known MDS matrices like the `AES`, `WHIRLPOOL` and `WHIRLWIND` matrices [17, 8, 7]. We also compare with the 14 lightweight candidate matrices C_0 to C_{13} for the `WHIRLPOOL` hash functions suggested during the NESSIE workshop [30, Section 6]. Table 1 is comparing our matrices with the ones explicitly provided in the previous articles. Recently, Gupta *et al.* [21] constructed some circulant matrices that is lightweight for both itself and its inverse. However we do not compare them in our table because their approach minimizes the number of XORs, look-up tables and temporary variables, which might be optimal for software but not for hardware implementations based purely on XOR count.

By Theorem 2 in Section 2, we only need to apply the algorithms from Section 5 for half the representations of $GF(2^8)$ when searching for optimal lightweight matrices. And as predicted by the discussion after Theorem 1, the lightweight matrices we found in Table 1 do come from $GF(2^8)$ representations with higher standard deviations.

We provide in the first column of the Table 1 the type of the matrices. They can be circulant, Hadamard or Cauchy-Hadamard. The subfield-Hadamard construction is based on the method of [24, Section 7.2] which we explain here.

[7] Using direct construction, there is no clear implication for the choice of the elements α_i and β_j that will generate lightweight entries c_{ij}. On the other hand, every lightweight entry chosen beforehand will greatly restrict the choices for the remaining entries if one wants to maintain two disjoint sets of elements $\{\alpha_i\}$ and $\{\beta_j\}$.

Consider the MDS involution $M = had(\mathsf{0x1}, \mathsf{0x4}, \mathsf{0x9}, \mathsf{0xd})$ over $\mathrm{GF}(2^4)/\mathsf{0x13}$ in the first row of Table 1. Using the method of [24, Section 7.2], we can extend it to a MDS involution over $\mathrm{GF}(2^8)$ by using two parallel copies of Q. The matrix is formed by writing each input byte x_j as a concatenation of two nibbles $x_j = (x_j^L || x_j^R)$. Then the MDS multiplication is computed on each half $(y_1^L, y_2^L, y_3^L, y_4^L) = M \cdot (x_1^L, x_2^L, x_3^L, x_4^L)$ and $(y_1^R, y_2^R, y_3^R, y_4^R) = M \cdot (x_1^R, x_2^R, x_3^R, x_4^R)$ over $\mathrm{GF}(2^4)$. The result is concatenated to form four output bytes (y_1, y_2, y_3, y_4) where $y_j = (y_j^L || y_j^R)$.

We could have concatenated different submatrices and this is done in the `WHIRLWIND` hash function [7], where the authors concatenated four MDS submatrices over $\mathrm{GF}(2^4)$ to form $(M_0|M_1|M_1|M_0)$, an MDS matrix over $\mathrm{GF}(2^{16})$. The submatrices are non-involutory Hadamard matrices $M_0 = had(\mathsf{0x5}, \mathsf{0x4}, \mathsf{0xa}, \mathsf{0x6}, \mathsf{0x2}, \mathsf{0xd}, \mathsf{0x8}, \mathsf{0x3})$ and $M_1 = (\mathsf{0x5}, \mathsf{0xe}, \mathsf{0x4}, \mathsf{0x7}, \mathsf{0x1}, \mathsf{0x3}, \mathsf{0xf}, \mathsf{0x8})$ defined over $\mathrm{GF}(2^4)/\mathsf{0x13}$. For fair comparison with our $\mathrm{GF}(2^8)$ matrices in Table 1, we consider the corresponding `WHIRLWIND`-like matrix $(M_0|M_1)$ over $\mathrm{GF}(2^8)$ which takes half the resource of the original `WHIRLWIND` matrix and is also MDS.

The second column of the result tables gives the finite field over which the matrix is defined, while the third column displays the first row of the matrix where the entries are bytes written in hexadecimal notation. The fourth column gives the XOR count to implement the first row of the $n \times n$ matrix. Because all subsequent rows are just permutations of the first row, the XOR count to implement the matrix is just n times this number. For example, to compute the XOR count for implementing $had(\mathsf{0x1}, \mathsf{0x4}, \mathsf{0x9}, \mathsf{0xd})$ over $\mathrm{GF}(2^4)/\mathsf{0x13}$, we consider the expression for the first row of matrix multiplication $\mathsf{0x1} \cdot x_1 \oplus \mathsf{0x4} \cdot x_2 \oplus \mathsf{0x9} \cdot x_3 \oplus \mathsf{0xd} \cdot x_4$. From Table 4 of Appendix F, the XOR count of multiplication by $\mathsf{0x1}, \mathsf{0x4}, \mathsf{0x9}$ and $\mathsf{0xd}$ are 0, 2, 1 and 3, which gives us a cost of $(0 + 2 + 1 + 3) + 3 \times 4 = 18$ XORs to implement one row of the matrix (the summand 3×4 account for the three XORs summing the four nibbles). For the subfield construction over $\mathrm{GF}(2^8)$, we need two copies of the matrix giving a cost of $18 \times 2 = 36$ XORs to implement one row.

Due to page constraints, we only give comparisons with known lightweight matrices over $\mathrm{GF}(2^8)$. The comparisons with $\mathrm{GF}(2^4)$ matrices will be provided in the full version of the paper. In fact, the subfield-Hadamard constructions in Table 1 already captures lightweight $\mathrm{GF}(2^4)$ matrices, and we show that our construction are lighter than known ones. For example in the lower half of Table 1, the $\mathrm{GF}(2^4)$ matrices M_0 and M_1 used in the `WHIRLWIND` hash function has XOR count 61 and 67 respectively while our Hadamard matrix $had(\mathsf{0x1}, \mathsf{0x2}, \mathsf{0x6}, \mathsf{0x8}, \mathsf{0x9}, \mathsf{0xc}, \mathsf{0xd}, \mathsf{0xa})$ has XOR count 54.

With our work, we can now see that one can use involutory MDS for almost the same price as non-involutory MDS. For example in the upper half of Table 1, the previous 4×4 MDS involution from [20] is about 3 times heavier than the `AES` matrix[8]; but in this paper, we have used an improved search technique to find

[8] We acknowledge that there are implementations that requires lesser XOR to implement directly the entire circulant `AES` matrix. However, the small savings obtained

an MDS involution lighter than the AES and ANUBIS matrix. Similarly, we have found 8×8 MDS involutions which are much lighter than the KHAZAD involution matrix, and even lighter than lightweight non-involutory MDS matrix like the WHIRLPOOL matrix. Thus, our method will be useful for future construction of lightweight ciphers based on involutory components like the ANUBIS, KHAZAD, ICEBERG and PRINCE ciphers.

on XOR count are completely outweighed by the extra memory cost required for such an implementation in terms of temporary variables.

Table 1: Comparison of MDS Matrices over GF(2^8). The upper table compares the involutory MDS matrices, while the lower table compares the non-involutory MDS matrices (the factor 2 is due to the fact that we have to implement two copies of the matrices)

INVOLUTORY MDS MATRICES

matrix type	finite field	coefficients of the first row	XOR count	reference
4×4 **matrix**				
Subfield-Hadamard	GF(2^4)/0x13	(0x1, 0x4, 0x9, 0xd)	$2 \times (6 + 3 \times 4) = \mathbf{36}$	Section B.2
Hadamard	GF(2^8)/0x165	(0x01, 0x02, 0xb0, 0xb2)	$16 + 3 \times 8 = \mathbf{40}$	Section B.2
Hadamard	GF(2^8)/0x11d	(0x01, 0x02, 0x04, 0x06)	$22 + 3 \times 8 = \mathbf{46}$	ANUBIS [5]
Compact Cauchy	GF(2^8)/0x11b	(0x01, 0x12, 0x04, 0x16)	$54 + 3 \times 8 = \mathbf{78}$	[15]
Hadamard-Cauchy	GF(2^8)/0x11b	(0x01, 0x02, 0xfc, 0xfe)	$74 + 3 \times 8 = \mathbf{98}$	[20]
8×8 **matrix**				
Hadamard	GF(2^8)/0x1c3	(0x01, 0x02, 0x03, 0x91, 0x04, 0x70, 0x05, 0xe1)	$46 + 7 \times 8 = \mathbf{102}$	Section B.2
Subfield-Hadamard	GF(2^4)/0x13	(0x2, 0x3, 0x4, 0xc, 0x5, 0xa, 0x8, 0xf)	$2 \times (36 + 7 \times 4) = \mathbf{128}$	Section B.2
Hadamard	GF(2^8)/0x11d	(0x01, 0x03, 0x04, 0x05, 0x06, 0x08, 0x0b, 0x07)	$98 + 7 \times 8 = \mathbf{154}$	KHAZAD [6]
Hadamard-Cauchy	GF(2^8)/0x11b	(0x01, 0x02, 0x06, 0x8c, 0x30, 0xfb, 0x87, 0xc4)	$122 + 7 \times 8 = \mathbf{178}$	[20]
16×16 **matrix**				
Hadamard-Cauchy	GF(2^8)/0x1c3	(0x08, 0x16, 0x8a, 0x01, 0x70, 0x8d, 0x24, 0x76, 0xa8, 0x91, 0xad, 0x48, 0x05, 0xb5, 0xaf, 0xf8)	$258 + 15 \times 8 = \mathbf{378}$	Section B.3
Hadamard-Cauchy	GF(2^8)/0x11b	(0x01, 0x03, 0x08, 0xb2, 0x0d, 0x60, 0xe8, 0x1c, 0x0f, 0x2c, 0xa2, 0x8b, 0xc9, 0x7a, 0xac, 0x35)	$338 + 15 \times 8 = \mathbf{458}$	[20]
32×32 **matrix**				
Hadamard-Cauchy	GF(2^8)/0x165	(0xd2, 0x06, 0x05, 0x4d, 0x21, 0xf8, 0x11, 0x62, 0x08, 0xd8, 0xe9, 0x28, 0x4b, 0x96, 0x10, 0x2c, 0xa1, 0x49, 0x4c, 0xd1, 0x59, 0xb2, 0x13, 0xa4, 0x03, 0xc3, 0x42, 0x79, 0xa0, 0x6f, 0xab, 0x41)	$610 + 31 \times 8 = \mathbf{858}$	Section B.3
Hadamard-Cauchy	GF(2^8)/0x11b	(0x01, 0x02, 0x04, 0x69, 0x07, 0xec, 0xcc, 0x72, 0x0b, 0x54, 0x29, 0xbe, 0x74, 0xf9, 0xc4, 0x87, 0x0e, 0x47, 0xc2, 0xc3, 0x39, 0x8e, 0x1c, 0x85, 0x58, 0x26, 0x1e, 0xaf, 0x68, 0xb6, 0x59, 0x1f)	$675 + 31 \times 8 = \mathbf{923}$	[20]

NON-INVOLUTORY MDS MATRICES

matrix type	finite field	coefficients of the first row	XOR count	reference
4×4 **matrix**				
Subfield-Hadamard	GF(2^4)/0x13	(0x1, 0x2, 0x8, 0x9)	$2 \times (5 + 3 \times 4) = \mathbf{34}$	Section C.2
Hadamard	GF(2^8)/0x1c3	(0x01, 0x02, 0x04, 0x91)	$13 + 3 \times 8 = \mathbf{37}$	Section C.2
Circulant	GF(2^8)/0x11b	(0x02, 0x03, 0x01, 0x01)	$14 + 3 \times 8 = \mathbf{38}$	AES [17]
8×8 **matrix**				
Hadamard	GF(2^8)/0x1c3	(0x01, 0x02, 0x03, 0x08, 0x04, 0x91, 0xe1, 0xa9)	$40 + 7 \times 8 = \mathbf{96}$	Section C.2
Circulant	GF(2^8)/0x11d	(0x01, 0x01, 0x04, 0x01, 0x08, 0x05, 0x02, 0x09)	$49 + 7 \times 8 = \mathbf{105}$	WHIRLPOOL [8]
Subfield-Hadamard	GF(2^4)/0x13	(0x1, 0x2, 0x6, 0x8, 0x9, 0xc, 0xd, 0xa)	$2 \times (26 + 7 \times 4) = \mathbf{108}$	Section C.2
Circulant	GF(2^8)/0x11d	WHIRLPOOL-like matrices	between **105** to **117**	[30]
Subfield-Hadamard	GF(2^4)/0x13	WHIRLWIND-like matrix	$33 + 39 + 2 \times 7 \times 4 = \mathbf{128}$	[7]
16×16 **matrix**				
Hadamard-Cauchy	GF(2^8)/0x1c3	(0xb1, 0x1c, 0x30, 0x09, 0x08, 0x91, 0x18, 0xe4, 0x98, 0x12, 0x70, 0xb5, 0x97, 0x90, 0xa9, 0x5b)	$232 + 15 \times 8 = \mathbf{352}$	Section C.3
32×32 **matrix**				
Hadamard-Cauchy	GF(2^8)/0x1c3	(0xb9, 0x7c, 0x93, 0xbc, 0xbd, 0x26, 0xfa, 0xa9, 0x32, 0x31, 0x24, 0xb5, 0xbb, 0x06, 0xa0, 0x44, 0x95, 0xb3, 0x0c, 0x1c, 0x07, 0xe5, 0xa4, 0x2e, 0x56, 0x4c, 0x55, 0x02, 0x66, 0x39, 0x48, 0x08)	$596 + 31 \times 8 = \mathbf{844}$	Section C.3

Acknowledgments

The authors would like to thank the anonymous referees for their helpful comments. We also wish to thank Wang HuaXiong for providing useful and valuable suggestions.

References

1. E. Andreeva, B. Bilgin, A. Bogdanov, A. Luykx, F. Mendel, B. Mennink, N. Mouha, Q. Wang, and K. Yasuda. PRIMATEs v1. Submission to the CAESAR Competition, 2014. `http://competitions.cr.yp.to/round1/primatesv1.pdf`.
2. D. Augot and M. Finiasz. Direct Construction of Recursive MDS Diffusion Layers using Shortened BCH Codes. In *FSE*, LNCS, 2014. To appear.
3. Daniel Augot and Matthieu Finiasz. Exhaustive search for small dimension recursive MDS diffusion layers for block ciphers and hash functions. In *ISIT*, pages 1551–1555, 2013.
4. Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A Lightweight Hash. In *CHES*, pages 1–15, 2010.
5. P. Barreto and V. Rijmen. The Anubis Block Cipher. Submission to the NESSIE Project, 2000.
6. P. Barreto and V. Rijmen. The Khazad Legacy-Level Block Cipher. First Open NESSIE Workshop, 2000.
7. Paulo S. L. M. Barreto, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Elmar Tischhauser. Whirlwind: a new cryptographic hash function. *Des. Codes Cryptography*, 56(2-3):141–162, 2010.
8. Paulo S. L. M. Barreto and Vincent Rijmen. Whirlpool. In *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 1384–1385. 2011.
9. Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013.
10. Thierry P. Berger. Construction of Recursive MDS Diffusion Layers from Gabidulin Codes. In *INDOCRYPT*, volume 8250 of *LNCS*, pages 274–285. Springer, 2013.
11. A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede. spongent: A Lightweight Hash Function. In *CHES*, pages 312–325, 2011.
12. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007.
13. J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In *ASIACRYPT*, pages 208–225, 2012.
14. C. De Cannière, O. Dunkelman, and M. Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *CHES*, pages 272–288, 2009.
15. T. Cui, C.i Jin, and Z. Kong. On compact cauchy matrices for substitution-permutation networks. *IEEE Transactions on Computers*, 99(PrePrints):1, 2014.
16. Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. In Eli Biham, editor, *FSE*, volume 1267 of *LNCS*, pages 149–165. Springer, 1997.

17. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
18. Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON Family of Lightweight Hash Functions. In *CRYPTO*, pages 222–239, 2011.
19. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In *CHES*, pages 326–341, 2011.
20. Kishan Chand Gupta and Indranil Ghosh Ray. On Constructions of Involutory MDS Matrices. In *AFRICACRYPT*, pages 43–60, 2013.
21. Kishan Chand Gupta and Indranil Ghosh Ray. On Constructions of Circulant MDS Matrices for Lightweight Cryptography. In *ISPEC*, pages 564–576, 2014.
22. Jorge Nakahara Jr. and lcio Abraho. A new involutory mds matrix for the aes. *I. J. Network Security*, 9(2):109–116, 2009.
23. Pascal Junod and Serge Vaudenay. Perfect Diffusion Primitives for Block Ciphers. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography*, volume 3357 of *LNCS*, pages 84–99. Springer, 2004.
24. K. Khoo, T. Peyrin, A. Poschmann, and H. Yap. FOAM: Searching for Hardware-Optimal SPN Structures and Components with a Fair Comparison. In *Cryptographic Hardware and Embedded Systems CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 433–450. Springer Berlin Heidelberg, 2014.
25. Jérôme Lacan and Jérôme Fimes. Systematic MDS erasure codes based on Vandermonde matrices. *IEEE Communications Letters*, 8(9):570–572, 2004.
26. S. Ling and C. Xing. *Coding Theory: A First Course*. Coding Theory: A First Course. Cambridge University Press, 2004.
27. F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 2nd edition, 1986.
28. M. Sajadieh, M. Dakhilalian, H. Mala, and P. Sepehrdad. Recursive Diffusion Layers for Block Ciphers and Hash Functions. In *FSE*, pages 385–401, 2012.
29. M.i 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.
30. T. Shirai and K. Shibutani. On the diffusion matrix employed in the Whirlpool hashing function. NESSIE Phase 2 Report NES/DOC/EXT/WP5/002/1.
31. François-Xavier Standaert, Gilles Piret, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. ICEBERG : An Involutional Cipher Efficient for Block Encryption in Reconfigurable Hardware. In *FSE*, pages 279–299, 2004.
32. Shengbao Wu, Mingsheng Wang, and Wenling Wu. Recursive Diffusion Layers for (Lightweight) Block Ciphers and Hash Functions. In *Selected Areas in Cryptography*, volume 7707 of *LNCS*, pages 355–371. Springer Berlin Heidelberg, 2013.
33. A. M. Youssef, S. Mister, and S. E. Tavares. On the Design of Linear Transformations for Substitution Permutation Encryption Networks. In *Workshop On Selected Areas in Cryptography*, pages 40–48, 1997.

A Maximum Distance Separable matrices

Maximum Distance Separable matrices are crucial components in cryptographic designs, as they ensure a perfect diffusion layer. Since we will search among many lightweight candidate matrices and only keep the MDS ones, we recall in this subsection a few definitions and properties regarding these mathematical objects. We denote by I_k the $k \times k$ identity matrix.

Definition 10 *The branch number of a $k \times k$ matrix M over $\mathrm{GF}(2^r)$ is the minimum number of nonzero entries in the input vector v and output vector $v \cdot M = u$ (denoted $wt(v)$ and $wt(u)$ respectively), as we range over all $v \in [\mathrm{GF}(2^r)]^k - \{0\}$. I.e. the branching number is equal to $\min_{x\neq 0}\{wt(v) + wt(u)\}$, and when the optimal value $k+1$ is attained, we say M is an MDS matrix.*

Definition 11 *A length n, dimension k and distance d binary linear code $[n, k, d]$ is called a MDS code if the Singleton bound $k = n - d + 1$ is met.*

From [16, Section 4], we have the following proposition to relate an MDS matrix to a MDS code.

Proposition 2 *A $k \times k$ matrix M is an MDS matrix if and only if the standard form generator matrix $[I_k|M]$ generates a $(2k, k, k+1)$-MDS code.*

There are various ways to verify if a matrix is MDS, in this paper we state two of the commonly used statements that can be used to identify MDS matrix.

Proposition 3 ([27], page 321, Theorem 8 - [26], page 53, Theorem 5.4.5) *Given a $k \times k$ matrix M, it is an MDS matrix if and only if*

1. *every square submatrix (formed from any i rows and any i columns, for any $i = 1, 2, ..., k$) of M is nonsingular,*
2. *any k columns of $[I_k|M]$ are linearly independent.*

The two following corollaries are directly deduced from the first statement of Proposition 3 when we consider submatrices of order 1 and 2 respectively.

Corollary 2 *All entries of an MDS matrix are nonzero.*

Corollary 3 *Given a $k \times k$ matrix M, if there exists pairwise distinct $i_1, i_2, j_1, j_2 \in \{0, 1, ..., k-1\}$ such that $M[i_1, j_1] = M[i_1, j_2] = M[i_2, j_1] = M[i_2, j_2]$, then M is not an MDS matrix.*

B Searching for MDS and involutory matrices

In this section, using the previous properties and equivalence classes given in Sections 3 and 4 for several matrix constructions, we will derive algorithms to search for lightweight involutory MDS matrices. First, we show that the circulant construction can not lead to such matrices, then we focus on the case of matrices of small dimension using the Hadamard construction. For bigger dimension, we add the Cauchy property to the Hadamard one in order to guarantee that the matrix will be MDS. We recall that, similarity to a circulant matrix, an Hadamard matrix will have the interesting property that each row is a permutation of the first row, therefore allowing to reuse the multiplication circuit to save implementation costs.

B.1 Circulant MDS involution matrix does not exist

The reason why we do not consider circulant matrices as potential candidates for MDS involution matrices is simple: it simply does not exist. In [22], the authors proved that circulant matrices of order 4 cannot be simultaneously MDS and involutory. And recently [21] proved that generic circulant MDS involutory matrices do not exist.

B.2 Small dimension lightweight MDS involution matrices

The computation complexity for checking if a matrix is MDS and the huge search space are two main complexity contributions to our exhaustive search algorithm for lightweight Hadamard MDS matrices. The latter is greatly reduced thanks to our equivalence classes and we now need an efficient algorithm for checking the MDS property. In this section, using properties of Hadamard matrix, we design a simple algorithm that can verify the MDS property faster than for usual matrices. First, let us prove some results using Proposition 3. Note that Lemma 6 and Corollary 4 are not restricted to Hadamard matrices. Also, Corollary 4 is the contra-positive of Lemma 6.

Lemma 6 *Given a $k \times k$ matrix M, there exists a $l \times l$ singular submatrix if and only if there exists a vector, $v \neq 0$, with at most l nonzero components such that $vM = u$ and the sum of nonzero components in v and u is at most k.*

Corollary 4 *Given a $k \times k$ matrix M, the sum of nonzero components of the input and output vector is at least $k + 1$ for any input vector v with l nonzero components if and only if all $l \times l$ submatrices of M are nonsingular.*

One direct way for checking the MDS property is to compute the determinant of all the submatrices of M and terminates the algorithm prematurely by returning **False** when a singular submatrix is found. If no such submatrix has been found among all the possible submatrices, the algorithm can return **True**. Using the fact that the product of a Hadamard matrix with itself is a multiple of an identity matrix, we can cut down the number of submatrices to be checked with the following proposition.

Proposition 4 *Given a $k \times k$ Hadamard matrix H with the sum c of first row being nonzero ($c \neq 0$), if all submatrices of order $l \leq \frac{k}{2}$ are nonsingular, then H is MDS.*

We can further reduce the computation complexity using the fact that Hadamard matrices are bisymmetric. Given a Hadamard matrix, we have four regions dissected by the left and right diagonal, namely top, left, right and bottom quadrant. For convention, we let the diagonal entries to be in both quadrants. See Figure 1 for illustration, where the top four entries "a" belong to both top and left quadrants, while the bottom four "a" belong to both bottom and right quadrant.

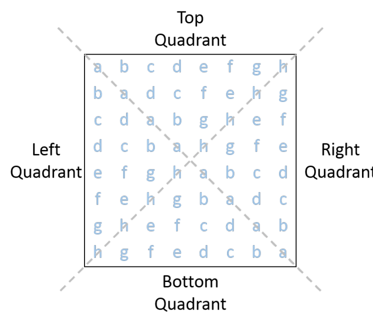


Fig. 1: The four quadrants of Hadamard matrix.

Proposition 5 *Given a $k \times k$ Hadamard matrix H, if all submatrices L with leading entry $L[0,0]$ in the top quadrant are nonsingular, then H is MDS.*

Thanks to Propositions 4 and 5, our algorithm for checking the MDS property of Hadamard matrices is now much faster than a naive method. Namely, given a $2^s \times 2^s$ Hadamard matrix, the algorithm to test its MDS property can be as follows. First, we check that all entries are nonzero and for $l = 2, \ldots, 2^{s-1}$ we check that the determinant of $l \times l$ submatrices with leading entry in top quadrant is nonzero. If one submatrix fails, we output **False**. Once all the submatrices are checked, we can output **True**.

Using this algorithm as the core procedure for checking the MDS property, we can find the lightest-weight MDS involution Hadamard matrix by choosing a set of elements that sum to 1 with the smallest XOR count, permute the entries as mentioned in Section 4.1 and use this core procedure to check if it is MDS. If all equivalence classes of Hadamard matrices are not MDS, we swap some element in the set with another element with a slightly higher XOR count and repeat the process until we find the first MDS involution Hadamard matrix with the lowest possible XOR count. Eventually, we found the lightest-weight MDS involution Hadamard matrix of order 4 and 8 over GF(2^4) and GF(2^8), which can be found in the upper half of Table 1 in Section 6. We emphasize that our results close the discussions on MDS involution Hadamard matrix of order 4 and 8, since our technique allows to take into account of all possible matrices.

B.3 Large dimension lightweight MDS involution matrices

The algorithm computation complexity grows exponentially with the matrix dimension, it is difficult to go to matrices of higher order. For that reason, we reduce the search space from Hadamard to Hadamard-Cauchy matrices, which guarantee the MDS property. Nevertheless, it is not feasible to generate and store all possible Hadamard-Cauchy matrices. For 16×16 Hadamard-Cauchy matrices over $GF(2^8)$, by Lemma 5 we know there are almost a trillion distinct candidates. This is where the idea of equivalence classes comes in handy again. By Theorem 4, instead of storing over 9.7×10^{11} matrices, all we need is to find the 11811 equivalence classes. Even if memory space is not an issue, using Algorithm 1 to exhaustively search for all Hadamard-Cauchy matrices requires about $2^{39.9}$ iterations. In this subsection, we propose a deterministic and randomized algorithm that only takes on average of $2^{16.9}$ iterations to find all the equivalence classes, which is equivalent to finding all possible Hadamard-Cauchy matrices.

First, we present two statements that are useful in designing the algorithm.

Lemma 7 *Based on Algorithm 1, given a basis of $s+1$ ordered elements $\{x_1, x_2, x_{2^2}, ..., x_{2^{s-1}}, z\}$, any permutation of the first s elements $\{\sigma(x_1), \sigma(x_2), \sigma(x_{2^2}), ..., \sigma(x_{2^{s-1}}), z\}$ will form a Hadamard-Cauchy matrix that belongs to the same equivalence class.*

Proposition 6 *Given two positive integers s and r, where $s < r$, doing exhaustive search through $1 \leq x_1 < x_2 < x_{2^2} < ... < x_{2^{s-1}} \leq 2^r$ and $1 \leq z \leq 2^r$ is sufficient to find all possible equivalence classes of involutory Hadamard-Cauchy matrices.*

We describe our search method in Algorithm 2 and one can see that it uses most of Algorithm 1 as core procedure. We denote ConstructH-C* the procedure `ConstructH-C` from Algorithm 2 where the values $x_1, x_2, x_{2^2}, ..., x_{2^{s-1}}$ and z are given as inputs instead of chosen in the procedure. We first choose $s+1$ linearly independent elements and apply Algorithm 1 to generate an involutory Hadamard-Cauchy matrix. We initialise an array *temp_mat* and a list *list_EC* to empty. Then, *temp_mat* is the matrix considered at the current iteration, it will be checked against *list_EC* which is the list of equivalence classes of involutory Hadamard-Cauchy matrices that have been found. If *temp_mat* is not a permutation of any matrix in *list_EC*, then a new equivalence class is found and *temp_mat* will be stored in *list_EC*. When all the equivalence classes are found, we terminates the algorithm, which will dramatically cut down the number of iterations required.

From a representative of an equivalence class, one can obtain all the involutory Hadamard-Cauchy matrices of the same equivalence class through $\mathcal{H}$-permutations. Note that the $\mathcal{H}$-permutation is also applicable to non-involutory Hadamard-Cauchy matrices.

Algorithm 2 Finding all $2^s \times 2^s$ equivalence classes of involutory Hadamard-Cauchy matrices over $\mathrm{GF}(2^r)/p(X)$.

INPUT: an irreducible polynomial $p(X)$ of $\mathrm{GF}(2^r)$, integers s, r satisfying $s < r$ and $r > 1$.
OUTPUT: a list of equivalence classes of involutory Hadamard-Cauchy matrix.

```
procedure GenECofInvH-C(r, p(X), s)
    compute the total number of equivalence classes, EC = ∏_{i=0}^{s-1} (2^{r-1} - 2^i)/(2^s - 2^i)
    initialize an empty set of arrays list_EC
    while (sizeof(list_EC) ≠ EC) do
        select s linearly independent elements x_1, x_2, x_{2^2}, ..., x_{2^{s-1}} from GF(2^r)/p(X)
in ascending order
        select element z as linearly independent of x_1, x_2, x_{2^2}, ..., x_{2^{s-1}}
        temp_mat = ConstructH-C*(r, p(X), s, True, x_1, x_2, x_{2^2}, ..., x_{2^{s-1}}, z)
        if temp_mat is not a permutation of any matrix in list_EC then
            store temp_mat into list_EC
        end if
    end while
    return list_EC
end procedure
```

We remark that for 2×2 and 4×4 Hadamard-Cauchy matrices, any permutation of the equivalence class is still an involutory Hadamard-Cauchy matrix.

Notice that Algorithm 2 is a deterministic search for the equivalence classes. To further reduce the iterations needed, we propose to choose the $s+1$ elements randomly. Using this randomized search, it takes about $2^{16.9}$ iterations before finding all the equivalence classes. Once all the equivalence classes of involutory Hadamard-Cauchy matrices are found, we can check which matrix has the lightest-weight.

Using the randomized search algorithm, we found the lightest-weight involution Hadamard-Cauchy matrix of order 16 and 32 over $\mathrm{GF}(2^8)$, which can be found in the upper half of Table 1.

C Searching for MDS matrices

The disadvantage of using non-involution matrices is that its inverse may have a high computation cost. But if the inverse is not required, non-involution matrices might be lighter than involutory matrices. In this paper, we look at encryption only and do not consider the reuse of component for encryption/decryption (which can be studied in future work). Note that the inverse of the matrix would not be required for popular constructions such as a Feistel network, or a CTR encryption mode.

C.1 Circulant matrices

As the discussion on lightweight MDS circulant matrix is well-explored in [24], we focus on Hadamard-based matrix and extend the exhaustive search for from involutory to non-involutory lightest-weight MDS matrix.

C.2 Small dimension lightweight MDS matrices

The results in Section 4.1 and B.2 can also be applied on non-involution Hadamard matrices. Thus the method of finding a lightweight MDS involution matrix is basically the same. We pick a set of low XOR count nonzero elements that does not sum to 0, else it would be non-MDS, and apply the permutation method which is discussed at the end of Section B.2 to check through all equivalence classes of Hadamard matrices.

C.3 Large dimension lightweight MDS matrices

After finding all the equivalence classes of involutory Hadamard-Cauchy matrices using the Algorithm 2, we can conveniently use this collection of equivalence classes to find lightest-weight non-involutory Hadamard-Cauchy matrix. That is to multiply by a nonzero scalar to each equivalence classes to generate all possible Hadamard-Cauchy matrices up to permutation. In this way, it is more efficient than exhaustive search on all possible Hadamard-Cauchy matrices as we eliminated all the permutations of the Hadamard-Cauchy matrices that have the same XOR count.

D Proofs

D.1 Proof of Theorem 1

We are interested in multiplying an arbitrary element α by β where $\alpha, \beta \in \mathrm{GF}(2^r)$. This can be done using a multiplication matrix $M_\beta \in \mathrm{GF}(2)^{r \times r}$, which by definition satisfies

$$(X^{r-1}, X^{r-2}, \dots, 1)M_\beta = (X^{r-1}\beta, X^{r-2}\beta, \dots, \beta).$$

To count the number of XORs needed to multiply α by β, it is enough to count the number of 1's per column of M_β: if there are i 1's, the number of XORs needed is $i - 1$.

Example 1 *Set $r = 2$, with irreducible polynomial $p(X) = X^2 + X + 1$. Then*

$$M_\beta = \begin{bmatrix} b_1 & b_0 + b_1 \\ b_0 & b_1 \end{bmatrix}$$

for $\beta = b_1 X + b_0$. Thus

- *when* $(b_1, b_0) = (0, 0)$*, the number of XORs is 0,*
- *when* $(b_1, b_0) = (0, 1)$*, the number of XORs is 0,*
- *when* $(b_1, b_0) = (1, 0)$*, the number of XORs is 1,*
- *when* $(b_1, b_0) = (1, 1)$*, the number of XORs is 1.*

This corresponds to the Table 4 in Appendix F.

Thus to count the total number of XORs needed when summing over all elements β, it is enough to count the number of 1's in the columns of M_β when summing over all possible β. The matrix M_β of course depends on the irreducible polynomial $p(X)$, however when summing over all β, the number of 1's that appears in each column does not depend on $p(X)$, as we prove next. The first column of M_β is $(b_{r-1}, \ldots, b_0)$ for $\beta = b_{r-1}X^{r-1} + \ldots + b_1X + b_0$.

Lemma 8 *The set* $\{(b_{r-1}, \ldots, b_0),\ b_i \in \mathrm{GF}(2),\ i = 0, \ldots, r-1\}$ *is in bijection with every column of* M_β*.*

Proof. This follows from $\mathrm{GF}(2^r)$ being a finite field, thus multiplication by any nonzero element is invertible. □

Corollary 5 *The number of XORs needed when summing over all* β *is* r *times the number counted in the first column of* M_β*.*

Lemma 9 *The number of XORs counted in the first column of* M_β *is* $\sum_{i=2}^{r} \binom{r}{i}(i-1)$*.*

Proof. It is enough to count the number of 1's in the vector $(b_{r-1}, \ldots, b_0)$ when all the b_i run through $\mathrm{GF}(2)$, $i = 0, \ldots, r-1$. There are $\binom{r}{i}$ possible patters of i 1's among the r coefficients, and whenever there are i 1's, the number of XORs needed is $(i-1)$. □

Corollary 6 *The total number of XORs needed when summing over all* β *is* $r\sum_{i=2}^{r} \binom{r}{i}(i-1)$*.*

Lemma 10 *We have that* $\sum_{i=2}^{r} \binom{r}{i}(i-1) = \sum_{i=2}^{r} 2^{i-2}(i-1)$*.*

Proof.
1. It is known that $\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}$. Thus $\sum_{i=2}^{r} \binom{r}{i}(i-1) = r2^{r-1} - 2^r + 1$.
2. It is known that $\sum_{k=1}^{n} kx^k = \frac{x-(n+1)x^{n+1}+nx^{n+2}}{(x-1)^2}$.

   Thus, we have $\sum_{i=2}^{r} 2^{i-2}(i-1) = 1 - r2^{r-1} + (r-1)2^r$. □

Corollary 7 *The number of XORs needed when summing over all* β *is* $r\sum_{i=2}^{r} 2^{i-2}(i-1)$*.*

D.2 Proof of Theorem 2

Proof. We already mentioned in the introduction that the finite field $\mathrm{GF}(2^r)/p(X)$ is isomorphic to polynomials in $\mathrm{GF}(2)[X]$ modulo the irreducible polynomial $p(X)$. Since $p(X) = 0$ in this field, we may alternatively describe $\mathrm{GF}(2^r)/p(X)$ as the field extension of $\mathrm{GF}(2)$, obtained by adding a root α of $p(X)$ to $\mathrm{GF}(2)$, in which case we write (and say) that $\mathrm{GF}(2^r)/p(X)$ is isomorphic to $\mathrm{GF}(2)(\alpha)$, with $p(\alpha) = 0$. Similarly $\mathrm{GF}(2^r)/\frac{1}{p}(X)$ contains an element, say β such that $\frac{1}{p}(\beta) = 0$, and $\mathrm{GF}(2^r)/\frac{1}{p}(X)$ is isomorphic to $\mathrm{GF}(2)(\beta)$.

Since

$$0 = \frac{1}{p}(\beta) = \beta^r p(\beta^{-1}),$$

it must be that $p(\beta^{-1}) = 0$. Write a generic element of $\mathrm{GF}(2)(\alpha)$ as $a_0 + a_1\alpha + \ldots + a_{r-1}\alpha^{r-1}$, $a_i \in \mathrm{GF}(2)$ by fixing $\{1, \ldots, \alpha^{r-1}\}$ as $\mathrm{GF}(2)$-basis, and similarly a generic element of $\mathrm{GF}(2)(\beta)$ as $b_0 + b_1\beta + \ldots + b_{r-1}\beta^{r-1}$, $b_i \in \mathrm{GF}(2)$, by fixing $\{1, \ldots, \beta^{r-1}\}$ as $\mathrm{GF}(2)$-basis. Define $\psi : \mathrm{GF}(2)(\alpha) \to \mathrm{GF}(2)(\beta)$ by $\psi : \sum_{i=0}^{r-1} a_i\alpha^i \mapsto \sum_{i=0}^{r-1} a_i\beta^{-i}$, $i = 0, \ldots, r-1$. Then ψ is a field isomorphism. Indeed

$$\begin{aligned}\psi(\sum_{i=0}^{r-1} a_i\alpha^i + \sum_{i=0}^{r-1} a_i'\alpha^i) &= \psi(\sum_{i=0}^{r-1}(a_i + a_i')\alpha^i)\\ &= \sum_{i=0}^{r-1}(a_i + a_i')\beta^{-i} = \psi(\sum_{i=0}^{r-1} a_i\alpha^i) + \psi(\sum_{i=0}^{r-1} a_i'\alpha^i)\end{aligned}$$

Also, to show that

$$\psi(\sum_{i=0}^{r-1} a_i\alpha^i \sum_{i=0}^{r-1} a_i'\alpha^i) = \psi(\sum_{i=0}^{r-1} a_i\alpha^i)\psi(\sum_{i=0}^{r-1} a_i'\alpha^i)$$

it is enough to show that $\psi(\alpha^r) = \psi(\alpha)^r$. Write $p(X) = p_0 + p_1X + \ldots + p_{r-1}X^{r-1} + X^r$. Now, recalling that α is a root of $p(X)$

$$\begin{aligned}\psi(\alpha^r) &= \psi(p_0 + p_1\alpha + \ldots + p_{r-1}\alpha^{r-1})\\ &= p_0 + p_1\psi(\alpha) + \ldots + p_{r-1}\psi(\alpha)^{r-1} = p_0 + p_1\beta^{-1} + \ldots + p_{r-1}\beta^{-r+1},\end{aligned}$$

while

$$\psi(\alpha)^r = \beta^{-r} = p_0 + p_1\beta^{-1} + \ldots + p_{r-1}\beta^{-r+1}$$

since $p(\beta^{-1}) = 0$. Note that ψ is necessarily injective since $\mathrm{GF}(2)(\alpha)$ is a field, ψ is then necessarily surjective since $|\mathrm{GF}(2)(\beta)|$ is finite. This shows that ψ is a field isomorphism.

Now α may or not be a primitive element. Recall that α is primitive if it is such that $\alpha^{2^r-1} = 1$ and there is no i, $0 < i < 2^r - 1$ such that $\alpha^i = 1$. Suppose first that α is a primitive element of $\mathrm{GF}(2)(\alpha)$ (this happens for example if $2^r - 1$ is prime). Take again a generic element of $\mathrm{GF}(2)(\alpha)$ as $a_0 + a_1\alpha + \ldots + a_{r-1}\alpha^{r-1}$,

$a_i \in \mathrm{GF}(2)$ by fixing the same GF(2)-basis, that is $\{1, \ldots, \alpha^{r-1}\}$. To compute the XOR of α^j in $\mathrm{GF}(2)(\alpha)$ (or equivalently in $\mathrm{GF}(2^r)/p(X)$), compute

$$(a_0 + a_1\alpha + \ldots + a_{r-1}\alpha^{r-1})\alpha^j,\ 1 \leq j \leq 2^r - 1$$

since α is primitive. The distribution of XOR counts obtained that way is the same as the distribution of XOR counts while computing instead

$$(a_0 + a_1\alpha^{-1} + \ldots + a_{r-1}\alpha^{-(r-1)})\alpha^j = \sum_{i=0}^{r-1} d_{ij}\alpha^{-i}$$

where d_{ij}, $0 \leq i \leq r-1$, decides the number of XOR of α^j, $1 \leq j \leq 2^r-1$. Indeed, the sets $\{\sum_{i=0}^{r-1} a_i\alpha^{i+j},\ 1 \leq j \leq 2^r - 1\}$ and $\{\sum_{i=0}^{r-1} a_i\alpha^{-i+j},\ 1 \leq j \leq 2^r - 1\}$ are the same, up to relabeling the a_i and recalling that $\alpha^{2^r-1} = 1$. Furthermore, the computations of α^i and α^{-i} need the same number of XOR, since $p(\alpha)$ and $\alpha^{-r}p(\alpha)$ have the same number of non-zero coefficients. Then

$$\begin{aligned}\psi((a_0 + a_1\alpha^{-1} + \ldots + a_{r-1}\alpha^{-(r-1)})\alpha^j) &= (a_0 + a_1\psi(\alpha)^{-1} + \ldots + a_{r-1}\psi(\alpha)^{-(r-1)})\psi(\alpha^j)\\ &= (a_0 + a_1\beta + \ldots + a_{r-1}\beta^{r-1})\beta^{-j}\\ &= \sum_{i=0}^{r-1} d_{ij}\beta^i\end{aligned}$$

thus the number of XOR of the element β^{-j} in $\mathrm{GF}(2)(\beta)$ in the GF(2)-basis $\{1, \ldots, \beta^{r-1}\}$ is the same as the XOR count of α^j in $\mathrm{GF}(2)(\alpha)$.

If α is not primitive, take α' a primitive element of $\mathrm{GF}(2)(\alpha)$, write it in the GF(2)-basis $\{1, \alpha, \ldots, \alpha^{r-1}\}$ and apply the same argument on α^i.

Consider for instance the finite field $\mathrm{GF}(2^4)/$`0x13` and $\mathrm{GF}(2^4)/$`0x19`, corresponding to the polynomials $p(X) = X^4+X+1$ and $\frac{1}{p}(X) = X^4+X^3+1$ respectively. In $\mathrm{GF}(2^4)/$`0x13`, 2 is a primitive element. In $\mathrm{GF}(2^4)/$`0x19`, compute the inverse of the polynomial X, which is X^3+X^2 since $X(X^3+X^2) = X^4+X^3 = 1 \pmod{X^4 + X + 1}$. The isomorphism ψ is thus sending 2 to 12. □

D.3 Proof of Proposition 1

Proof. If two Hadamard matrices H_1 and H_2 are equivalent, $H_1 \sim H_2$, then for every pair of input and output vectors for H_1, there is a corresponding pair of vectors for H_2 with the same sum of nonzero components. Therefore, by taking the minimum over all pairs, we deduce that both matrices have the same branch number. □

D.4 Proof of Lemma 1

Proof. By definition of the Hadamard matrix, we can express the two matrices as $H = had(h_0, h_1, ..., h_{k-1})$ and $H^{(\alpha)} = had(h_\alpha, h_{\alpha\oplus1}, ..., h_{\alpha\oplus(k-1)})$. Let $v =$

$(v_0, v_1, \ldots, v_{k-1})$ and $v^{(\alpha)} = (v_\alpha, v_{\alpha\oplus 1}, \ldots, v_{\alpha\oplus(k-1)})$ be the input vector for H and $H^{(\alpha)}$, u and $u^{(\alpha)}$ be the output vector respectively.

From our example with σ_1, we see that if the same permutation α is applied to H and to the input vector v, the output vectors are equal, i.e. $u^{(\alpha)} = u$. This is indeed true in general, it is known that the $(j+1)$-th component of the output vector is the sum (or XOR as we are working over $\mathrm{GF}(2^r)$) of the product of the input vector and $(j+1)$-th column of the matrix. We can express the $(j+1)$-th component of $u^{(\alpha)}$ as

$$u_j^{(\alpha)} = \bigoplus_{i=0}^{k-1} v_i^{(\alpha)} H^{(\alpha)}[i,j] = \bigoplus_{i=0}^{k-1} v_{\alpha\oplus i} h_{\alpha\oplus i\oplus j},$$

since XOR is commutative, the order of XOR is invariant, therefore $u_j^{(\alpha)} = u_j$. □

D.5 Proof of Lemma 3

Proof. For simplicity, we see how the indices of the elements are permuted. As mentioned, we need to map identity to itself, $\sigma(0) = 0$. After index 0 is fixed, index 1 can be mapped to any of the remaining $2^s - 1$ indices. Similarly for index 2, there are $2^s - 2$ choices. But for index 3, because of the linear relation, its image is defined by the mapping of index 1 and 2: $\sigma(3) = \sigma(1) \oplus \sigma(2)$.

Following this pattern, we can choose the permutation for index 4 among the $2^s - 4$ index, while 5, 6 and 7 are defined by $\sigma(1)$, $\sigma(2)$ and $\sigma(4)$. Therefore, the total number of possible permutations is

$$(2^s - 1)(2^s - 2)(2^s - 4)\ldots(2^s - 2^{s-1}) = \prod_{i=0}^{s-1} (2^s - 2^i).$$ □

D.6 Proof of Theorem 3

Proof. To prove this theorem, we use the *double counting* proof technique that is commonly used in combinatorics. We count the total number of permutations of Hadamard matrices for a given set of elements.
Counting 1: there is a total of $(2^s)!$ permutations for the given set of elements.
Counting 2: for each of the equivalence classes of Hadamard matrix, by Lemma 2 and 3, there are $\prod_{i=0}^{s-1}(2^s - 2^i)$ linear permutations. For each of these permutations, by Lemma 1, there are 2^s permutations by defining a new Hadamard from one of the row. Therefore the total number of permutations is

$$\{\#\text{ of equivalence classes}\} \left(\prod_{i=0}^{s-1} (2^s - 2^i)\right) (2^s)\,.$$

Equating these two expressions together, we get

$$\{\# \text{ of equivalence classes}\} = \frac{(2^s-1)!}{\prod_{i=0}^{s-1}(2^s-2^i)}. \qquad \square$$

D.7 Proof of Lemma 4

Proof. We first show that a $\mathcal{H}$-permutation of the first row of a Hadamard-Cauchy matrix is equivalent to choosing a different set of basis. Let $K = hc(\frac{1}{z}, \frac{1}{z\oplus x_1}, \frac{1}{z\oplus x_2}, \frac{1}{z\oplus x_3}, ..., \frac{1}{z\oplus x_{2^s-1}})$ be an involutory Hadamard-Cauchy matrix. Under the type 1 of $\mathcal{H}$-permutation, for some $\alpha \in \{1, ..., 2^s-1\}$, we have $K' = hc(\frac{1}{z\oplus x_\alpha}, \frac{1}{z\oplus x_1\oplus x_\alpha}, \frac{1}{z\oplus x_2\oplus x_\alpha}, \frac{1}{z\oplus x_3\oplus x_\alpha}, ..., \frac{1}{z\oplus x_{2^s-1}\oplus x_\alpha})$. From this, we can see that $z' = z \oplus x_\alpha$ while the first s elements $\{x_{2^j}\}, \forall j = 0, 1, ..., s-1$, remain unchanged. Since z' is not a linear combination of the s elements, we have our $s+1$ linearly independent elements. Under the type 2 of $\mathcal{H}$-permutation, since $\sigma(0) = 0$, the last element z remain unchanged. Therefore, it is a linear permutation (w.r.t. XOR) on the set S and the new s elements $\{x'_{2^j}\}, \forall j = 0, 1, ..., s-1$ are still linearly independent. Again, we have our $s+1$ linearly independent elements. Finally, as mentioned before in Lemma 1 and 3, there are $2^s \cdot \prod_{i=0}^{s-1}(2^s-2^i)$ ways of $\mathcal{H}$-permutations. $\square$

D.8 Proof of Lemma 5

Proof. As we can see from Algorithm 1, we need to choose $s+1$ many linearly independent ordered elements from GF(2^r) to construct a Hadamard-Cauchy matrix. For the $(t+1)$-th element, where $0 \le t \le s$, it cannot be a linear combination of the t previously chosen elements, hence there are $2^r - 2^t$ many choices. Therefore, there are $(2^r-1)(2^r-2)(2^r-4)...(2^r-2^s)$ ways to choose $s+1$ linearly independent ordered elements. $\square$

D.9 Proof of Theorem 4

Proof. Again, we use the double counting to prove this theorem. We count the total number of distinct Hadamard-Cauchy matrices that can be generated from Algorithm 1.
Counting 1: by Lemma 5, the total number of distinct Hadamard-Cauchy matrices generated from the algorithm is $\prod_{i=0}^{s}(2^r-2^i)$.
Counting 2: for each of the equivalence classes of involutory Hadamard-Cauchy matrices, by Lemma 4, there are $2^s \cdot \prod_{i=0}^{s-1}(2^s-2^i)$ involutory Hadamard-Cauchy matrices that are related. Moreover, for each of the involutory Hadamard-Cauchy matrices, we can multiply by a nonzero scalar to obtain another related Hadamard-Cauchy matrix, thus there are another factor 2^r-1 of distinct Hadamard-Cauchy

matrices. Therefore, the total number of distinct Hadamard-Cauchy matrices is

$$\{\# \text{ of equivalence classes}\} \left(2^s \cdot \prod_{i=0}^{s-1}(2^s - 2^i)\right)(2^r - 1) \, .$$

Equating these two expressions together, we get

$$\{\# \text{ of equivalence classes}\} = \prod_{i=0}^{s-1} \frac{2^{r-1} - 2^i}{2^s - 2^i} . \qquad \square$$

D.10 Proof of Corollary 1

Proof. We count the number of distinct compact Cauchy matrices that can be generated from one equivalence class. Taking the first row of an equivalence class of involutory Hadamard-Cauchy matrices, we can multiply it by a nonzero scalar. The inverse of these entries corresponds to a set of 2^s nonzero elements. Each of these elements can be defined to be z and we have a set S and $z \in \mathrm{GF}(2^r) \setminus S$. Note that S is closed under XOR operation and in the context of [15], we can regard S as a subgroup of $\mathrm{GF}(2^r)$ defined under XOR operation. Finally, by fixing the first element of S and $z+S$ to be 0 and z repectively, we have $(2^s-1)!$ permutation for each set S and $z + S$. Each arrangement generates a distinct compact Cauchy matrix. Therefore, considering all equivalence classes, we can obtain

$$\left(\prod_{i=0}^{s-1} \frac{2^{r-1} - 2^i}{2^s - 2^i}\right)(2^r - 1)(2^s)\left((2^s - 1)!\right)^2$$

distinct compact Cauchy matrices, which coherent to Theorem 3 of [15]. □

D.11 Proof of Lemma 6

Proof. Suppose there exists a $l \times l$ singular submatrix, by the first statement of Proposition 3, M is not MDS and thus from the second statement, there exists k columns of $[I_k|M]$ that are linearly dependent, in particular, $k-l$ columns from I_k and l columns from M. Let L be the square matrix comprising these k linearly dependent columns. From linear algebra, there exists nonzero vector, v, such that the output is a zero vector, $vL = 0$. For the columns from I_k, there is exactly one 1 and 0 for the other entries, this implies that the components of v corresponding to these columns are zero, else the output will be nonzero. Therefore, there are at most l nonzero components in v. Now, let us consider $vM = u$, for the l columns of M that are also in L, the corresponding output components are zero, as $vL = 0$. Thus, there are at most $k - l$ nonzero components in u. Hence, the sum of nonzero components in v and u is at most k. The converse is similar, we consider $v[I_k|M] = [v|u]$, since there are at most k nonzero components in $[v|u]$, we pick $k-l$ columns of I_k and k columns of M corresponding to the zero components in $[v|u]$ to form a singular square matrix L. The determinant of L is equal to some $l \times l$ submatrix of M, which is also singular. □

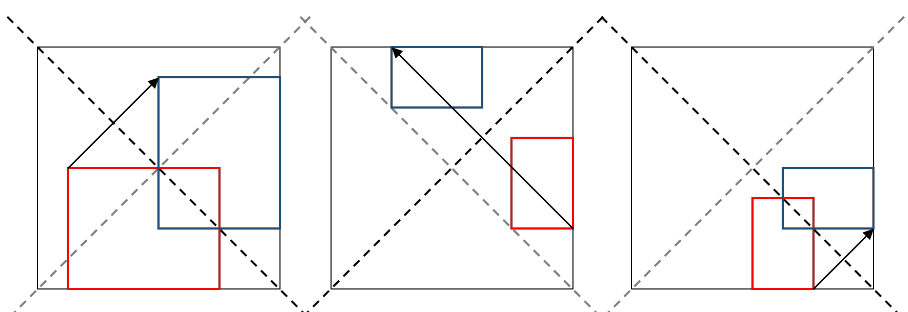
D.12 Proof of Proposition 4

Proof. Suppose not, there exists submatrix of order $l \geq \frac{k}{2} + 1$ that is singular. By Lemma 6, there exists a vector, $v \neq 0$, with at most l nonzero components such that $vH = u$ and u has at most $k - l$ nonzero components. Right multiply H to the equation and we get $c^2 v = uH$, where $c \neq 0$, hence the number of nonzero component of $c^2 v$ is the same as v. However, since u has $k - l \leq \frac{k}{2}$ nonzero components, by Corollary 4, the sum of nonzero components is at least $k + 1$. This contradicts that v has at most l nonzero components. □

D.13 Proof of Proposition 5

Proof. It is known that the determinant of a matrix remains unchanged when it undergoes left or right diagonal reflection. Thus, it is sufficient to show that for any submatrix, it corresponds to some submatrix with the leading entry in top quadrant. This can be shown by looking at the reflection through the left and/or right diagonal. Consider the submatrices case by case:

- case A: the leading entry is in left quadrant. Through the left diagonal reflection, we can see that it is same as a submatrix with leading entry in top quadrant. Refer to Figure 2a, the red submatrix is reflected at the blue matrix with leading entry in top quadrant.
- case B1: the leading entry is not in left quadrant and ending entry is in right quadrant. Through the right diagonal reflection, the ending entry $L[l-1, l-1]$ of red submatrix is reflected to the leading entry in the top quadrant of the blue submatrix, see Figure 2b. Since the determinant does not change, the red submatrix will be nonsingular if the blue matrix is nonsingular.
- case B2: the leading entry is not in left quadrant and ending entry is in bottom quadrant. From Figure 2c, we see that through left diagonal reflection, the ending entry is now in the right quadrant, which is the case B1. □



(a) Case A. (b) Case B1. (c) Case B2.

Fig. 2: Submatrices Reflections

D.14 Proof of Lemma 7

Proof. Since we are taking the span of the ordered set $\{x_1, x_2, x_{2^2}, ..., x_{2^{s-1}}\}$ and adding z to the span, it is obvious that permuting $\{x_{2^i}\}$ will only permute the order of the entries of K. □

D.15 Proof of Proposition 6

Proof. Any linearly independent ordered set of elements $\{x_1, x_2, x_{2^2}, ..., x_{2^{s-1}}\}$ that are not in ascending order is simply some permutation of a set in ascending order. By Lemma 7, it forms a Hadamard-Cauchy matrix of the same equivalence class. □

E Primitive mapping between finite fields

Table 2: Primitive mapping from $GF(2^4)$/0x13 to $GF(2^4)$/0x19

order	0x13 (10011)		0x19 (11001)	
	x	XOR	x	XOR
α	2	1	12	1
α^2	4	2	6	2
α^3	8	3	3	3
α^4	3	5	13	5
α^5	6	5	10	5
α^6	12	5	5	5
α^7	11	6	14	6

order	0x13 (10011)		0x19 (11001)	
	x	XOR	x	XOR
α^8	5	6	7	6
α^9	10	8	15	8
α^{10}	7	9	11	9
α^{11}	14	8	9	8
α^{12}	15	6	8	6
α^{13}	13	3	4	3
α^{14}	9	1	2	1

Table 3: Primitive mapping from finite field to its reciprocal finite field

finite field	$p(X)$	$\frac{1}{p}(X)$	**primitive mapping**
$GF(2^2)$	0x7	-	$\phi : 2 \mapsto 3$
$GF(2^3)$	0xb	0xd	$\phi : 2 \mapsto 6$
$GF(2^4)$	0x13	0x19	$\phi : 2 \mapsto 12$
	0x1f	-	$\phi : 3 \mapsto 5$
$GF(2^5)$	0x25	0x29	$\phi : 2 \mapsto 20$
	0x3d	0x2f	$\phi : 2 \mapsto 23$
	0x37	0x3b	$\phi : 2 \mapsto 29$
$GF(2^6)$	0x43	0x61	$\phi : 2 \mapsto 48$
	0x57	0x75	$\phi : 3 \mapsto 59$
	0x67	0x73	$\phi : 2 \mapsto 57$
	0x49	-	$\phi : 3 \mapsto 37$
$GF(2^7)$	0x83	0xc1	$\phi : 2 \mapsto 96$
	0xab	0xd5	$\phi : 2 \mapsto 106$
	0x8f	0xf1	$\phi : 2 \mapsto 120$
	0xfd	0xbf	$\phi : 2 \mapsto 95$
	0xb9	0x9d	$\phi : 2 \mapsto 78$
	0x89	0x91	$\phi : 2 \mapsto 72$
	0xe5	0xa7	$\phi : 2 \mapsto 83$
	0xef	0xf7	$\phi : 2 \mapsto 123$
	0xcb	0xd3	$\phi : 2 \mapsto 105$

finite field	$p(X)$	$\frac{1}{p}(X)$	**primitive mapping**
$GF(2^8)$	0x11d	0x171	$\phi : 2 \mapsto 184$
	0x177	0x1dd	$\phi : 3 \mapsto 239$
	0x1f3	0x19f	$\phi : 6 \mapsto 103$
	0x169	0x12d	$\phi : 2 \mapsto 150$
	0x1bd	0x17b	$\phi : 7 \mapsto 95$
	0x1e7	0x1cf	$\phi : 2 \mapsto 231$
	0x12b	0x1a9	$\phi : 2 \mapsto 212$
	0x1d7	-	$\phi : 7 \mapsto 116$
	0x165	0x14d	$\phi : 2 \mapsto 166$
	0x18b	0x1a3	$\phi : 6 \mapsto 104$
	0x163	0x18d	$\phi : 2 \mapsto 198$
	0x11b	0x1b1	$\phi : 3 \mapsto 217$
	0x13f	0x1f9	$\phi : 3 \mapsto 253$
	0x15f	0x1f5	$\phi : 2 \mapsto 250$
	0x1c3	0x187	$\phi : 2 \mapsto 195$
	0x139	-	$\phi : 3 \mapsto 157$

F Tables of XOR count

Table 4: XOR count for $GF(2^2)$, $GF(2^3)$ and $GF(2^4)$

x	$GF(2^2)$	$GF(2^3)$		$GF(2^4)$		
	0x7	0xb	0xd	0x13	0x19	0x1f
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	1	1	1	1	1	3
3	1	4	2	5	3	5
4	–	2	3	2	3	3
5	–	1	4	6	5	5
6	–	4	1	5	2	6
7	–	3	4	9	6	6
8	–	–	–	3	6	3
9	–	–	–	1	8	5
10	–	–	–	8	5	6
11	–	–	–	6	9	6
12	–	–	–	5	1	6
13	–	–	–	3	5	6
14	–	–	–	8	6	5
15	–	–	–	6	8	3
mean		1.88	1.88	4.25	4.25	4.25
σ		1.4569	1.4569	2.6800	2.6800	1.7075