FOAM: Searching for Hardware-Optimal SPN Structures and Components with a Fair Comparison

Khoongming Khoo[1], Thomas Peyrin[2], Axel Y. Poschmann[3], and Huihui Yap[1]

[1] DSO National Laboratories, 20 Science Park Drive, Singapore 118230.
{kkhoongm,yhuihui}@dso.org.sg
[2] SPMS, Nanyang Technological University, Singapore.
thomas.peyrin@ntu.edu.sg
[3] NXP Semiconductors
axel.poschmann@gmail.com

Abstract. In this article, we propose a new comparison metric, the *figure of adversarial merit* (FOAM), which combines the inherent security provided by cryptographic structures and components with their implementation properties. To the best of our knowledge, this is the first such metric proposed to ensure a fairer comparison of cryptographic designs. We then apply this new metric to meaningful use cases by studying Substitution-Permutation Network permutations that are suited for hardware implementations, and we provide new results on hardware-friendly cryptographic building blocks. For practical reasons, we considered linear and differential attacks and we restricted ourselves to fully serial and round-based implementations. We explore several design strategies, from the geometry of the internal state to the size of the S-box, the field size of the diffusion layer or even the irreducible polynomial defining the finite field. We finally test all possible strategies to provide designers an exhaustive approach in building hardware-friendly cryptographic primitives (according to area or FOAM metrics), also introducing a model for predicting the hardware performance of round-based or serial-based implementations. In particular, we exhibit new diffusion matrices (circulant or serial) that are surprisingly more efficient than the current best known, such as the ones used in `AES`, `LED` and `PHOTON`.

Key words: SPN, lightweight cryptography, figure of adversarial merit, diffusion matrices.

1 Introduction

RFID is a rising technology that is likely to be widely deployed in many different situations of everyday life, leading to new security challenges that the cryptography community has to handle. Significant advances in this area have already been obtained. In particular, many lightweight block ciphers [10,12,17,22] have recently been proposed, and designing such ciphers is not an easy task as showed by the numerous candidates that eventually got broken. Moreover, it is interesting to note that in most privacy-preserving RFID protocols proposed [3,18,19] a hash function is required, and since a hash function can be easily built from a block cipher (for example with the Davies-Meyer mode) or a permutation (for example with the sponge construction [9]), a crucial question for the researchers is how to design a hardware efficient permutation (that can later be utilized to build a hash function and/or a block cipher).

Hardware efficiency can have very different meanings depending on the utilization scenario targeted by the designer. For example, a classical metric is to estimate the minimum silicon area required by the primitive to perform the cryptographic operations. This, of course, depends on the parameters of the function itself (the area is highly dependent on the amount of memory required) and most lightweight block ciphers have a rather small block size of 64 bits. It is to be noted that the area is usually not directly linked to the security of a primitive, as adding extra rounds will have an impact on the throughput of the implementation, but only a very limited one concerning the area (we assumed that the function has no weakness that is independent of the number of rounds). Area and other metrics such as throughput, latency or power dissipation can be traded-off for one another, making the comparison between different primitives difficult. In the direction of fairer comparisons of hardware implementations of cryptographic primitives, Bogdanov *et al.* [11] introduced the efficiency metric throughput/area in order to take in account these tradeoffs. However, the possibility of trading off throughput for power was not taken in account and Badel *et al.* [4] proposed instead a *figure of merit*, defined as FOM = throughput/area^2.

However, as of today, no metric takes in account the inherent security of a building block, therefore making it hard to compare for example two diffusion matrices that have different area footprint and different branching number.

The construction of good diffusion matrices has always been an important research topic in cryptography, equally important as the search for good confusion functions. The `AES` [15] for example uses a 4×4 matrix with elements in $GF(2^8)$. This matrix is Maximum Distance Separable (MDS), which means that it has a branching number of 5, optimal for a 4×4 matrix. However, this security feature comes at a cost that computations in $GF(2^8)$ might not be the best choice for some hardware purposes, even though special care has been taken by the designers to choose a circulant matrix instantiated with lightweight coefficient (i.e. low Hamming weight coefficients, such as 0x01, 0x02 and 0x03). Recently, Guo *et al.* [16,17] described a new type of diffusion matrix, so-called serial, that trades more clock cycles in the execution for a smaller area. This idea was later extended to the use of linear Feistel-like structures or Linear Feedback Shift Registers (LFSR) to build the diffusion matrix [21,23]. On the opposite side, `PRESENT` [10] uses a simple bit permutation layer, the real diffusion coming in fact directly from the S-box application. The advantage being of course that a bit permutation layer is basically free in a hardware implementation. Now, one may ask the following question: what is better when the goal is to maximize some hardware metric, a very weak diffusion matrix with a low area footprint, or a strong diffusion matrix but requiring more silicon to be performed?

More generally, many different trade-offs exist when building an `AES`-like Substitution-Permutation Network (SPN) primitive, such as the general geometry (number of lines and columns), what size of S-box, what type of matrix, with what branching number, in what finite field, with which irreducible polynomial, etc. When a cryptographer would like to design a permutation with a specific hardware efficiency metric in mind, it is not trivial for him to make the best construction choices directly. Since implementing many different trade-offs is very time consuming, he will have to rely on his own intuition when picking the basic building blocks and choosing the general structure of the primitive, therefore accepting that his final design might not be optimal.

Our contributions. In this article, we study the problem of designing hardware efficient permutations for lightweight symmetric key cryptography purposes, and we propose new promising diffusion matrices as building blocks. We first explain in Section 2 the family of functions that we will study, namely `AES`-like SPN permutations, and we describe a new generalized diffusion layer (i.e. the `ShiftRows` function in `AES`), that allows a provable optimal diffusion even for non-square internal state matrices. Then, we introduce in Section 3 a new metric, the *figure of adversarial merit* (FOAM), that for the first time takes into account the inherent security provided by the primitive. We then explain in Section 4 the various SPN design tradeoffs that we will consider for our comparisons, such as the geometry of the SPN, the S-box size, the type of matrix (circulant or serial), the field size for the diffusion or even the irreducible polynomial. The goal being that the designer only has to input the type of implementation (round and/or serial) and the size of the permutation he would like to build, and he can directly get the SPN structure and its internal components that are the best suited for him. We study in Section 5 the security of the `AES`-like SPN permutations by only taking in account simple linear/differential attacks. We then describe how one can estimate the hardware implementations efficiency in Section 6 and in particular the non trivial task of estimating the area consumption induced by the control logic. Moreover, we show that in some situations, a coefficient rewrapping trick can be used to significantly improve the efficiency of a diffusion matrix. We chose to focus our work on designing permutations only since many cryptographic primitives can be built from them. Therefore, we will not cover other components such as key schedule for a block cipher, or message expansion for a hash function. Moreover, due to the obviously vast amount of implementation trade-offs, we restricted ourselves to the two most important cases: fully serialized and round-based.

Finally, the results obtained by our analysis are given in Section 7, with the best diffusion matrices and SPN parameters we could find for many different scenarios. Notably, we show that the diffusion matrices of ciphers such as `AES`, `LED` or `PHOTON` are not the best possible choices. For example, in the case of `AES` encryption, a circulant matrix with coefficients (0x01,0x01,0x04,0x8d) would have been, surprisingly, a better choice in terms of implementation while keeping the same Maximum Distance Separable (MDS) security.

2 Generic SPN with generalized optimal diffusion

In this section, we describe the family of AES-like SPN functions we are considering. Our scope is quite classical, but we propose a new generalized diffusion layer (i.e. the ShiftRows function in AES), that allows an optimal diffusion even for non-square internal state matrices.

2.1 Extended AES-like permutations

An n-bit AES-like SPN permutation transforms an $r \times c$ array of s-bit cells ($n = r \times c \times s$). During one round, each cell is first transformed by an s-bit S-box (similar to the AES SubBytes operation). Then each r-cell column is transformed by an $r \times r$ diffusion matrix (similar to the AES MixColumns operation), followed by an optimal diffusion which permutes the c cells of each row to provide further mixing (similar to the AES ShiftRows operation). Finally, an $(r \times c)$-cell constant is XORred to complete a round transformation (in a block-cipher design, this phase is a subkey addition, but we will not consider key-schedules in this article). Note that in AES, we have a square array $r = c = 4$ and cell size $s = 8$-bit. The diffusion matrix is usually defined over the finite field $GF(2^s)$ because of the s-bit cell size. Sometimes, we might actually use a smaller subfield of size $GF(2^i)$, i divides s, in order to define the diffusion matrix. This framework captures many known ciphers such as AES, PRESENT, LED, etc.

In this paper, a cell is called differentially (resp. linearly) active if its value (resp. mask value) is non-zero in a differential (resp. linear) attack. The differential branch number of a diffusion matrix is the minimum number of differentially active input and output cells (among all non-zero inputs). The notion of linear branch number is similar, except that we consider the transpose of the diffusion matrix instead. From this point onwards, we will not distinguish between differential and linear branch number unless necessary. That is, if it is stated that a matrix has branch number, say, 3, we mean that both the differential and linear branch number are 3. The maximum branch number for an r by r diffusion matrix is $r + 1$, and a matrix which achieves this optimal branch number is called Maximum Distance Separable (MDS). If the diffusion matrix has branch number r, then it is called almost-MDS.

2.2 The generalized optimal diffusion

In this section, we generalize the concept of optimal diffusion [15] for non-square state array. This has been done already when $r < c$ with a security bound equivalent to the case where $r = c$ (square array) [15]. When $r > c$ and c divides r, a simple generalization has been proposed in [13] where a 4-round security bound is proven when the diffusion matrix is MDS. In this section, we propose a generalized optimal diffusion for the case $r > c$ where c may not divide r and the diffusion matrix may not be MDS, i.e. for all branch number $B \leq r + 1$.

An example of optimal diffusion is the ShiftRows operation of AES which helps to diffuse the effect of the AES SubBytes and MixColumns operation over 32-bit to the whole 128-bit block. The AES ShiftRows transforms a 4×4 byte-array by rotating row r to the left by r bytes, for $r = 0, 1, 2, 3$. The effect of ShiftRows is that each byte of an input column is mapped to a different output column. This is captured by the concept of optimal diffusion (another example is the ArrayTranspose map of the SQUARE cipher [14]).

Definition 1. *For an r-by-r cell-array, the optimal diffusion map is a cell-permutation that maps each cell of an input column to a different output column.*

However, the optimal diffusion only applies for $r \times c$ cell array where $r \leq c$. When $r > c$, there are not enough output columns c to map each of the r cells of an input column. Thus, we extend a new concept from [13] called Generalized Optimal Diffusion (GOD) for $r \times c$ cell-array when $r > c$, which we describe below. Our strategy is to distribute the cells of an input column as uniformly as possible to each output column.

Note that here, without loss of generality, we apply the permutation operations from right-to-left, i.e. SC (SubCells) is first applied, followed by MC (MixColumn) and then the optimal diffusion.

The Generalized Optimal Diffusion (GOD) defined in [13] applies only when r is a multiple c. Here, we deinfe GOD for any $r > c$.

Definition 2. *For an $r \times c$ cell-array, a generalized optimal diffusion is a cell-permutation such that looking at any r-cell column:*

1. *$\lceil r/c \rceil$ input cells are mapped to each of $(r \mod c)$ output columns.*
2. *$\lfloor r/c \rfloor$ input cells are mapped to each of $c - (r \mod c)$ output columns.*

Example 1. Consider $r = 5$, $c = 3$. For each input column of 5 cells, $\lceil 5/3 \rceil = 2$ input cells are mapped to each of $(5 \mod 3) = 2$ columns. $\lfloor 5/3 \rfloor = 1$ input cell is mapped to $3 - (5 \mod 3) = 1$ column. One example is given by the transform of the following arrays:

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & c_4 \\ a_5 & b_5 & c_5 \end{pmatrix} \text{ maps to } \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ c_3 & a_3 & b_3 \\ c_4 & a_4 & b_4 \\ b_5 & c_5 & a_5 \end{pmatrix}$$

Theorem 1. *Consider a 4-round* `AES`*-like SPN as follows (omitting the constant addition since it has no effect on our reasoning):*

$$\texttt{GOD} \circ \texttt{MC} \circ \texttt{SC} \circ \texttt{GOD} \circ \texttt{MC} \circ \texttt{SC} \circ \texttt{GOD} \circ \texttt{MC} \circ \texttt{SC} \circ \texttt{GOD} \circ \texttt{MC} \circ \texttt{SC},$$

where

1. `SubCells` *is a nonlinear substitution layer with $r \times c$ s-bit S-boxes acting in parallel.*
2. `MixColumns` *is a layer of c parallel* `MixColumn` *transforms each mapping r cells to r cells with branch number B, i.e.* $\texttt{MixColumns}(\texttt{x}_1, \ldots, \texttt{x}_\texttt{c}) = (\texttt{MixColumn}(\texttt{x}_1), \ldots, \texttt{MixColumn}(\texttt{x}_\texttt{c}))$*, each x_i corresponding to a column of r cells.*
3. `GOD` *(generalized optimal diffusion) is as defined above which distributes the r cells of an input column almost uniformly to c output columns.*

Then the number of active S-boxes over 1 and 2 rounds are at least 1 and B respectively. For 4 rounds it is at least $B \times B'$ where $B' = \max\{2; x + y\}$ and:

$$\begin{cases} y = \min\{2 \times (r \bmod c)); \lfloor B/\lceil r/c \rceil \rfloor\} \\ x = \quad \lceil (B - \lceil r/c \rceil \times y)/\lfloor r/c \rfloor \rceil \end{cases}$$

We provide the proof of this theorem in Appendix A.1. We note that it is tight in the sense that it naturally provides a 4 round path that corresponds to a "luckiest" scenario for the attacker, which involves the minimum number of active Super-Sboxes (the $(c \times s)$-bit S-boxes composed of two `SubCells` layers surrounding one `MixColumns`).

Let us look at an application example of Theorem 1 to derive the number of active S-boxes of an `AES`-like SPN structure, which cannot be deduced by the known results of [13,15]. Consider an SPN structure with state size 24-cell, the diffusion matrices being an 8×8 matrix with branch number 7, i.e. $r = 8$, $c = 3$ and $B = 7$. By Theorem 1, we have $y = 2$ and $x = 1$, therefore $B' = \max\{2; x + y\} = 3$ and there are $B \times B' = 7 \times 3 = 21$ active S-boxes guaranteed over 4 rounds of this 24-cell SPN structure.

As a corollary, we explore the cases when the formula for the number of active S-boxes over 4 rounds can be simplified (the proof is given in Appendix A.2).

Corollary 1. *In Theorem 1, we have the following special cases:*

1. *If $c > r$, then the number of active S-boxes over four rounds is at least B^2 (known result from [15]).*
2. *If c divides r and $B = r + 1$, i.e.* `MixColumn` *is MDS, then the number of active S-boxes over four rounds is at least $(r + 1) \times (c + 1)$ (known result from [13]).*
3. *If c divides r and $B = r$, i.e.* `MixColumn` *is almost-MDS, then the number of active S-boxes over four rounds is at least $r \times c$.*

3 FOAM: Figure Of Adversarial Merit

As explained in the introduction, the various trade-offs inherent in any design of a cryptographic primitive make a fair and consistent comparison of software and hardware implementations thereof a challenging task. For hardware implementations exist a few metrics, like the *Area-Time (AT) product*, which multiplies the area in *Gate Equivalents* (GE) occupied by the design with the number of clock cycles required (the smaller the number, the more efficient is the design). Closely related is the *hardware efficiency* [11], which divides the throughput at a given frequency by the area (hence the greater the number, the better the design). In order to also address the area-power trade-off, [4] proposed a new *Figure of Merit (FOM)*: throughput divided by the area squared. The latter two metrics are frequency dependent, which can complicate comparisons.

We propose a new metric called Figure of Adversarial Merit (FOAM) in order to resolve the aforementioned shortcomings. It is defined as

$$FOAM(x) = \frac{1}{S(x) \times A^2}$$

where $S(x)$ and A are basically equivalent to special definitions of speed and area, respectively. More precisely, $S(x)$ denotes the speed of the cipher based on the number of rounds required to achieve a certain security x against some set of attacks (in this article, we will later restrict ourselves to simple differential/linear attacks). For a round-based permutation, it is defined as $S(x) = p(x) \times t$ where $p(x)$ represents the number of rounds required to achieve security x, and t the number of clock cycles to perform one round. Moreover, for SPN-based primitives, we decompose the area requirements A into six parts: the intermediate state memory cost C_{mem}, the S-boxes implementation cost C_{sbox}, the diffusion matrix implementation cost C_{diff}, the constant addition C_{cst}, the control logic cost C_{log}, and the IO logic cost C_{io}:

$$FOAM(x) = \frac{1}{S(x) \times A^2} = \frac{1}{p(x) \times t \times (C_{mem} + C_{sbox} + C_{diff} + C_{cst} + C_{log} + C_{io})^2}$$

This FOAM metric will be useful to compare different design strategies, different building blocks (such as diffusion matrices) with a simple value computation. Even better, we would like to roughly compare all these possible design trade-offs without having the hassle to implement all of them: in Section 6 we will detail how to estimate these six subparts of the area cost and the number t of clock cycles required to perform one round. The value $p(x)$ can be deduced by the number of active S-boxes proven in Theorem 1 and the S-box cryptographic properties (see Section 5). Note that in the rest of the paper, we consider that the security aimed by the designer is equal to the permutation size, i.e. we are aiming at a security of 2^n computations (thus $p(x) = p(2^n)$).

4 Trade-offs considered

In this section, we explain all the various trade-offs we will consider when building an `AES`-like SPN permutation. The goal being that a designer specifies a permutation bitsize n, the metric he would like to maximize (area, FOAM), the degree up to which serial or round-based implementations are important, and he directly obtains the best parameters to build his permutation.

The S-box. One of the first choice of the designer is the size of the S-box, and we will consider two possible trade-offs: $s = 4$ and $s = 8$. Note that, for simplicity, we will consider that the S-box chosen has perfect differential and linear properties relative to its size (one could further extend the trade-offs to non-optimal but smaller S-boxes, but the search space being very broad we leave this as an open problem).

The geometry of the internal state. When building an `AES`-like SPN permutation, one can consider several internal state geometries (the values r and c). The classical case is a square state, like for `AES`. However, depending on the diffusion matrices available, it might be worth considering more line-shaped or column-shaped designs.

Diffusion matrix field size. The designer can choose the field size 2^i in which the matrix computations will take place. The classical case, like in `AES`, being that the field size for the diffusion matrix is the same as the S-box. However, depending on the diffusion matrices available, it might be worth considering designs with thinner diffusion layers but repeated several times. For example, in the case of `AES`, instead of the `MixColumns` matrix one could use a 4×4 diffusion matrix on $GF(2^4)$ applied two times (one time on the 4 MSB and one time on the 4 LSB of the 8-bit cells in the `AES` column). Overall, we will cover a scope from binary matrices (in GF(2)) up to matrices on the same field size as the S-box (in $GF(2^s)$).

Irreducible polynomial for the diffusion matrix field. Once the field size 2^i is fixed, the designer can choose the irreducible polynomial defining the field. For $i = 1$ and $i = 2$ only a single polynomial exists, while for $i = 4$ at most 3 choices are possible ($\alpha^4 + \alpha + 1$, $\alpha^4 + \alpha^3 + 1$ and $\alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1$). For the $i = 8$ case, many polynomials are possible (this was already observed by [5]), thus in order to focus the search space we will only consider the irreducible polynomial used in `AES` ($\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1$) and in `WHIRLPOOL` hash function [7] ($\alpha^8 + \alpha^4 + \alpha^3 + \alpha^2 + 1$).

Type of diffusion matrix. The designer can choose what type of matrix he will implement, the two main hardware-friendly types being circulant or serial. In the circulant case, the designer picks r coefficients $Z = (Z_0, \ldots, Z_{r-1})$ and the matrix Z is defined as

$$\begin{pmatrix} Z_0 & Z_1 & Z_2 & . & . & . & Z_{r-2} & Z_{r-1} \\ Z_{r-1} & Z_0 & Z_1 & . & . & . & Z_{r-3} & Z_{r-2} \\ Z_{r-2} & Z_{r-1} & Z_0 & . & . & . & Z_{r-4} & Z_{r-3} \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ Z_1 & Z_2 & Z_3 & . & . & . & Z_{r-1} & Z_0 \end{pmatrix}$$

In the serial case, the designer picks r coefficients $Z = (Z_0, \ldots, Z_{r-1})$ and the matrix Z is defined as

$$\begin{pmatrix} 0 & 1 & 0 & 0 & . & . & 0 & 0 \\ 0 & 0 & 1 & 0 & . & . & 0 & 0 \\ 0 & 0 & 0 & 1 & . & . & 0 & 0 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & . & 0 & 1 \\ Z_0 & Z_1 & Z_2 & . & . & . & Z_{r-2} & Z_{r-1} \end{pmatrix}^r$$

The matrix therefore takes r operations to be computed.

Branching number of the diffusion matrix. In general, implementing a matrix with very good diffusion property will cost more area and/or cycles than a weak one. For example, the `AES` matrix has ideal MDS diffusion property, but certainly requires more area to implement than a simple binary matrix with weaker properties. Since the former is bigger but stronger and the latter is smaller and weaker, it is not clear which alternative will lead to the best FOAM. Therefore, the designer can choose between a wide range of possibilities concerning the branching number B of the diffusion matrix, from $B = 3$ to $B = r + 1$ (MDS).

5 Security assessment of `AES`-like primitives

The FOAM metric takes into account the security of the permutation with regards to simple differential/linear attacks. We would like to evaluate this security for the `AES`-like SPN permutations we are considering. Theorem 1 gives us the minimal number of active S-boxes for a given number of rounds,

We note that the number of active S-boxes given by Theorem 1 is tight if the number of rounds is not equal to 3 modulo 4 (even in that case the theorem gives a very close estimation). This does not mean that the maximum differential and linear characteristic probabilities computed are tight, since it is unknown how many active S-boxes can use the maximum differential and linear characteristic probabilities at the same time (this remains an open problem).

and knowing the S-box cryptographic properties we can compute the maximum differential and linear characteristic probabilities of our generic SPN ciphers easily. In other words, we can easily compute the number of rounds $p(x) = p(2^n)$ required to achieve the aimed security 2^n.

As stated before, for simplicity, in the rest of this article we will consider that the S-boxes have perfect differential and linear properties: for a 4-bit S-box the maximum differential and linear characteristic probabilities are 2^{-2} (e.g. PRESENT S-box), while for a 8-bit S-box the maximum differential and linear characteristic probabilities are 2^{-6} (e.g. AES S-box). Of course, one can extend the trade-off by considering other S-boxes, that might require a smaller area, but will have worse security properties.

We reuse the example from Section 2.2 (i.e. SPN structure with state size 24-cell, the diffusion matrix being a 8×8 matrix with branch number 7) as an illustration. Previously, we know from Theorem 1 that there are at least 21 active S-boxes over 4 rounds of this SPN permutation. Suppose that 8-bit S-boxes (i.e. $s = 8$) having maximum differential and linear probabilities 2^{-6} are used. Then the maximum differential and linear characteristic probabilities over four rounds are upper-bounded by $(2^{-6})^{21} = 2^{-126}$.

We are aware that other attacks rather than simple differential/linear might exist, some perhaps much more complex and powerful (it would be interesting to give some generic description of various attacks like impossible differential attack, boomerang attack, etc. only using the parameters of the `AES`-like SPN permutation). However, our goal here is not to fully specify a permutation, but it is to compare many trade-offs and design strategies that will lead to good hardware performances. Therefore, we emphasize that the number of rounds $p(x)$ is of course not the number of rounds that should be chosen by a designer. This number should be carefully chosen after thorough cryptanalysis work on the entire primitive. Yet, we believe that this simple differential/linear criterion is a quite accurate way to compare the security of various `AES`-like SPN permutations.

6 Implementations in ASIC

In this section, we introduce some notation before we present formulas to estimate serialized and round-based implementations (we restricted ourselves to these two important practical cases due to the obviously vast amount of implementation trade-offs). Please note that all estimates have to be seen as *lower bounds*, as we use scan flip-flops, and consider neither reset nor I/O requirements, which can significantly impact the area count in practice. We argue that those requirements –though very important in practice– are highly application specific, and thus need to be determined on a case by case basis. In practice, a higher throughout can be achieved by using pipelining techniques to reduce the critical path at the cost of additional area. As this design goal is, again, highly application specific and FOAM is designed to be frequency independent, we have not considered it in our analysis.

We have estimated all serial architectures with the single optimization goal of minimal area in mind. In practice, some design decisions will most likely use another trade-off point more in favor of smaller time and larger area. To reflect this, we have estimated all round-based architectures optimized for maximum FOAM.

The table below provides an overview over the hardware building blocks we used, their notation and typical area requirements for a UMC 180 nm technology. In addition, we denote i the exponent for the field $GF(2^i)$.

Notation	Description	GE
DFF	1-input flip-flop	4.67
SFF	2-input flip-flop	6
MUX	2-input multiplexer	2.33

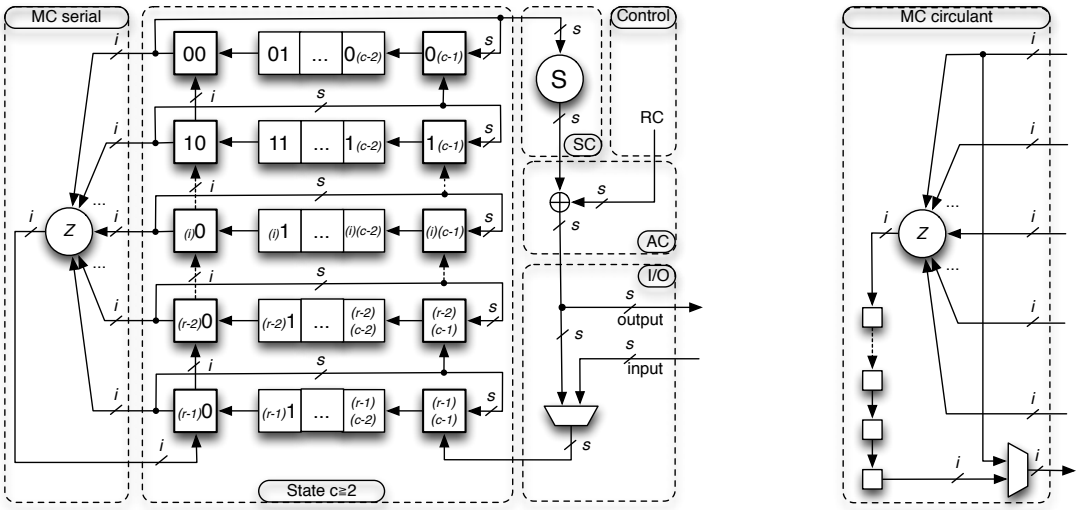
Notation	Description	GE
XOR	2-input exclusive Or	2.67
$SB4$	4 x 4 S-box (PRESENT)	22
$SB8$	8 x 8 S-box (AES)	233

In this section, we give more details about the ASIC implementation estimates given in Section 6. A general serialized architectures using a serial diffusion matrix is depicted in Figure 1a, while Figure 1b shows the circulant diffusion matrix module. In the special case of $c = 1$ the state module can be simplified to one of the two the architectures shown in Figure 1c, dependent whether $i \neq s$ (left) or $i = s$ (right). Finally, Figure 1d depicts a round-based architecture.

This is just *one example* for a technology and the area of the building blocks can be easily adapted for other technologies.

(a) Serialized using a serialized diffusion matrix, $c \geq 2$

(b) Circulant diffusion matrix



(c) State modules for $c = 1$, $i \neq s$ (left) and $i = s$ (right)

(d) Round-based

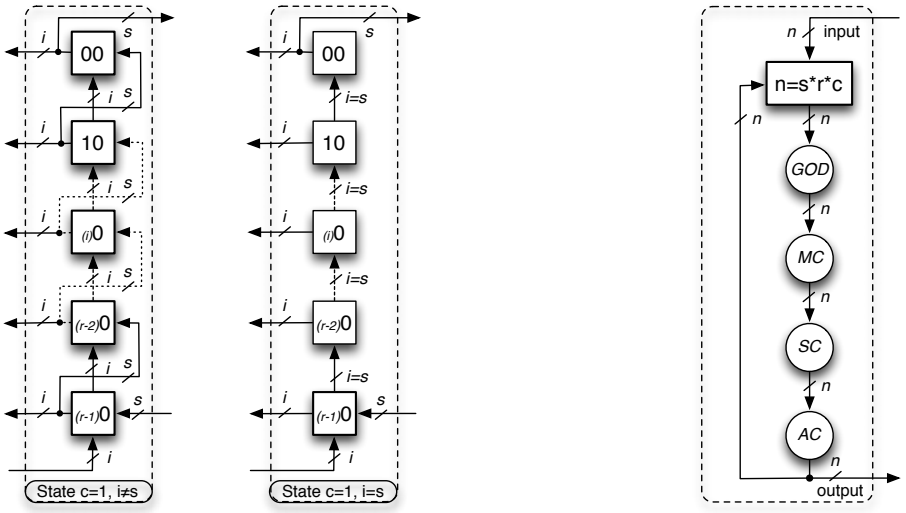


Fig. 1: Hardware architectures.

6.1 Serial architectures

Memory cost. The memory is arranged in an $r \times c$ array of s-bit cells. In case $c \geq 2$ the two outer columns need to be 2-input flip-flops SFF, while 1-input flip-flops DFF suffice for the remaining $c-2$ internal columns. In case $c = 1$ it depends whether $i \neq s$ or $i = s$. In the former case the whole state consists of 2-input flip-flops, while in the latter case it suffices to use 1-input flip-flops for the upper $r-1$ rows and only use 2-input flip-flops for the last row.

$$C_{mem} = \begin{cases} s \cdot (r - \lfloor \frac{i}{s} \rfloor) \cdot SFF + \lfloor \frac{i}{s} \rfloor s \cdot DFF \ , c = 1 \\ 2 \cdot s \cdot r \cdot SFF + s \cdot r \cdot (c-2) \cdot DFF \ , c \geq 2 \end{cases}$$

S-boxes cost. We use the S-boxes of AES and PRESENT to estimate the area for 8-bit and 4-bit S-boxes, respectively. In a UMC 180 nm technology they require $SB8 = 233$ GE and $SB4 = 22$ GE.

$$C_{sbox} = \begin{cases} SB4 \ , s = 4 \\ SB8 \ , s = 8 \end{cases}$$

Diffusion matrix cost. A denotes the numbers of XORs required to implement the last row of the serial or circulant matrix, and we provide an extensive list of good matrices and their XOR count in Section 7. In case a serial matrix is used A XORs is all that is required to implement the diffusion matrix.

Once one column has been computed, all columns are rotated by one column to the left and the next column can be processed. In case a circulant matrix is used, additional temporary storage of $s \times r - i$ 1-input flip-flops are required before the result is fed back to the leftmost column. Then all columns are rotated by one column to the left. One additional i-bit multiplexer is required.

$$C_{diff} = \begin{cases} A \cdot XOR \text{ , for serial mat.} \\ A \cdot XOR + (s \cdot r - i) \cdot DFF + i \cdot MUX \text{ , for circulant mat.} \end{cases}$$

Constant addition cost. To add a constant s XOR gates are required.

$$C_{cst} = s \cdot XOR$$

Control logic cost. A particular challenge is to estimate the control logic C_{log} required for a given architecture. Our estimates contain four parts: three counters a_r (rows), a_c (columns), and a_p (rounds); the finite state machine b; clock gating logic cg; and other combinational logic oc. The area for counters is mainly determined by the storage required for the minimal number of bits, some simple (e.g. NOT, NAND, NOR) feedback function and at least a 1-bit MUX. In total we estimate the area for any such counter to be $a_x = DFF \times \lceil log_2(x) \rceil + 5$, where x denotes either the number of rows, columns or rounds. The number of states is dependent on the geometry. In case $c \geq 2$ the serialized architecture we based our estimates on requires $c - 1$ states for `GOD`, two for `MixColumns`, and each one for `SubCells`, `IDLE`, and `INIT`; thus in total $c + 4$ states are required. The area for the finite state machine is estimated with $b_x = SFF \times \lceil log_2(c + 4) \rceil$. Based on post-synthesis figures for different variants of `PHOTON` we have derived the following formulas for clock gating (cg) and other combinational logic (oc): $cg = r \times 10 + 5$ and $oc = 8 \times r + 20$, respectively. In case $c = 1$ no clock gating logic, no column counter a_c, and only one state is required for `MixColumns`, thus the state machine estimates simplify to $b = SFF \times 2$. In total we get the following formula:

$$C_{log} = \begin{cases} a_r + a_p + SFF \cdot 2 + oc \text{ , } c = 1 \\ a_r + a_c + a_p + b + cg + oc \text{ , } c \geq 2 \end{cases}$$

Input/Output logic cost. For our serialized architecture we need only one additional s-bit multiplexer.

$$C_{io} = s \cdot MUX$$

Timing cost. Below are the formulas to estimate the time required to compute one round, dependent on the geometry and whether a serial or a circulant matrix is used:

$$t = \begin{cases} r \cdot c + (c - 1) + (\frac{s}{i} \cdot r + 1 - \lfloor \frac{i}{s} \rfloor) \cdot c \text{ , } c \geq 2 \text{ serial mat.} \\ r \cdot c + (c - 1) + (2 \cdot \frac{s}{i} \cdot r) \cdot c \text{ , } c \geq 2 \text{ circulant mat.} \\ r \cdot c + \frac{s}{i} \cdot r \text{ , } c = 1 \text{ serial mat.} \\ r \cdot c + (2 \cdot \frac{s}{i} \cdot r - 1) \text{ , } c = 1 \text{ circulant mat.} \end{cases}$$

For $c \geq 2$ the three summands represent the time required for: 1) `AddConstant` and `SubCells`; 2) `GOD`; 3) `MixColumns`. Please note that in case a serial matrix is used and $i = s$, it is possible to optimize the architecture in a way that for no extra hardware c clock cycles can be saved per round [1]. In case $c = 1$ there is no `GOD` and the first summand stays the same, while `MixColumns` does not require to rotate the columns regardless whether a circulant or serial diffusion matrix is used.

6.2 Round-based architectures

Memory cost. The $n = s \times r \times c$-bit state can be stored in 2-input flip-flops.

$$C_{mem} = s \cdot r \cdot c \cdot SFF$$

S-boxes cost. In total $r \times c$ S-boxes are required.

$$C_{sbox} = \begin{cases} r \cdot c \cdot SB4 \ , s = 4 \\ r \cdot c \cdot SB8 \ , s = 8 \end{cases}$$

Diffusion matrix cost. We need $r \times c \times \frac{s}{i}$ implementations of the last row of the matrix regardless whether a serial or circulant matrix is used.

$$C_{diff} = A \cdot r \cdot c \cdot \frac{s}{i} \cdot XOR$$

Constant addition cost. In total $s \times r \times c$ XORs are required to add all constants.

$$C_{cst} = s \cdot r \cdot c \cdot XOR$$

Control logic cost. In a round-based implementation basically only a round-counter a_p and -optionally- a very simple finite state machine is required. If we assume only three states IDLE, INIT, and ROUND, the area requirement for the finite state machine is $b = 2 \cdot SFF$.

$$C_{log} = a_p + b$$

Input/Output logic cost. One of the two inputs of the 2-input flip-flops used to store the state can be used for multiplexing the input. Hence, no additional logic is required

$$C_{io} = 0$$

Timing cost. In a round-based implementation one round is computed in one clock cycle.

$$t = 1$$

We now present the formulas for the estimates for the various parts of the ASIC implementations in Table 1.

6.3 Discussion

Table 2 compares FOAM of some real implementations of `LED` and `PHOTON` with our estimated FOAM. Please note that the authors of [17] and [16] did not use the special optimization trick, described above. To reflect this, we provide two FOAMs, one taking the optimization into account and one which does not. For `LED` [17] reports 966 GE for `LED`-64, where 299 GE are required to store the key state and around 40 GE are required for the key addition, multiplexers, and control logic. We thus compare with 627 GE. [20] reports 2,400 GE for `AES`-128 out of which 835 GE are required for the key schedule, thus we compare our FOAM to 1,565 GE. The authors chose a different optimization point, as can be seen in the higher area and significantly lower cycle count. We used the formulas above with the parameters of `PHOTON`-224 ($r = 8, c = 8, i = 4, s = 4, p = 8$, serialized MDS) for the estimation of the 256-bit permutation. As `PHOTON` is, contrary to `LED`, an unkeyed permutation, the last row of Table 2 is actually the best suited comparison. In summary, Table 2 underlines how close FOAM is to real implementations.

Please note that this is an upper bound for serial matrices, as Z may have coefficients that require less XORs than r times $(Z_0, \ldots, Z_{r-1})$. At the same time is the critical path of the serial matrix around r times longer than the one for a circulant matrix.

Note that it is not easy to compare to the implementation with a hard-wired key, as it is unclear how much area is required for the selection multiplexer.

Table 1: ASIC implementations estimation for the intermediate state memory cost C_{mem}, the S-boxes implementation cost C_{sbox}, the diffusion matrix implementation cost C_{diff}, the constant addition C_{cst}, the control logic cost C_{log}, the IO logic cost C_{io} and the number t of clock cycles to perform one round.

	Serial architectures	Round-based architectures
C_{mem}	$s \cdot (r - \lfloor \frac{i}{s} \rfloor) \cdot SFF + \lfloor \frac{i}{s} \rfloor s \cdot DFF$, $c = 1$ $2 \cdot s \cdot r \cdot SFF + s \cdot r \cdot (c-2) \cdot DFF$, $c \geq 2$	$s \cdot r \cdot c \cdot SFF$
C_{sbox}	$SB4$, $s = 4$ $SB8$, $s = 8$	$r \cdot c \cdot SB4$, $s = 4$ $r \cdot c \cdot SB8$, $s = 8$
C_{diff}	$A \cdot XOR$, for serial mat. $A \cdot XOR + (s \cdot r - i) \cdot DFF + i \cdot MUX$, for circulant mat.	$A \cdot r \cdot c \cdot \frac{s}{i} \cdot XOR$
C_{cst}	$s \cdot XOR$	$s \cdot r \cdot c \cdot XOR$
C_{log}	$a_r + a_p + SFF \cdot 2 + oc$, $c = 1$ $a_r + a_c + a_p + b + cg + oc$, $c \geq 2$	$a_p + b$
C_{io}	$s \cdot MUX$	0
t	$r \cdot c + (c-1) + (\frac{s}{i} \cdot r + 1 - \lfloor \frac{i}{s} \rfloor) \cdot c$, $c \geq 2$ serial mat. $r \cdot c + (c-1) + (2 \cdot \frac{s}{i} \cdot r) \cdot c$, $c \geq 2$ circulant mat. $r \cdot c + \frac{s}{i} \cdot r$, $c = 1$ serial mat. $r \cdot c + (2 \cdot \frac{s}{i} \cdot r - 1)$, $c = 1$ circulant mat.	1

7 Results and new diffusion matrices

In this section we provide the results of our framework, as well as new diffusion matrices that are very interesting for hardware implementations. As explained in Section 4, the designer's input is the permutation bitsize n, the metric he would like to maximize (area or FOAM), and the degree up to which serial or round-based implementations are important. To illustrate our method, we focused on the case where the designer would like to build a 64-bit permutation (which is a typical state size for a lightweight block cipher). For the implementation types, we focused on three scenarios: only serial implementation is important, only round-based implementation is important, serial and round-based implementations are equally important for the designer.

Before describing our results, we first explain how we found good diffusion matrices (circulant and serial) and then list these matrices in the next three subsections. Our optimal matrices outperform known ones from the `AES`, `LED` ciphers and the `PHOTON` hash function.

7.1 Lightweight coefficients

Consider the `AES` matrix, a circulant matrix with coefficients (0x01, 0x01, 0x02, 0x03) over $GF(2^8)$ defined by the irreducible polynomial $\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1$. The matrix appears to be very lightweight due to the low Hamming weight of its entries. But surprisingly, we found an even lighter circulant matrix over the same field with coefficients (0x01,0x01,0x04,0x8d). We now explain why this is so.

We first illustrate how to compute the number of XORs required to implement a multiplication by a finite field element x. To do so, we use $GF(2^8)$ defined by $\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1$ as an example. Let $x = x_7 \cdot \alpha^7 + x_6 \cdot \alpha^6 + \cdots x_1 \cdot \alpha + x_0 = (x_7, x_6, \cdots, x_1, x_0)$. Further, for ease of explanation, we employ hexadecimal encoding: $(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0)$ can be encoded as a tuple of hexadecimal numbers

We use the binary representation to represent finite field elements. E.g., 0x8d is 10001101 in binary, which corresponds to the finite field element $\alpha^7 + \alpha^3 + \alpha^2 + 1$ in $GF(2^8)$.

Table 2: Comparison of estimated FOAM with real implementations.

Bits	Permutation	p	t_s	Area	FOAM	Notes
64	Table 7 row 3	8	35	586	10.39	using optimization trick
			39	586	9.33	not optimized
	LED-64 [17]		39	627	8.09	Area /wo key schedule
			Difference	+7.0%	-13.3%	
256	PHOTON-224	8	135	1842	0.27	using optimization trick
			143	1842	0.26	not optimized
	PHOTON-224 [16]		143	1735	0.29	
			Difference	-5.6%	+12.7%	

(0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01). Then, the multiplication of 0x04 and 0x08 by x can be represented respectively as:

$$\begin{aligned}0\text{x}04 \cdot x =&(x_5, x_4, x_3 + x_7, x_2 + x_6 + x_7, x_1 + x_6, x_0 + x_7, x_6 + x_7, x_6)\\ =&(0\text{x}20,\ 0\text{x}10,\ 0\text{x}88,\ 0\text{xc}4,\ 0\text{x}42,\ 0\text{x}81,\ 0\text{xc}0,\ 0\text{x}40),\end{aligned}$$

$$\begin{aligned}0\text{x}08 \cdot x =&(x_4, x_3 + x_7, x_2 + x_6 + x_7, x_1 + x_5 + x_6, x_0 + x_5 + x_7, x_6 + x_7, x_5 + x_6, x_5)\\ =&(0\text{x}10,\ 0\text{x}88,\ 0\text{xc}4,\ 0\text{x}62,\ 0\text{xa}1,\ 0\text{xc}0,\ 0\text{x}60,\ 0\text{x}20).\end{aligned}$$

It can be seen that the number of XORs required for the multiplication of 0x04 and 0x08 by x is 6 and 9 respectively. Now we can compute

$$\begin{aligned}0\text{x}8d \cdot x &= (\alpha^7 + \alpha^3 + \alpha^2 + 1) \cdot x\\ &= (0\text{xb}1,\ 0\text{x}58,\ 0\text{x}2\text{c},\ 0\text{x}96,\ 0\text{xfa},\ 0\text{x}4\text{c},\ 0\text{xa}6,\ 0\text{x}62)\\ &\quad \oplus(0\text{x}10,\ 0\text{x}88,\ 0\text{xc}4,\ 0\text{x}62,\ 0\text{xa}1,\ 0\text{xc}0,\ 0\text{x}60,\ 0\text{x}20)\\ &\quad \oplus(0\text{x}20,\ 0\text{x}10,\ 0\text{x}88,\ 0\text{xc}4,\ 0\text{x}42,\ 0\text{x}81,\ 0\text{xc}0,\ 0\text{x}40)\\ &\quad \oplus(0\text{x}80,\ 0\text{x}40,\ 0\text{x}20,\ 0\text{x}10,\ 0\text{x}08,\ 0\text{x}04,\ 0\text{x}02,\ 0\text{x}01)\\ &= (0\text{x}01,\ 0\text{x}80,\ 0\text{x}40,\ 0\text{x}20,\ 0\text{x}11,\ 0\text{x}09,\ 0\text{x}04,\ 0\text{x}03)\\ &= (x_0, x_7, x_6, x_5, x_0 + x_4, x_0 + x_3, x_2, x_0 + x_1)\end{aligned}$$

Due to the '*cancellation of XORs*', we see that multiplication of x by 0x8d requires only 3 XORs. Similarly, the multiplication of x by 0x02 and 0x03 requires 3 and 11 XORs respectively.

In a similar fashion, for the purpose of finding lightweight diffusion matrices, we compute the XOR count for each field element. Table 3 and Table 4 shows the XOR count for every element of the finite fields $GF(2^4)$ and $GF(2^8)$ defined by different irreducible polynomials, respectively.

Now we explain how to use the tables to calculate the number of XORs required to implement a row of a matrix, as denoted by A in Section 6. Denote a given row of an $r \times r$ matrix by $(x_1, x_2, \cdots x_r)$ over a finite field $GF(2^i)$. Let γ_j be the XOR count in Table 3 and Table 4 ($i = 4$ and $i = 8$ respectively) corresponding to the field element x_j. Then A is equal to $(\gamma_1 + \cdots + \gamma_r) + (z-1) \cdot i$, where z is the number of non-zero elements in the row. We give some examples: row (0x1,0x1,0x4,0x9) uses $(0+0+2+1)+3\times 4$ = 15 XORs to implement over $GF(2^4)$; the AES matrix with coefficients (0x01,0x01,0x02,0x03) uses $(0 + 0 + 3 + 11) + 3 \times 8 = 38$ XORs to implement per row over $GF(2^8)$; the circulant matrix with coefficients (0x01,0x01,0x04,0x8d) uses $(0 + 0 + 6 + 3) + 3 \times 8 = 33$ XORs to implement per row over $GF(2^8)$. This explains why the circulant matrix with coefficients (0x01,0x01,0x04,0x8d) is lighter than the AES matrix.

For a fair comparison, when decryption also needs to be lightweight, it is to be noted that this matrix presents less interesting inverse coefficients (0x71,0x12,0xdd,0x20) compared to the ones in the AES diffusion matrix inverse (0x0b,0x0d,0x09,0x0e). According to Table 4, the former matrix requires 98 XORs, while the latter requires 86 XORs to be implemented.

Table 3: XORs required to implement a multiplication by x over $GF(2^4)$ using different irreducible polynomials.

x	$\alpha^4+\alpha+1$	$\alpha^4+\alpha^3+1$	$\alpha^4+\alpha^3+\alpha^2+\alpha+1$
0	0	0	0
0x1	0	0	0
0x2	1	1	3
0x3	5	3	5
0x4	2	3	3
0x5	6	5	5
0x6	5	2	6
0x7	9	6	6
0x8	3	6	3
0x9	1	8	5
0xa	8	5	6
0xb	6	9	6
0xc	5	1	6
0xd	3	5	6
0xe	8	6	5
0xf	6	8	3

7.2 Subfield construction

In this section, we describe the subfield construction which allows us to outperform the `AES` matrix even more than the optimal matrix found in Section 7.1. As computed in the previous subsection, the MDS circulant matrix $circ(0x1, 0x1, 0x4, 0x9)$ over $GF(2^4)$ defined by $\alpha^4+\alpha+1$ requires 15 XORs to implement per row. Using the method of [13, Section 3.3], we can form a circulant MDS matrix over $GF(2^8)$ by using two parallel copies of $Q = circ(0x1, 0x1, 0x4, 0x9)$ over $GF(2^4)$. The matrix is formed by writing each byte q_j as a concatenation of two nibbles $q_j = (q_j^L||q_j^R)$. Then the MDS multiplication is computed on each half $(u_1^L, u_2^L, u_3^L, u_4^L) = Q \cdot (q_1^L, q_2^L, q_3^L, q_4^L)$ and $(u_1^R, u_2^R, u_3^R, u_4^R) = Q \cdot (q_1^R, q_2^R, q_3^R, q_4^R)$ over $GF(2^4)$. The result is concatenated to form four output bytes (u_1, u_2, u_3, u_4) where $u_j = (u_j^L||u_j^R)$. This matrix needs just $15 \times 2 = 30$ XORs to implement per row. In comparison, the lightest MDS circulant matrix $circ(0x01,0x01,0x04,0x8d)$ over $GF(2^8)$ defined by $\alpha^8+\alpha^4+\alpha^3+\alpha+1$ requires more XORs (33 XORs per row).

Further, we can serialize the above multiplication to do the left half followed by the right half, in which case only 15 XORs are needed to implement one row of the MDS matrix over $GF(2^8)$. Another advantage of subfield construction is exemplified by the `SPN-Hash` construction [13]. It is difficult to find an 8×8 serial MDS matrix over $GF(2^8)$ exhaustively. Instead, two parallel copies of the `PHOTON` 8×8 serial MDS matrix over $GF(2^4)$ were concatenated to form the 8×8 serial MDS matrix over $GF(2^8)$ for `SPN-Hash`.

It is straightforward to generalize this method to form a diffusion matrix with branch number B over $GF(2^s)$ from s/i copies of a diffusion matrix of the same branch number over a subfield $GF(2^i)$, where i divides s.

7.3 Good matrices

In this section, we list optimal low-weight circulant and serial matrices of different branch number over the finite fields $GF(2)$, $GF(2^2)$, $GF(2^4)$ and $GF(2^8)$. Using the construction of Section 7.2, we can form diffusion matrices to transform nibbles and bytes from these subfields.

The optimal matrices are found by exhaustively checking the branch number of all matrices and choosing the one with the least number of XORs according to the method explained in Section 7.1. To

This idea of subfield construction was used in the `SHA3` submission `ECHO` [8] and later in `WHIRLWIND` [6] and `SPN-Hash` [13].

Table 4: XORs required to implement a multiplication by x over $GF(2^8)$ using different irreducible polynomials. `11b` and `11d` denote $\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1$ and $\alpha^8 + \alpha^4 + \alpha^3 + \alpha^2 + 1$ respectively

x	11b	11d	x	11b	11d	x	11b	11d	x	11b	11d	x	11b	11d	x	11b	11d	x	11b	11d	x	11b	11d
0	0	0	0x20	16	16	0x40	19	19	0x60	23	19	0x80	22	21	0xa0	28	29	0xc0	27	20	0xe0	25	24
0x01	0	0	0x21	14	14	0x41	27	15	0x61	21	25	0x81	16	29	0xa1	32	27	0xc1	21	16	0xe1	29	30
0x02	3	3	0x22	13	13	0x42	16	16	0x62	30	26	0x82	27	16	0xa2	27	30	0xc2	22	25	0xe2	30	19
0x03	11	11	0x23	11	11	0x43	24	12	0x63	28	32	0x83	21	24	0xa3	31	28	0xc3	16	21	0xe3	34	25
0x04	6	6	0x24	22	26	0x44	17	17	0x64	17	17	0x84	18	17	0xa4	24	21	0xc4	31	28	0xe4	25	32
0x05	14	14	0x25	20	24	0x45	25	13	0x65	15	23	0x85	12	25	0xa5	28	19	0xc5	25	24	0xe5	29	38
0x06	13	13	0x26	19	23	0x46	10	10	0x66	24	24	0x86	27	12	0xa6	23	26	0xc6	30	33	0xe6	30	31
0x07	21	21	0x27	17	21	0x47	18	6	0x67	22	30	0x87	21	20	0xa7	27	24	0xc7	24	29	0xe7	34	37
0x08	9	9	0x28	25	21	0x48	22	30	0x68	26	26	0x88	21	20	0xa8	27	20	0xc8	20	17	0xe8	10	21
0x09	17	17	0x29	23	19	0x49	30	26	0x69	24	32	0x89	15	28	0xa9	31	18	0xc9	14	13	0xe9	14	27
0x0a	18	16	0x2a	24	18	0x4a	21	27	0x6a	31	29	0x8a	28	15	0xaa	24	25	0xca	13	26	0xea	17	16
0x0b	26	24	0x2b	22	16	0x4b	29	23	0x6b	29	35	0x8b	22	23	0xab	28	23	0xcb	7	22	0xeb	21	22
0x0c	15	15	0x2c	31	27	0x4c	20	24	0x6c	28	16	0x8c	9	12	0xac	23	12	0xcc	24	25	0xec	18	25
0x0d	23	23	0x2d	29	25	0x4d	28	20	0x6d	26	22	0x8d	3	20	0xad	27	10	0xcd	18	21	0xed	22	31
0x0e	20	22	0x2e	30	28	0x4e	15	21	0x6e	33	23	0x8e	20	3	0xae	20	17	0xce	21	30	0xee	25	20
0x0f	28	30	0x2f	28	26	0x4f	23	17	0x6f	31	29	0x8f	14	11	0xaf	24	15	0xcf	15	26	0xef	29	26
0x10	12	12	0x30	20	18	0x50	27	25	0x70	23	23	0x90	22	35	0xb0	36	29	0xd0	27	28	0xf0	25	34
0x11	12	12	0x31	26	24	0x51	27	29	0x71	29	21	0x91	24	35	0xb1	32	35	0xd1	29	32	0xf1	21	32
0x12	19	21	0x32	17	17	0x52	24	20	0x72	26	32	0x92	31	28	0xb2	35	32	0xd2	22	31	0xf2	34	31
0x13	19	21	0x33	23	23	0x53	24	24	0x73	32	30	0x93	33	28	0xb3	31	38	0xd3	24	35	0xf3	30	29
0x14	22	18	0x34	26	24	0x54	25	19	0x74	13	21	0x94	22	27	0xb4	32	21	0xd4	31	28	0xf4	29	38
0x15	22	18	0x35	32	30	0x55	25	23	0x75	19	19	0x95	24	27	0xb5	28	27	0xd5	33	32	0xf5	25	36
0x16	29	27	0x36	27	19	0x56	22	14	0x76	20	26	0x96	31	24	0xb6	27	24	0xd6	26	35	0xf6	34	35
0x17	29	27	0x37	33	25	0x57	22	18	0x77	26	24	0x97	33	24	0xb7	23	30	0xd7	28	39	0xf7	30	33
0x18	17	17	0x38	21	23	0x58	34	28	0x78	26	34	0x98	21	26	0xb8	31	24	0xd8	28	13	0xf8	22	31
0x19	17	17	0x39	27	29	0x59	34	32	0x79	32	32	0x99	23	26	0xb9	27	30	0xd9	30	17	0xf9	18	29
0x1a	26	22	0x3a	16	22	0x5a	29	23	0x7a	31	39	0x9a	28	19	0xba	32	31	0xda	25	20	0xfa	29	28
0x1b	26	22	0x3b	22	28	0x5b	29	27	0x7b	37	37	0x9b	30	19	0xbb	28	37	0xdb	27	24	0xfb	25	26
0x1c	19	23	0x3c	27	33	0x5c	32	26	0x7c	24	32	0x9c	13	22	0xbc	27	24	0xdc	32	21	0xfc	34	39
0x1d	19	23	0x3d	33	39	0x5d	32	30	0x7d	30	30	0x9d	15	22	0xbd	23	30	0xdd	34	25	0xfd	30	37
0x1e	28	32	0x3e	26	32	0x5e	27	25	0x7e	33	37	0x9e	20	15	0xbe	24	27	0xde	29	28	0xfe	37	32
0x1f	28	32	0x3f	32	38	0x5f	27	29	0x7f	39	35	0x9f	22	15	0xbf	20	33	0xdf	31	32	0xff	33	30

check the branch number of matrix Q, we concatenate it with the identity matrix I_r to form $(I_r|Q)$, the generating matrix of the corresponding linear code, and use the MAGMA software to find the distance. When we say a matrix has branch number B, we mean the matrix has both differential and linear branch number equal to B. That is, we check that both Q and its transpose Q^t has branch number B.

We are aware that better techniques than naive exhaustive search might be used here. However, such improvements are not the goal of this article and we leave them as potential future work.

Circulant matrices. In Table 5, we list optimal $r \times r$ circulant matrices with branch numbers 3 to $r+1$. The "First Row" column represents the first row of the circulant matrix (as described in Section 4). The "Number of XORs" column represents the number of XOR gates needed to implement one row of the circulant matrix.

The matrices are optimal in the sense that they need minimal number of XORs to implement. In the events of a tie between two matrices, possibly over different finite field representations, we just list one of them. For example, the circulant matrices $circ$(0x01,0x01,0x04,0x8d) over $GF(2^8)$ defined by $\alpha^8+\alpha^4+\alpha^3+\alpha+1$ and $circ$(0x01,0x01,0x04,0x8e) over $GF(2^8)$ defined by $\alpha^8+\alpha^4+\alpha^3+\alpha^2+1$ both outperforms the `AES` matrix by using 33 XORs to implement one row, so we just list the latter. We use "-" to represent that no circulant matrix with branch number B exists (verified either by exhaustive search or by coding theory bounds). For example, it can be verified that 8×8 circulant MDS matrix does not exist in the finite field $GF(2^4)$.

The only exception where we did not find the optimal matrix is for 8×8 circulant MDS matrix over $GF(2^8)$. Because the search space is too big to exhaust, we just list the `WHIRLPOOL` matrix which is MDS and low weight.

Serial matrices. Here, in a similar fashion to the case of circulant matrices, we provide optimal low-weight serial matrices of various branch numbers over different finite fields.

In Tables 5, we list optimal $r \times r$ serial matrices with branch numbers 3 to $r+1$. The "Last Row" column represents the last row of the serial matrix (as described in Section 4). The "Number of XORs" column represents the number of XOR gates needed to implement the last row. Again, we simply list one matrix in the event of a tie, and use "-" to represent that no serial matrix with branch number B exists. In addition, we use "*" to denote that we have not found the serial matrix with branch number B at this point of time due to the huge search space. For instance, as the search space is too big to exhaust, we could not find a 8×8 serial MDS matrix over $GF(2^8)$. In this case, we can employ the method of subfield construction (described in Section 7.2), i.e. use two parallel copies of the 8×8 MDS serial matrix with last row (0x2,0xd,0x2,0x4,0x3,0xd,0x5,0x2) (refer to second row of 8×8 subtable of Table 5) over $GF(2^4)$ to obtain the desired 8×8 serial MDS matrix over $GF(2^8)$.

Note that the special structure of the serial matrices leads to the fact that only XORs for the last row are required. In particular, for an $r \times r$ serial matrix, we just need to implement the last row as an LFSR feedback function and iterate it r times to obtain the required matrix multiplication. This tactic has been adopted in [16,17] to define the `PHOTON` and `LED` MDS matrices over $GF(2^4)$ respectively. The last rows of the serial matrices are (0x2,0x4,0x2,0xb,0x2,0x8,0x5,0x6) and (0x4,0x1,0x2,0x2) respectively, which require 53 and 16 XORs to implement per row. In fact, we have found lighter serial matrices over the same finite field: 8×8 serial matrix with last row (0x2,0xd,0x2,0x4,0x3,0xd,0x5,0x2) requiring 50 XORs; and 4×4 serial matrix with last row (0x2,0x1,0x1,0x4) requiring 15 XORs (refer to 4×4 and 8×8 subtables of Table 5).

7.4 Application: FOAM Comparison for 64-bit SPN Structures

In this section, we compare the FOAM metric for 64-bit SPN Structures (a typical blocksize for lightweight block ciphers). Table 6 (resp. Table 7) gives the results for a SPN structure based circulant matrices (resp. serial matrices) and with 4-bit `PRESENT` S-box or 8-bit `AES` S-box. The diffusion matrices are based on the optimal matrices found in Section 7.3. To compute $p(2^{64})$, the number of rounds to achieve differential/linear probability $\leq 2^{-64}$, we use the fact that the differential/linear probability of the `PRESENT` S-box is 2^{-2} and that of the `AES` S-box is 2^{-6}. Then we lower bound the number of active S-boxes by concatenating 4-round bounds with $B \times B'$ active S-boxes from Theorem 1, 2-round bounds with B

Table 5: Good circulant and serial matrices of Size 2 × 2, 4 × 4 and 8 × 8

2 × 2					
Finite Field	Branch Number B	Circulant matrices		Serial matrices	
		First Row (hexadecimal)	Number of XORs A	Last Row (hexadecimal)	Number of XORs A
$GF(2^8)$, $\alpha^8+\alpha^4+\alpha^3+\alpha^2+1$	3	1,2	11	1,2	11
$GF(2^4)$, $\alpha^4+\alpha+1$	3	1,2	5	1,2	5
$GF(2^2)$, $\alpha^2+\alpha+1$	3	1,2	3	1,2	3
$GF(2)$	3	-	-	-	-

4 × 4					
Finite Field	Branch Number B	Circulant matrices		Serial matrices	
		First Row (hexadecimal)	Number of XORs A	Last Row (hexadecimal)	Number of XORs A
$GF(2^8)$, $\alpha^8+\alpha^4+\alpha^3+\alpha^2+1$	5	1,1,4,8e	33	1,2,1,4	33
	4	1,1,1,0	16	1,0,2,1	19
	3	1,0,0,2	11	1,0,0,1	8
$GF(2^4)$, $\alpha^4+\alpha+1$	5	1,1,4,9	15	2,1,1,4	15
	4	1,1,1,0	8	1,0,2,1	9
	3	1,0,0,2	5	1,0,0,1	4
$GF(2^2)$, $\alpha^2+\alpha+1$	5	-	-	-	-
	4	1,1,1,0	4	1,0,2,1	5
	3	1,0,0,2	3	1,0,0,1	2
$GF(2)$	5	-	-	-	-
	4	1,1,1,0	2	-	-
	3	-	-	1,0,0,1	1

8 × 8					
Finite Field	Branch Number B	Circulant matrices		Serial matrices	
		First Row (hexadecimal)	Number of XORs A	Last Row (hexadecimal)	Number of XORs A
$GF(2^8)$, $\alpha^8+\alpha^4+\alpha^3+\alpha^2+1$	9	1,1,4,1,8,5,2,9	105	*	*
	8	1,0,1,1,2,2,1,8e	57	1,1,2,0,1,8d,2,1	57
	7	1,0,0,1,1,1,2,8e	46	1,1,2,1,0,0,1,8d	46
	6	1,0,0,0,1,1,1,2	35	1,1,0,0,1,1,2,0	35
	5	1,0,0,0,0,1,1,2	27	1,0,0,1,1,1,0,0	24
	4	1,0,0,0,0,0,1,1	16	1,0,0,0,0,1,1,0	16
	3	1,0,0,0,0,0,0,2	11	1,0,0,0,0,0,1,0	8
$GF(2^4)$, $\alpha^4+\alpha+1$	9	-	-	2,d,2,4,3,d,5,2	50
	8	1,0,1,1,2,9,2,1	27	*	*
	7	1,0,0,1,1,1,2,9	22	1,0,2,1,1,1,2,0	22
	6	1,0,0,0,1,1,1,2	17	1,1,0,0,1,1,2,0	17
	5	1,0,0,0,0,1,1,2	13	1,0,0,1,1,1,0,0	12
	4	1,0,0,0,0,0,1,1	8	1,0,0,0,0,1,1,0	8
	3	1,0,0,0,0,0,0,2	5	1,0,0,0,0,0,1,0	4
$GF(2^2)$, $\alpha^2+\alpha+1$	9 - 8	-	-	-	-
	7	-	-	2,1,0,3,1,2,0,1	13
	6	1,0,0,0,1,1,1,2	9	1,0,0,1,1,1,0,2	9
	5	1,0,0,0,0,1,1,2	7	1,0,0,1,1,1,0,0	6
	4	1,0,0,0,0,0,1,1	4	1,0,0,0,0,1,1,0	4
	3	1,0,0,0,0,0,0,2	3	1,0,0,0,0,0,1,0	2
$GF(2)$	9 - 6	-	-	-	-
	5	-	-	1,0,0,1,1,1,0,0	3
	4	1,0,0,0,0,0,1,1	2	1,0,0,0,0,1,1,0	2
	3	-	-	1,0,0,0,0,0,1,0	1

active S-boxes and 1-round bound which involves only 1 active S-box. We also write down t, the time to compute one round for serialized implementation (the time t for round based implementation is the constant 1, so it is not presented).

We compute the FOAM for round-based and serialized implementation based on the formula found in Section 6. We also present the FOAM for half-half implementation, where we take the average, i.e. equal weighting, of the round-based and serialized FOAM. This corresponds to implementations which are good for both scenarios. However, please note that this represents just one example, as the weighting of the scenarios is clearly a designer's choice. The structure with the best area and FOAMs are presented in bold font.

We see that for designing 64-bit SPN:

1. For minimal area the geometry is the most important criterion, while the choice of the field of the MDS matrix is of less importance. The geometry should be chosen, such that c is maximized, and consequently, many internal columns can be realized with 1-input flip-flops. A serial matrix is favorable over a circulant matrix and in general smaller fields allow to save a few GE, but come at a high timing overhead.
2. When circulant matrices are used with `PRESENT` S-box in Table 6, the 4×4 almost-MDS circulant matrix $circ(0x1, 0x1, 0x1, 0x0)$ over $GF(2^4)$ gives the best FOAM for round-based, serial and half-half implementations.
3. When circulant matrices are used with `AES` S-box in Table 6, two parallel copies of the 4×4 MDS circulant matrix $circ(0x1, 0x1, 0x4, 0x9)$ over $GF(2^4)$ defined by $\alpha^4 + \alpha + 1$ gives the best FOAM for round-based implementation. The 4×4 MDS circulant matrix $circ(0x01, 0x01, 0x04, 0x8e)$ over $GF(2^8)$ defined by $\alpha^8+\alpha^4+\alpha^3+\alpha^2+1$ gives the best FOAM for serial and half-half implementations.
4. When serial matrices are used with `PRESENT` S-box in Table 7, the 4×4 almost-MDS serial matrix with last row $(0x1, 0x0, 0x2, 0x1)$ over $GF(2^4)$ defined by $\alpha^4 + \alpha + 1$ gives the best FOAM for round-based, serial and half-half implementations.
5. When serial matrices are used with `AES` S-box in Table 7, two parallel copies of the 4×4 MDS serial matrix with last row $(0x2, 0x1, 0x1, 0x4)$ over $GF(2^4)$ defined by $\alpha^4 + \alpha + 1$ gives the best FOAM for round-based implementation. The 8×8 serial matrix (having branch number 6) with last row $(0x01, 0x01, 0x00, 0x00, 0x01, 0x01, 0x02, 0x00)$ over $GF(2^8)$ defined by $\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1$ gives the best FOAM for serial and half-half implementations and is also very competitive for round-based FOAMs. It is thus a very interesting choice for many different applications.
6. Structures based on `PRESENT` S-box have higher FOAM for round-based and half-half implementations than those based on `AES` S-box. On the other hand, structures based on `AES` S-box have higher FOAM for serial implementation than `PRESENT` S-box, because they need significantly less rounds.
7. For structures using both types of S-boxes, 4×4 matrices have higher FOAM than 2×2 and 8×8 matrices.
8. Based on the above observations, we do not always go for the matrix with the best branch number: for `PRESENT` S-box in Tables 6 and 7, we use almost-MDS 4×4 matrix which gives better trade-offs and a higher FOAM than MDS matrix. Moreover in Tables 6 and 7, when `AES` S-box is used with 8×8 matrices, we go for the one with branch number 6 instead of the optimal 9.

7.5 Designs with Optimal FOAM for Different Block Sizes

In Section 7.4, we showed a detailed comparison table for all possible configurations of 64-bit SPN structures based on `AES` and `PRESENT` S-box. From it, we extract the optimal design with the highest FOAM, which gives the best trade-off between speed, area and security.

In this section, we apply the same computations to other common block sizes which are used in the construction of block ciphers and hash functions. The block sizes we consider are 48, 64, 96, 128, 256 and 512 bits. For conciseness and ease of reference, we only list the best FOAM values for each of these block sizes. In Table 8, we list the designs with the best FOAM values based on circulant matrices. In Table 9, we list the designs with the best FOAM values based on serial matrices.

When computing Table 8, we found that out of the 19 configurations for 128-bit block size based on `AES` S-box and circulant matrices, the best FOAM is given by a 4×4 state array with an MDS matrix. This corresponds to the `AES` block cipher structure and shows that the best design picked by our FOAM measure corresponds to the best design picked by human intuition.

Table 6: FOAM for 64-bit SPN based on circulant matrices and 4-bit PRESENT S-box or 8-bit AES S-box

4-bit PRESENT S-box										
Finite Field	r	c	B	$p(2^{64})$	t	Area (GE) rd based	Area (GE) serial	FOAM $\times 10^{-9}$ rd based	FOAM $\times 10^{-9}$ serial	FOAM $\times 10^{-9}$ half-half
$GF(2^4)$	2	8	3	16	55	1156	541	46.76	3.88	25.32
$GF(2^2)$	2	8	3	16	87	1199	**540**	43.48	2.46	22.97
$GF(2^4)$	4	4	5	8	51	1579	652	50.16	5.77	27.96
	4	4	4	8		1280	633	**76.34**	**6.12**	**41.23**
	4	4	3	16		1156	630	46.76	3.09	24.92
$GF(2^2)$	4	4	4	8	83	1280	627	76.34	3.83	40.08
	4	4	3	16		1199	629	43.48	1.90	22.69
$GF(2)$	4	4	4	8	147	1280	624	76.34	2.18	39.26
$GF(2^4)$	8	2	8	8	49	2091	873	28.58	3.35	15.96
	8	2	7	10		1882	864	28.22	2.73	15.48
	8	2	6	12		1669	851	29.92	2.35	16.14
	8	2	5	14		1498	840	31.83	2.07	16.95
	8	2	4	16		1284	827	37.89	1.87	19.88
	8	2	3	22		1161	823	33.73	1.37	17.55
$GF(2^2)$	8	2	6	12	81	1712	834	28.45	1.48	14.96
	8	2	5	14		1541	829	30.09	1.28	15.69
	8	2	4	16		1284	821	37.89	1.15	19.52
	8	2	3	22		1204	823	31.38	0.83	16.10
$GF(2)$	8	2	4	16	145	1284	818	37.89	0.64	19.27

8-bit AES S-box										
Finite Field	r	c	B	$p(2^{64})$	t	Area (GE) rd based	Area (GE) serial	FOAM $\times 10^{-9}$ rd based	FOAM $\times 10^{-9}$ serial	FOAM $\times 10^{-9}$ half-half
$GF(2^8)$	2	4	3	6	27	2685	822	23.12	9.14	16.13
$GF(2^4)$	2	4	3	6	43	2663	**815**	23.49	5.83	14.66
$GF(2^2)$	2	4	3	6	75	2706	**815**	22.76	3.35	13.05
$GF(2^8)$	4	2	5	4	25	3150	1029	25.19	**9.44**	**17.32**
	4	2	4	6		2792	988	21.39	6.82	14.10
	4	2	3	8		2685	975	17.34	5.26	11.30
$GF(2^4)$	4	2	5	4	41	3086	990	**26.25**	6.22	16.23
	4	2	4	6		2792	976	21.39	4.26	12.82
	4	2	3	8		2663	968	17.62	3.25	10.44
$GF(2^2)$	4	2	4	6	73	2792	970	21.39	2.42	11.91
	4	2	3	8		2706	968	17.07	1.83	9.45
$GF(2)$	4	2	4	6	137	2792	967	21.39	1.30	11.34
$GF(2^8)$	8	1	9	4	23	4688	1336	11.38	6.09	8.73
	8	1	8	4		3663	1208	18.63	7.45	13.04
	8	1	7	4		3428	1179	21.28	7.82	14.55
	8	1	6	4		3193	1149	24.52	8.23	16.38
	8	1	5	5		3027	1133	21.83	6.78	14.31
	8	1	4	6		2792	1103	21.39	5.95	13.67
	8	1	3	8		2685	1090	17.34	4.58	10.96
$GF(2^4)$	8	1	8	4	39	3599	1148	19.30	4.86	12.08
	8	1	7	4		3385	1135	21.82	4.98	13.40
	8	1	6	4		3171	1121	24.86	5.10	14.98
	8	1	5	5		3005	1115	22.14	4.12	13.13
	8	1	4	6		2792	1102	21.39	3.52	12.45
	8	1	3	8		2663	1094	17.62	2.68	10.15
$GF(2^2)$	8	1	6	4	71	3214	1105	24.20	2.89	13.54
	8	1	5	5		3048	1104	21.53	2.31	11.92
	8	1	4	6		2792	1096	21.39	1.95	11.67
	8	1	3	8		2706	1093	17.07	1.47	9.27
$GF(2)$	8	1	4	6	135	2792	1093	21.39	1.03	11.21

Table 7: FOAM for 64-bit SPN based on serial matrices and 4-bit PRESENT S-box or 8-bit AES S-box

4-bit PRESENT S-box										
Finite Field	r	c	B	$p(2^{64})$	t	Area (GE) rd based	Area (GE) serial	FOAM $\times 10^{-9}$ rd based	FOAM $\times 10^{-9}$ serial	FOAM $\times 10^{-9}$ half-half
$GF(2^4)$	2	8	3	16	39	1156	513	46.76	6.09	26.42
$GF(2^2)$	2	8	3	16	63	1199	**508**	43.48	3.85	23.67
$GF(2^4)$	4	4	5	8	35	1579	586	50.16	10.39	30.27
	4	4	4	8		1322	570	**71.48**	**10.99**	**41.23**
	4	4	3	16		1113	561	50.41	5.66	28.04
$GF(2^2)$	4	4	4	8	55	1365	559	67.08	7.26	37.17
	4	4	3	16		1113	556	50.41	3.67	27.04
$GF(2)$	4	4	3	16	87	1113	553	50.41	2.35	26.38
$GF(2^4)$	8	2	9	6	33	3074	794	17.64	8.01	12.82
	8	2	7	10		1882	724	28.22	5.78	17.00
	8	2	6	12		1669	711	29.92	5.00	17.46
	8	2	5	14		1455	697	33.73	4.45	19.09
	8	2	4	16		1284	687	37.89	4.02	20.95
	8	2	3	22		1118	681	36.36	2.97	19.67
$GF(2^2)$	8	2	7	10	51	2053	700	23.72	4.00	13.86
	8	2	6	12		1712	689	28.45	3.44	15.94
	8	2	5	14		1455	681	33.73	3.02	18.37
	8	2	4	16		1284	676	37.89	2.68	20.29
	8	2	3	22		1118	675	36.36	1.95	19.16
$GF(2)$	8	2	5	14	83	1455	673	33.73	1.90	17.81
	8	2	4	16		1284	671	37.89	1.67	19.78
	8	2	3	22		1118	673	36.36	1.21	18.78

8-bit AES S-box										
Finite Field	r	c	B	$p(2^{64})$	t	Area (GE) rd based	Area (GE) serial	FOAM $\times 10^{-9}$ rd based	FOAM $\times 10^{-9}$ serial	FOAM $\times 10^{-9}$ half-half
$GF(2^8)$	2	4	3	6	19	2685	766	23.12	14.96	19.04
$GF(2^4)$	2	4	3	6	31	2663	750	23.49	9.56	16.53
$GF(2^2)$	2	4	3	6	47	2706	**744**	22.76	6.40	14.58
$GF(2^8)$	4	2	5	4	17	3150	898	25.19	18.22	21.71
	4	2	4	6		2856	866	20.44	13.08	16.76
	4	2	3	8		2621	836	18.20	10.51	14.36
$GF(2^4)$	4	2	5	4	27	3086	850	**26.25**	12.80	19.53
	4	2	4	6		2834	839	20.75	8.77	14.76
	4	2	3	8		2621	826	18.20	6.79	12.49
$GF(2^2)$	4	2	4	6	43	2877	828	20.13	5.65	12.89
	4	2	3	8		2621	820	18.20	4.32	11.26
$GF(2)$	4	2	3	8	75	2621	818	18.20	2.49	10.35
$GF(2^8)$	8	1	8	4	16	3663	928	18.63	18.15	18.39
	8	1	7	4		3428	899	21.28	19.35	20.32
	8	1	6	4		3193	869	24.52	**20.68**	**22.60**
	8	1	5	5		2963	844	22.79	17.53	20.16
	8	1	4	6		2792	823	21.39	15.38	18.38
	8	1	3	8		2621	802	18.20	12.15	15.18
$GF(2^4)$	8	1	9	4	24	4581	920	11.91	12.31	12.11
	8	1	7	4		3385	845	21.82	14.59	18.20
	8	1	6	4		3171	832	24.86	15.06	19.96
	8	1	5	5		2963	823	22.79	12.30	17.54
	8	1	4	6		2792	812	21.39	10.52	15.95
	8	1	3	8		2621	802	18.20	8.10	13.15
$GF(2^2)$	8	1	7	4	40	3556	821	19.77	9.27	14.52
	8	1	6	4		3214	810	24.20	9.52	16.86
	8	1	5	5		2963	807	22.79	7.68	15.23
	8	1	4	6		2792	802	21.39	6.48	13.93
	8	1	3	8		2621	796	18.20	4.93	11.56
$GF(2)$	8	1	5	5	72	2963	799	22.79	4.35	13.57
	8	1	4	6		2792	796	21.39	3.65	12.52
	8	1	3	8		2621	794	18.20	2.76	10.48

Table 8: FOAM for different block sizes based on circulant matrices and 4-bit PRESENT S-box or 8-bit AES S-box

4-bit PRESENT S-box											
Block Size	Finite Field	r	c	B	$p(2^{64})$	t	Area (GE) rd based	Area (GE) serial	FOAM $\times 10^{-9}$ rd based	FOAM $\times 10^{-9}$ serial	FOAM $\times 10^{-9}$ half-half
48	$GF(2^2)$	2	6	3	12	65	908	**465**	101.06	5.92	53.49
	$GF(2^4)$	2	6	3	12	41	876	466	**108.59**	9.35	**58.97**
	$GF(2^4)$	4	3	5	8	38	1192	577	88.01	**9.89**	48.95
64	$GF(2^2)$	2	8	3	16	87	1199	**540**	43.48	2.46	22.97
	$GF(2^4)$	4	4	4	8	51	1280	633	**76.34**	**6.12**	**41.23**
96	$GF(2^2)$	2	12	3	22	131	1785	**699**	14.26	0.71	7.49
	$GF(2^4)$	4	6	5	8	77	2353	812	22.59	**2.46**	**12.53**
	$GF(2^4)$	4	6	4	12	77	1909	798	**22.88**	1.70	12.29
128	$GF(2^2)$	2	16	3	29	175	2367	**854**	6.16	0.27	3.21
	$GF(2^4)$	4	8	4	16	103	2533	947	**9.74**	0.68	**5.21**
	$GF(2^2)$	4	8	4	16	167	2533	941	**9.74**	0.42	5.08
	$GF(2)$	4	8	4	16	295	2533	938	**9.74**	0.24	4.99
	$GF(2^4)$	8	4	8	8	99	4152	1176	7.25	**0.91**	4.08
256	$GF(2^4)$	2	32	3	58	223	4527	**1468**	0.84	0.04	0.44
	$GF(2^2)$	2	32	3	58	351	4698	**1468**	0.78	0.02	0.40
	$GF(2^4)$	8	8	8	8	199	8272	1785	**1.83**	**0.20**	**1.01**
512	$GF(2^2)$	2	64	3	115	703	9355	**2678**	0.099	0.002	0.051
	$GF(2^4)$	8	16	8	16	399	16518	2996	**0.229**	**0.017**	**0.123**

8-bit AES S-box											
Block Size	Finite Field	r	c	B	$p(2^{64})$	t	Area (GE) rd based	Area (GE) serial	FOAM $\times 10^{-9}$ rd based	FOAM $\times 10^{-9}$ serial	FOAM $\times 10^{-9}$ half-half
48	$GF(2^2)$	2	3	3	4	56	2033	**735**	60.50	8.26	34.38
	$GF(2^4)$	2	3	3	4	20	2017	742	61.47	**22.68**	**42.07**
	$GF(2^8)$	2	3	3	4	32	2001	736	**62.46**	14.43	38.44
64	$GF(2^4)$	2	4	3	6	43	2663	**815**	23.49	5.83	14.66
	$GF(2^2)$	2	4	3	6	75	2706	**815**	22.76	3.35	13.05
	$GF(2^8)$	4	2	5	4	25	3150	1029	25.19	**9.44**	**17.32**
	$GF(2^4)$	4	2	5	4	41	3086	990	**26.25**	6.22	16.23
96	$GF(2^4)$	2	6	3	8	65	3980	**975**	7.89	2.02	4.96
	$GF(2^2)$	2	6	3	8	113	4044	**975**	7.64	1.16	4.40
	$GF(2^8)$	4	3	5	5	38	4717	1188	8.99	**3.73**	**6.36**
	$GF(2^4)$	4	3	5	5	62	4621	1149	**9.37**	2.44	5.91
128	$GF(2^4)$	2	8	3	11	87	5301	**1129**	3.24	0.82	2.03
	$GF(2^2)$	2	8	3	11	151	5386	**1129**	3.13	0.47	1.80
	$GF(2^8)$	4	4	5	4	51	6274	1333	6.35	**2.76**	**4.56**
	$GF(2^4)$	4	4	5	4	83	6146	1294	**6.62**	1.80	4.21
256	$GF(2^4)$	2	16	3	20	175	10570	**1742**	0.45	0.09	0.27
	$GF(2^2)$	2	16	3	20	303	10741	**1742**	0.43	0.05	0.24
	$GF(2^4)$	4	8	5	8	167	12270	1907	**0.83**	0.21	0.52
	$GF(2^8)$	8	4	9	4	99	18673	2434	0.72	**0.43**	**0.57**
512	$GF(2^4)$	2	32	3	40	351	21105	**2953**	0.056	0.008	0.032
	$GF(2^2)$	2	32	3	40	607	21447	**2953**	0.054	0.005	0.030
	$GF(2^8)$	8	8	9	6	199	37324	3645	0.120	**0.063**	0.091
	$GF(2^8)$	8	8	7	8	199	27243	3487	0.168	0.052	**0.110**
	$GF(2^4)$	8	8	7	8	327	26901	3433	**0.173**	0.032	0.103

Table 9: FOAM for different block sizes based on serial matrices and 4-bit PRESENT S-box or 8-bit AES S-box

4-bit PRESENT S-box											
Block Size	Finite Field	r	c	B	$p(2^{64})$	t	Area (GE) rd based	Area (GE) serial	FOAM $\times 10^{-9}$ rd based	FOAM $\times 10^{-9}$ serial	FOAM $\times 10^{-9}$ half-half
48	$GF(2^2)$	2	6	3	12	47	908	**433**	101.06	9.47	55.26
	$GF(2^4)$	2	6	3	12	29	876	438	**108.59**	14.97	**61.78**
	$GF(2^4)$	4	3	5	8	26	1192	511	88.01	**18.38**	53.19
64	$GF(2^2)$	2	8	3	16	63	1199	**508**	43.48	3.85	23.67
	$GF(2^4)$	4	4	4	8	35	1322	570	**71.48**	**10.99**	**41.23**
96	$GF(2^2)$	2	12	3	22	95	1785	**666**	14.26	1.08	7.67
	$GF(2^4)$	4	6	5	8	53	2353	746	**22.59**	**4.23**	**13.41**
128	$GF(2^2)$	2	16	3	29	175	2367	**854**	6.16	0.27	3.21
	$GF(2^4)$	4	8	5	12	103	3131	966	8.50	**0.87**	4.68
	$GF(2^4)$	4	8	4	16	103	2618	950	**9.12**	0.67	**4.89**
256	$GF(2^4)$	2	32	3	58	159	4527	**1468**	0.84	0.05	0.45
	$GF(2^2)$	2	32	3	58	255	4698	**1468**	0.78	0.03	0.41
	$GF(2^4)$	8	8	9	8	135	12198	1842	0.84	**0.27**	0.56
	$GF(2^4)$	8	8	7	12	135	7423	1776	**1.51**	0.20	**0.85**
512	$GF(2^2)$	2	64	3	115	703	9355	**2678**	0.099	0.002	0.051
	$GF(2^4)$	8	16	9	16	399	24379	3057	0.105	**0.017**	0.061
	$GF(2^4)$	8	16	6	30	399	13105	2974	**0.194**	0.009	**0.102**

8-bit AES S-box											
Block Size	Finite Field	r	c	B	$p(2^{64})$	t	Area (GE) rd based	Area (GE) serial	FOAM $\times 10^{-9}$ rd based	FOAM $\times 10^{-9}$ serial	FOAM $\times 10^{-9}$ half-half
48	$GF(2^2)$	2	3	3	4	35	2033	**665**	60.50	16.15	38.33
	$GF(2^8)$	2	3	3	4	14	2017	686	61.47	**37.90**	**49.68**
	$GF(2^4)$	2	3	3	4	23	2001	670	**62.46**	24.18	43.32
64	$GF(2^2)$	2	4	3	6	47	2706	**744**	22.76	6.40	14.58
	$GF(2^4)$	4	2	5	4	27	3086	850	**26.25**	12.80	19.53
	$GF(2^8)$	8	1	6	4	16	3193	869	24.52	**20.68**	**22.60**
96	$GF(2^2)$	2	6	3	8	71	4044	**905**	7.64	2.15	4.90
	$GF(2^8)$	4	3	5	5	26	4717	1057	8.99	**6.88**	**7.94**
	$GF(2^4)$	4	3	5	5	41	4621	1009	**9.37**	4.79	7.08
128	$GF(2^4)$	2	8	3	11	87	5301	**1129**	3.24	0.82	2.03
	$GF(2^2)$	2	8	3	11	151	5386	**1129**	3.13	0.47	1.80
	$GF(2^8)$	4	4	5	4	51	6274	1333	6.35	**2.76**	**4.56**
	$GF(2^4)$	4	4	5	4	83	6146	1294	**6.62**	1.80	4.21
256	$GF(2^4)$	2	16	3	20	175	10570	**1742**	0.45	0.09	0.27
	$GF(2^2)$	2	16	3	20	303	10741	**1742**	0.43	0.05	0.24
	$GF(2^8)$	4	8	5	8	103	12526	1946	0.80	**0.32**	**0.56**
	$GF(2^4)$	4	8	5	8	167	12270	1907	**0.83**	0.21	0.52
512	$GF(2^4)$	2	32	3	40	351	21105	**2953**	0.056	0.008	0.032
	$GF(2^2)$	2	32	3	40	607	21447	**2953**	0.054	0.005	0.030
	$GF(2^8)$	8	8	7	8	199	27243	3487	0.168	**0.052**	**0.110**
	$GF(2^4)$	8	8	7	8	327	26901	3433	**0.173**	0.032	0.103

8 Conclusion

We have introduced FOAM (Figure of Adversarial Merit) which for the first time allows comparison of security-time-area trade-offs. Previous metrics, such as FOM only take into account the trade-off between speed and power, or in other words implementation trade-offs. By integrating the cryptographic strength (due to the vast amount of distinct attacks, we only took in account the simple but most meaningful cryptanalysis techniques), FOAM enables a fairer comparison of the vast amount of design choices, thus easing the optimization of designs for target applications. Implementation estimates are crucial at an early design stage to make the right choices, as, e.g. for serialized architectures, the choice of the S-box size, geometry, subfield and type of matrix leads to an area range from 508 GE to 1336 GE ($\times 2.6$). In this work we have made a step into a generic hardware estimation metric that for the first time also considers control logic, which is hard to estimate.

Furthermore, we have generalized the SPN structure from a square array to a rectangular array which allows us to construct structures with more flexible sizes. A new bound for the number of active S-boxes for such structures is proven in Theorem 1. We also introduced new ways to compute lightweight coefficients of diffusion matrices, which we use to find circulant matrices which are lighter than the `AES` matrix and serial matrices which are lighter than the `LED` and `PHOTON` matrices.

Possible future works include defining a similar FOAM for software, finding new ways to define lightweight diffusion matrices (other than circulant and serial) and compute their FOAM. For example, we can construct the FOAM for SPN structures using the diffusion matrices from [2], which are baesd on BCH codes. Lastly, we need not limit ourselves to SPN structures, but also extend FOAM to different Feistel and generalized Feistel structures such as CLEFIA, SMS4 and Skipjack structures.

Acknowledgments. Thomas Peyrin is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. Anandakumar, N., Peyrin, T., Poschmann, A.: A Very Compact FPGA Implementation of the LED Block Cipher and PHOTON Hash Function. private communication (October 2013)
2. Augot, D., Finiasz, M.: Direct Construction of Recursive MDS Diffusion Layers using Shortened BCH Codes. In: FSE. Lecture Notes in Computer Science (2014) To appear.
3. Avoine, G., Oechslin, P.: A Scalable and Provably Secure Hash-Based RFID Protocol. In: PerCom Workshops, IEEE Computer Society (2005) 110–114
4. Badel, S., Dagtekin, N., Nakahara, J., Ouafi, K., Reffé, N., Sepehrdad, P., Susil, P., Vaudenay, S.: ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In: CHES. (2010) 398–412
5. Barkan, E., Biham, E.: In How Many Ways Can You Write Rijndael? In: ASIACRYPT. (2002) 160–175
6. Barreto, P.S.L.M., Nikov, V., Nikova, S., Rijmen, V., Tischhauser, E.: Whirlwind: a new cryptographic hash function. Des. Codes Cryptography **56**(2-3) (2010) 141–162
7. Barreto, P.S.L.M., Rijmen, V.: Whirlpool. In: Encyclopedia of Cryptography and Security (2nd Ed.). (2011) 1384–1385
8. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 Proposal: ECHO. Submission to NIST (2008)
9. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge functions. Ecrypt Hash Workshop 2007 (May 2007)
10. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In Paillier, P., Verbauwhede, I., eds.: CHES. Volume 4727 of LNCS., Springer (2007) 450–466 http://lightweightcrypto.org/present/.
11. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash Functions and RFID Tags: Mind the Gap. In: CHES. (2008) 283–299
12. Cannière, C.D., Dunkelman, O., Knezevic, M.: KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: CHES. (2009) 272–288
13. Choy, J., Yap, H., Khoo, K., Guo, J., Peyrin, T., Poschmann, A., Tan, C.H.: SPN-Hash: Improving the Provable Resistance against Differential Collision Attacks. In: AFRICACRYPT. (2012) 270–286
14. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In: FSE. (1997) 149–165
15. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
16. Guo, J., Peyrin, T., Poschmann, A.: The `PHOTON` Family of Lightweight Hash Functions. In: CRYPTO. (2011) 222–239
17. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In: CHES. (2011) 326–341
18. Henrici, D., Götze, J., Müller, P.: A Hash-based Pseudonymization Infrastructure for RFID Systems. In: SecPerU. (2006) 22–27
19. Lee, S.M., Hwang, Y.J., Lee, D.H., Lim, J.I.: Efficient Authentication for Low-Cost RFID Systems. In: ICCSA (1). (2005) 619–627

20. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: A very compact and a threshold implementation of aes. In: EUROCRYPT. (2011) 69–88
21. Sajadieh, M., Dakhilalian, M., Mala, H., Sepehrdad, P.: Recursive Diffusion Layers for Block Ciphers and Hash Functions. In: FSE. (2012) 385–401
22. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An Ultra-Lightweight Blockcipher. In: CHES. (2011) 342–357
23. Wu, S., Wang, M., Wu, W.: Recursive Diffusion Layers for (Lightweight) Block Ciphers and Hash Functions. In Knudsen, L., Wu, H., eds.: Selected Areas in Cryptography. Volume 7707 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 355–371

A Proof for number of active S-boxes in a GOD

A.1 Proof for Theorem 1

Proof. The case of 1 and 2 rounds being trivial (with a direct reasoning with the branching number B of the diffusion layer), we will only give the proof for the 4-round case. We can interchange the order of S-box and `GOD` and still get the same output because `GOD` is a cell-permutation. So we transform

$$\texttt{GOD} \circ \texttt{MC} \circ \texttt{SC} \circ \texttt{GOD} \circ \texttt{MC} \circ \texttt{SC} \circ \texttt{GOD} \circ \texttt{MC} \circ \texttt{SC} \circ \texttt{GOD} \circ \texttt{MC} \circ \texttt{SC}$$

to

$$\texttt{GOD} \circ \texttt{MC} \circ \texttt{GOD} \circ \texttt{SC} \circ \texttt{MC} \circ \texttt{SC} \circ \texttt{GOD} \circ \texttt{MC} \circ \texttt{GOD} \circ \texttt{SC} \circ \texttt{MC} \circ \texttt{SC}$$

This is the well-known `SuperSbox` structure for analyzing the number of active S-boxes in an SPN structure, where $\texttt{SuperSbox} = \texttt{SubCells} \circ \texttt{MixColumns} \circ \texttt{SubCells}$ and $\texttt{SuperMix} = \texttt{GOD} \circ \texttt{MixColumns} \circ \texttt{GOD}$. The number of active S-boxes in an active `SuperSbox` is B.

`SuperMix` takes in c columns and output c columns, and the minimum number of active columns in both input-output is its branch number B'. This implies there are at least B' number of active `SuperSbox` and $B' \times B$ number of active S-boxes in total. To lower bound the number of active input-output columns in `SuperMix`, we concentrate on the `MixColumns` layer.

There is at least one active `MixColumn` among the c `MixColumn` transforms in Mix. This `MixColumn` has B active cells in its input and output. The minimum number of active input-output columns after propagating these B active cells in the forward and backward directions through the two `GOD` transforms happens when there is a maximum number of columns taking $\lceil r/c \rceil$ cells (call this y) with the rest of the columns taking $\lfloor r/c \rfloor$ cells (call this x) from the active `MixColumn`. I.e. we want:

$$x \times \lfloor r/c \rfloor + y \times \lceil r/c \rceil = B, \text{ where } x \text{ and } y \text{ are integers.}$$

Actually since x and y are integers, we can only get as close to B as possible. y is upper bounded by $2(r \mod c)$ because by definition of `GOD`, there are at most $(r \mod c)$ columns in the input and output taking $\lceil r/c \rceil$ input cells each. It is also naturally upper bounded by $\lfloor B/\lceil r/c \rceil \rfloor$ for the integer equation $x \times \lfloor r/c \rfloor + y \times \lceil r/c \rceil = B$ to be true. These are the only two upper bounds on y, so we take the minimum of them to bound y.

Then x is just computed directly from y as $(B - \lceil r/c \rceil \times y)/\lfloor r/c \rfloor$. But since x may not be an integer, we need to round it up by taking ceiling. Finally, there may be cases where $x + y$ is 1 but branch number is always lower bounded by 2. So we need to use $\max\{2; x + y\}$ to compute B'.

A.2 Special cases in Corollary 1

Proof. 1. This is equivalent to the case $c = r$ where the generalized optimal diffusion is an optimal diffusion from [15]. For $c = r$, we have $(r \mod c) = 0$ and $y = 0$, $x = B$ from Theorem 1. So $B' = B$ and number of active S-boxes at least $B \times B' = B^2$.
2. When c divides r, we have $(r \mod c) = 0$ and $y = 0$, $x = \lceil B \times c/r \rceil$ from Theorem 1. If $B = r + 1$, then $x = c + 1$ and $B' = c + 1$. Number of active S-boxes is at least $B \times B' = (r + 1) \times (c + 1)$.
3. When c divides r, we have $y = 0$ and $x = \lceil B \times c/r \rceil$ as above. If $B = r$, then $x = c$ and $B' = c$. Number of active S-boxes is at least $B \times B' = r \times c$.

□