Column Parity Mixers

Ko Stoffelen[1*] and Joan Daemen[1,2]

[1] Digital Security Group, Radboud University, Nijmegen, The Netherlands
[2] STMicroelectronics, Belgium
k.stoffelen@cs.ru.nl,joan@cs.ru.nl

Abstract. We present column parity mixers (CPM), a generalization of the θ mixing layer that is used in Keccak. Thanks to our description using matrix arithmetic, we can easily derive algebraic, diffusion, and mask propagation properties, leading to a surprising distinction between two types of CPMs. We compare CPMs to other popular types of mixing layers and argue that CPMs can be more efficient. While Keccak has a bit-oriented structure, we make the case that CPMs are also suitable for nibble- or byte-oriented designs.
We outline a general substitution-permutation-network-based design strategy using a CPM, for which we show how one can attain strong bounds for differential and linear trails. We apply this strategy concretely to design a 256-bit permutation with an efficient inverse and strong trail bounds. Our permutation design uses a number of ideas that are of independent interest and allows a fast bitsliced implementation that compares quite well with other established ciphers and permutations.

Keywords: mixing layers · iterative permutations · iterative block ciphers

1 Introduction

In recent years, there has been a lot of research on lightweight cryptography. A need is perceived for cryptographic primitives that can be implemented on resource-constrained platforms, such as devices in the Internet-of-Things. Historically the most important primitives were block ciphers, but since a few years cryptographic permutations gain in popularity. Most of these ciphers consist of the repeated application of an invertible round function, where in the case of block ciphers each round takes a round key. As inspired by DES [77] and later AES [DR02], often the round function consists of a number of separate layers, each with a particular task. There is typically a non-linear S-box layer and a linear diffusion layer. In some modern ciphers, this linear layer consists of a sub-layer that mixes the bits (or bytes) and typically has a high branch number [DR02], and a transposition sub-layer that moves bits (or bytes around). In AES, the mixing layer MixColumns consists of the parallel application of a maximum-distance separable (MDS) mapping on each of the 4-byte columns. This mapping has branch number 5, the maximum attainable value. In combination with the transposition layer ShiftRows, this allows to give a simple proof for a strong upper bound on the differential probability of 4-round differential trails (characteristics) and the correlation of 4-round linear trails.

With lightweight cryptography in mind, there has been a substantial amount of publications in the last few years on constructions for lightweight MDS mappings, see e.g., [BKL16; GPP11; LS16; LW16]. This has led to a better understanding of the implementation cost of such mappings in relation to their dimensions: the number and

*This work was supported by the European Commission through the Horizon 2020 program under project number ICT-645622 (PQCRYPTO).

size of elements (e.g., 4 bytes in MixColumns). We can now build block ciphers and permutations with the same design philosophy as AES, leading to easily provable bounds with more lightweight components.

Another cipher where the designers emphasize the mixing layer is Keccak-f, the permutation underlying Keccak and the SHA-3 functions [Ber+11b; NIS15]. Its mixing layer θ does not operate on separate subblocks as MixColumns does, and it has a branch number of 4. However, despite the fact that it requires only 2 XOR operations per bit, in combination with the other layers of the round function it appears to have quite good diffusion. In particular, [MDA17] reports on computer-aided proofs for quite promising upper bounds for the differential probability of differential trails. It appears that θ-like mappings would be a good candidate for a linear mixing layer, or in general for a mixing layer with a good trade-off between implementation cost and mixing power.

There is a remarkable difference between AES and Keccak-f. In the former, all step mappings are defined in terms of operations on bytes and in the latter each step mapping treats groupings of bits along different axes. The AES design approach has received quite some following and it can be called byte-oriented (although there are also ciphers where these units are 4-bit nibbles or even 5-bit units). We call the Keccak-f design approach bit-oriented. The byte-oriented design approach has the advantage that one can easily prove bounds. We believe that θ-like mappings are suitable for both design approaches.

1.1 Our contributions

In this paper we present a generalization of the θ mixing layer in Keccak-f called column parity mixers (CPMs). CPMs operate on two-dimensional arrays and in their definition parities computed over the columns play a central role, hence the name *column parity mixers*. In Section 2 we provide an elegant description using matrix arithmetic, allowing us to easily derive algebraic, diffusion, and mask propagation properties. We also show that column parity mixers operating on states with an even number of rows have quite different properties from those operating on an odd number of rows.

The former are involutions and are ideally suited for block ciphers and permutations that need to have an efficient inverse. Coincidentally, those are the ones that are typically required to have a permutation width (or 'block size') that is a power of two, and this is nicely compatible with an even number of rows. An example is disk encryption where the disk sector size is a power of two.

The latter may have an inverse with high implementation cost but also with very interesting diffusion properties (see Section 4). They are suited for permutations used in MACs or in stream encryption, or in conjunction with authenticated encryption modes that do not require the inverse such as the sponge and duplex constructions [Ber+11a], where the permutation width is also less bound to a power of two.

We see Keccak-f as a representative example of a bit-oriented design that makes use of a column parity mixing layer and we make the case that they are also suitable for byte-oriented designs. In Section 5 we outline a general design strategy with attention for how strong bounds can be attained for differential and linear trails. We apply this strategy concretely to design a 256-bit permutation called Mixifer with an efficient inverse. This can be used as a permutation in an Even-Mansour block cipher [EM93] or in modes such as proposed in [Men16]. The width of 256 is large enough to make the birthday bound 2^{128} far enough to not pose a practical problem and it is small enough to still be called lightweight.

Our permutation design, presented in Section 6, uses a number of ideas that are of independent interest. It operates on an array of 4 rows of 16 nibbles each and we arrange the bits of the nibbles in such a way that a bitsliced implementation is immediate. Its design can be seen as a hybrid between that of AES and that of Keccak-f. We show

in Section 6.6 that in comparison with AES and some established permutations, the performance of our permutation is better or comparable.

2 Column parity mixers and their properties

Column parity mixers are generalizations of the mixing layer θ in KECCAK. Therefore, we adopt the terminology proposed in [Ber+11b]. Unlike the descriptions therein, we will treat the states that these mixing layers operate on as two-dimensional arrays that we will interpret as matrices. In the context of this section we limit ourselves to matrices with elements of GF(2), but one can easily generalize our treatment to elements of GF(p) with p a prime or even elements of an arbitrary finite group.

2.1 Matrices

We use $\mathbf{I}$ to denote a (square) identity matrix and $\mathbf{0}$ to denote an all-zero matrix. We assume that the dimensions of these matrices are determined by the context. The transpose of a matrix A is denoted as A^T.

We use $\mathbf{1}_x$ to denote a column vector of x components that are all equal to 1. Consequently, $\mathbf{1}_x^\mathsf{T}$ is an all-1 row vector with x components. We use $\mathbf{1}_x^y$ to denote a matrix with x rows and y columns with all components 1. Clearly, $\mathbf{1}_x^y = \mathbf{1}_x\mathbf{1}_y^\mathsf{T}$.

The element of a matrix A at row i and column j is denoted by $A_{i,j}$. If $B = A^\mathsf{T}$, we have $B_{i,j} = A_{j,i}$. The trace of a square matrix is the linear function that simply takes the sum of its diagonal elements. It is denoted by $\mathrm{tr}(A)$, so $\mathrm{tr}(A) = \sum_i A_{i,i}$.

2.2 Definition of column parity mixers

We consider linear mappings θ that operate on arrays with m rows and n columns.

Definition 1. The *column parity* of a matrix A is a (row) vector defined as $\mathbf{1}_m^\mathsf{T}A$.

In a matrix A, a column x is called even (odd) if the component with index x in $\mathbf{1}_m^\mathsf{T}A$ is zero (one).

Definition 2. The *expanded column parity* of A is a matrix with m rows all equal to the column parity of A, and it is given by $\mathbf{1}_m^m A$.

A column parity mixer (CPM) makes use of a linear transformation operating on the column parity of a matrix, called its *parity-folding transformation*. We denote the parity-folding transformation by multiplying the column parity with a square matrix Z at the right. We call the $n \times n$ matrix Z the *parity-folding matrix* of θ. We are now ready to define the θ-effect of a matrix A.

Definition 3. The *θ-effect* of A with respect to Z is a row vector, denoted as $\mathbf{e}_\mathbf{Z}(A)$ (or just $\mathbf{e}(A)$ if Z is clear from the context) and is defined by $\mathbf{e}_\mathbf{Z}(A) = \mathbf{1}_m^\mathsf{T}AZ$.

For a given input A and parity-folding matrix Z, a column x is called *unaffected* (affected) if the component with index x in $\mathbf{e}_\mathbf{Z}(A)$ is zero (non-zero). Whether a column is affected or not is fully determined by the column parity of A and the column x of the parity-folding matrix Z.

Definition 4. The *expanded θ effect* of A with respect to Z is a matrix with m rows all equal to the CPM effect, namely, $\mathbf{E}_\mathbf{Z}(A) = \mathbf{1}_m^m AZ$.

A column parity mixer θ simply consists in computing the expanded θ-effect of a matrix A and adding it to A.

Definition 5. The *column parity mixer* θ using parity-folding matrix Z is defined as:

$$\theta(A) = A + \mathbf{E}_{\mathbf{Z}}(A) = A + \mathbf{1}_m^m AZ\,.$$

Note that a column parity mixer is fully defined by a parity-folding matrix Z and m.

Example 1. KECCAK [Ber+11b] uses a three-dimensional structure, so, for the sake of this example, let us first 'flatten' the state by looking at a single sheet. With KECCAK-f[200], this array would have $m = 5$ rows and $n = 8$ columns. Consider the following state A:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

Then the *column parity* of A is $[1, 1, 0, 0, 0, 1, 0, 0]$, so the first two columns and the sixth column are *odd*, while the rest is *even*. The θ step in KECCAK-f affects the adjacent sheets, but one can modify it slightly such that the operation is performed within the same sheet. To the reader who is familiar with the KECCAK specification, we change $x - 1 \bmod 5$ and $x + 1 \bmod 5$ to $x \bmod 5$ in the computation of $D[x, z]$ given $C[x, z]$. This means that we can express the *parity-folding matrix* Z as follows:

$$Z = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

This yields a θ-*effect* of $\mathbf{e}(A) = [0, 1, 0, 0, 1, 1, 0, 1]$, so the second, fifth, sixth, and eighth column are *affected*, the rest is *unaffected*. The result of the *column parity mixer* defined by Z and m on A is then:

$$\theta(A) = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

2.3 Group properties

Composition of two column parity mixers is again a column parity mixer.

Lemma 1. *Let $\alpha = \theta' \circ \theta$ with θ and θ' column parity mixer with parity-folding matrices Z and Z' respectively. Then α is a column parity mixer. If m is even, the parity-folding matrix of α is $Z + Z'$. If m is odd, its parity-folding matrix is $(Z' + \mathbf{I})(Z + \mathbf{I}) + \mathbf{I}$.*

Proof. We have

$$\begin{aligned} \theta'(\theta(A)) &= A + \mathbf{1}_m^m AZ + \mathbf{1}_m^m \left(A + \mathbf{1}_m^m AZ\right) Z' \\ &= A + \mathbf{1}_m^m AZ + \mathbf{1}_m^m AZ' + (\mathbf{1}_m^m)^2 AZZ'\,. \end{aligned}$$

If m is even, we have $(\mathbf{1}_m^m)^2 = \mathbf{0}$ and this reduces to $A + \mathbf{1}_m^m A(Z + Z')$. If m is odd, we have $(\mathbf{1}_m^m)^2 = \mathbf{1}_m^m$ and we have

$$\begin{aligned}\theta'(\theta(A)) &= A + \mathbf{1}_m^m A(Z + Z' + ZZ') \\ &= A + \mathbf{1}_m^m A((Z + \mathbf{I})(Z' + \mathbf{I}) + \mathbf{I})\,.\end{aligned}$$

□

This implies the following corollary:

Corollary 1. *The set of all column parity mixers for given dimensions $m \times n$, with m even, form a group with composition that is isomorphic to the abelian group $(\mathbb{Z}_2^{n^2}, +)$.*

Proof. Lemma 1 says that the composition of two CPMs with parity-folding matrices Z and Z' is the CPM with parity-folding matrix $Z + Z'$. It follows that the set of all CPMs of given dimensions $m \times n$ and m even, is isomorphic to the set of binary $n \times n$ matrices with addition. This addition is closed and inherits the associativity and commutativity from the addition in GF(2). Its neutral element is $\mathbf{0}$ and each element is its own inverse. This is $(\mathbb{Z}_2^{n^2}, +)$. □

It follows that column parity mixers operating on matrices with an even number m of rows are involutions.

For m odd, not all column parity mixers are invertible. We have the following:

Corollary 2. *The set of column parity mixers for given dimensions $m \times n$, with m odd and $Z + \mathbf{I}$ non-singular, form a group with composition that is isomorphic to the group of binary invertible $n \times n$ matrices with multiplication.*

Proof. Lemma 1 says that the composition of two CPMs with parity-folding matrices Z and Z' is the CPM with parity-folding matrix $(Z + \mathbf{I})(Z' + \mathbf{I}) + \mathbf{I}$. Let us call $Z + \mathbf{I}$ the associated matrix of a CPM. Then the associated matrix of the composition of two CPMs is the product of their associate matrices. It follows that the set of all CPMs of given dimensions $m \times n$ and m odd, is isomorphic to the set of invertible binary $n \times n$ matrices with multiplication. This is the general linear group $GL(n, 2)$ [DF04]. □

For m odd, the order of a CPM is the multiplicative order of its associated matrix $Z + \mathbf{I}$. The associated matrix of the inverse of an invertible CPM with parity-folding matrix Z is $(Z + \mathbf{I})^{-1}$.

2.4 The special case of circulant parity-folding matrices

Z is a circulant matrix if its elements satisfy $Z_{i+j \bmod n,\ j} = Z_{i,0}$ for all i, j. Circulant matrices have become a popular building block in many ciphers, including AES [DR02] and Keccak [Ber+11b]. As the product and sum of two circulant matrices is a circulant matrix and both $\mathbf{I}$ and $\mathbf{0}$ are circulant matrices, it follows that both for m even and odd the set of invertible circulant CPMs form subgroups of the set of invertible CPMs, for given dimensions.

We can express multiplication by a circulant matrix as multiplication by a polynomial. For that purpose, we express a matrix A as a bivariate polynomial, where the element in row i and column j corresponds with the coefficient of monomial $x^i y^j$. Computing the column parity of A corresponds with multiplication by the polynomial $\sum_{i=0}^{n-1} y^i$ modulo $1 + y^n$. This polynomial can also be expressed as $\frac{1+y^n}{1+y}$. Multiplication by Z then corresponds with multiplication by a polynomial $z(x)$ modulo $1 + x^n$. So for our column parity mixer θ, we have:

$$\theta(a(x, y)) = a(x, y) + \frac{1 + y^m}{1 + y} z(x) a(x, y) \bmod (1 + x^n)(1 + y^m)\,.$$

We call $z(x)$ the parity-folding polynomial.

Circulant CPMs with even m are involutions. A circulant CPM with odd m is invertible if its associated polynomial $1 + z(x)$ is coprime to $1 + x^n$ and its inverse has the associated polynomial $y(x) = (z(x) + 1)^{-1} \bmod (1 + x^n)$. A necessary condition for invertibility is that $z(x)$ has an even number of non-zero terms. If not, $1 + x$ divides both $1 + z(x)$ and $1 + x^n$.

The transpose of θ is determined by the parity-folding polynomial $z(x^{-1}) \bmod 1 + x^n$, i.e., the polynomial $z(x)$ where the terms x^i are replaced by x^{n-i}.

2.5 Computational cost

Computing the column parities costs $m - 1$ additions in GF(2) per column totalling to $(m - 1)n$ additions. Adding the effect to the matrix costs one addition per bit totalling to mn binary additions. So the total computational cost is $(2m - 1)n$ plus the computational cost of applying Z to the parity.

For circulant mixers with h the Hamming weight of $z(x)$, a straightforward parity-folding implementation would cost $(h - 1)n$, totalling to $(2m + h - 2)n$. Dividing by the total number of bits in the state, this results in a computational cost per bit of $2+(h-2)/m$. Remarkably, for parity-folding polynomials with two non-zero terms, the cost is exactly 2 additions per bit.

The number of additions gives a good idea of the circuit complexity in dedicated hardware and the number of binary XOR gates in bitslice-oriented software implementations. In the latter, the efficiency additionally depends on the way the state can be mapped onto CPU words and computing θ may involve additional (cyclic) shift operations. Section 6.5 shows the exact cost for a concrete example.

3 Propagation of linear masks

After having explained what CPMs are, this section discusses its properties related to linear cryptanalysis [Mat94]. We first provide a brief overview of linear propagation for generic iterated permutations, before discussing CPM-specific details.

3.1 Linear propagation in iterated permutations

Linear cryptanalysis (LC) exploits large correlations across a cryptographic primitive and resistance against it is one of the main criteria of modern cryptographic design. It can be described in different ways and we adopt the notation and formalism used in [DR02].

In LC we consider a sum (in GF(2)) of bits (usually called *parities*) at the output of a mapping and try to find sums of bits at the input of the mapping that have a high correlation with the sum at the output. Exactly which bits are included in the sums is described with *masks*. Masks have the same dimensions and shape as the state and indicate the bits included in a sum by having a 1 in the corresponding positions and 0 in all other positions. While the correlations are between sums of input bits and sums of output bits, we will indicate these by the term mask to make the descriptions more readable. Masks play a role in LC similar to that of *differences* in differential cryptanalysis (DC).

In iterated permutations or block ciphers, a correlation between an output mask v and an input mask u can be split into a number of *linear trails*. The value of the correlation is the sum of the *correlation contributions* of these individual trails. Note that correlations and correlation contributions have a sign, so there can be destructive interference. A linear trail Q over an n-round permutation (or block cipher) consists of a sequence of $n + 1$ masks: a mask at the input of each round q_i and a mask q_n at the output of the last round. A pair of consecutive masks q_i, q_{i+1} has a certain correlation over round i, denoted

as $C(q_i, q_{i+1})$. The correlation contribution $C(Q)$ of a linear trail Q is simply the product of the correlations over all its rounds: $\prod_{i=0}^{n-1} C(q_i, q_{i+1})$.

The value of an input-output correlation $C(u, v)$ is given by:

$$C(u, v) = \sum_{Q \text{ with } q_0=u, q_n=v} C(Q) \,.$$

Large input-output correlations can hence occur if there are linear trails with large correlation contributions (e.g., in the block cipher DES [Mat94]) but it can in principle also occur when there is systematic clustering of huge numbers of trails with very small correlation contribution. (An artificial example is a permutation P that consists of a permutation P' followed by its inverse P'^{-1}. The permutation P is the identity and hence has many large input-output correlations but no high-correlation linear trails if P' has none.)

Modern ciphers should be designed taking LC into account and hence should not have linear trails with high correlation contributions. A way to achieve this with a relatively small number of rounds is called the *wide trail strategy* [DR02], underlying many modern designs including AES and Keccak. In this strategy, the mixing layer in the round function plays an important role. To study the correlation contribution of linear trails, we need to study correlations over the round function. The round function typically consists of a non-linear layer and a linear layer. The description of the correlation properties of non-linear S-box layers can be efficiently computed using the Walsh-Hadamard transform. Linear layers have the property that every output mask is correlated to exactly one input mask and that the correlation is one. In other words, there is a deterministic function that maps masks v at the output of a linear layer to masks u at its input. In the following subsection, we will derive an expression for this function $u = f(v)$.

3.2 Mask propagation in column parity mixers

We want to express this function in our matrix notation. As an intermediate step, we first need a way to express a sum of specific bits of a matrix, for which we use the trace function.

The trace function has a number of interesting properties that we will exploit in our derivation:

- Linearity: $\mathrm{tr}(A + B) = \mathrm{tr}(A) + \mathrm{tr}(B)$.
- Transpose-invariance: $\mathrm{tr}(A^\mathsf{T}) = \mathrm{tr}(A)$.
- Commute-invariance: for A and B with compatible dimensions and AB a square matrix, it is easy to show that $\mathrm{tr}(AB) = \mathrm{tr}(BA)$ [LLM16].
- Multiplication-cyclic invariance: for any sequence of matrices $A, B, C, \ldots, Z$ with compatible dimensions and with $AB \cdots Z$ a square matrix: $\mathrm{tr}(AB \cdots Z) = \mathrm{tr}(BC \cdots ZA) = \mathrm{tr}(CD \cdots ZAB) = \ldots$.

A mask V has the same dimensions as the matrix A and it specifies the sum (in GF(2)) of the elements of A in positions where V has a one, i.e., $\sum_{i,j} V_{i,j} A_{i,j}$. This cannot be expressed as a matrix multiplication, but we can express it with the trace function.

Lemma 2. *The sum of the elements of A selected by the mask V can be expressed as* $\mathrm{tr}(V^\mathsf{T} A)$ *.*

Proof. Let $B = V^\mathsf{T} A$, then $B_{j,j} = \sum_i V_{i,j} A_{i,j}$. As $\mathrm{tr}(B) = \sum_j B_{j,j}$, it follows that $\mathrm{tr}(V^\mathsf{T} A) = \sum_{i,j} V_{i,j} A_{i,j}$. □

Lemma 3. *A sum of bits at the input of θ defined by a mask U equals a sum of bits at the output of θ defined by mask V if and only if*

$$U = V + \mathbf{1}_m^m V Z^\mathsf{T} .$$

Proof. Let B be the image of A through the column parity matrix. Then given $\mathrm{tr}(U^\mathsf{T} A) = \mathrm{tr}(V^\mathsf{T} B)$, we want to derive the relation between U and V. In particular, given the mask V at the output of θ, we can compute the mask U at its input by filling in the expression for B:

$$\begin{aligned}\mathrm{tr}(U^\mathsf{T} A) &= \mathrm{tr}(V^\mathsf{T}(A + \mathbf{1}_m^m A Z)) \\ &= \mathrm{tr}(V^\mathsf{T} A + V^\mathsf{T} \mathbf{1}_m^m A Z) \\ &= \mathrm{tr}(V^\mathsf{T} A) + \mathrm{tr}(V^\mathsf{T} \mathbf{1}_m^m A Z) .\end{aligned}$$

For the rightmost term, using multiplication-cyclic invariance gives $\mathrm{tr}(V^\mathsf{T} \mathbf{1}_m^m A Z) = \mathrm{tr}(Z V^\mathsf{T} \mathbf{1}_m^m A)$. Moreover, using $Z V^\mathsf{T} \mathbf{1}_m^m = (\mathbf{1}_m^{m\mathsf{T}} V Z^\mathsf{T})^\mathsf{T}$ with $\mathbf{1}_m^{m\mathsf{T}} = \mathbf{1}_m^m$ yields

$$\begin{aligned}\mathrm{tr}(U^\mathsf{T} A) &= \mathrm{tr}(V^\mathsf{T} A) + \mathrm{tr}((\mathbf{1}_m^m V Z^\mathsf{T})^\mathsf{T} A) \\ &= \mathrm{tr}(V^\mathsf{T} A + (\mathbf{1}_m^m V Z^\mathsf{T})^\mathsf{T} A) \\ &= \mathrm{tr}((V + \mathbf{1}_m^m V Z^\mathsf{T})^\mathsf{T} A) .\end{aligned}$$

□

We call the mapping from V to U specified in Lemma 3 the transpose of θ and denote it as θ^T. The expression of Lemma 3 is essential when searching for linear trails, as reported on in Section 5.3.

4 Diffusion properties

The diffusion properties of an MDS matrix are typically summarized by its differential and linear branch numbers [DR02] that are both the same and equal to the dimension of the matrix plus 1. This dimension is also the number of elements a single-element difference propagates to. The study of the diffusion properties of an MDS matrix is largely independent of the other steps in the round function. For example, the proof of the fact that every 4-round trail in AES has 25 active S-boxes only requires from the MixColumns matrix that it is MDS.

Diffusion in column parity mixers is more subtle: their properties only become meaningful in the context of a design approach, where interaction with other steps of the round function must be taken into account. In this section we discuss some concepts that are shared by all column parity mixers.

4.1 The column parity kernel

The set of states A with column parity equal to zero form a vector space with dimension $n(m-1)$. Following [Ber+11b], we call this the *(column parity) kernel*. For states A in the kernel the parity is zero and consequently θ reduces to the identity mapping.

There are states in the kernel with Hamming weight 2, namely all states with a pair of active bits in the same column. Again following [Ber+11b], we call this an *orbital*. States in the kernel have an even number of active bits per column and as observed in [MDA17], they can be seen as a collection of orbitals. Due to the existence of single-orbital states, the (Hamming) branch number, both differentially and linearly, of every column parity mixer is at most 4. We prove that for all reasonable choices of m, n and Z, the branch number is 4.

Lemma 4. *An invertible CPM θ with $m \geq 2$ and where Z has no all-zero rows or columns, has differential and linear branch numbers 4, with the only exception of the case where $m = n = 2$ and $Z = \mathbf{I}$.*

Proof. The branch number is at most 4 as θ will map a single-orbital state to a single-orbital state.

Let us first look at the differential branch number. The differential branch number can only become smaller than 4 if there is a single-bit state that is mapped to a state with less than 3 bits by θ or θ^{-1}.

Let us first look at a single-bit state at the input of θ. A single-bit state A leads to a single-bit parity. Let the number of affected columns in the θ-effect be x. We know that $x \geq 1$ as Z has at least one 1 per row. If these affected columns overlap with the column with the single 1 in A, then the Hamming weight of $\theta(A)$ is $xm - 1$. If not, it is $xm + 1$. Let ν denote the sum of the Hamming weights of A and $\theta(A)$. It follows that $\nu = xm$ and $\nu = xm + 2$, respectively. We now distinguish between the cases where m is even and where m is odd.

- Even m: if there was no overlap, $\nu = xm + 2 \geq 4$. If there was overlap, $\nu = xm = 2$ only if $x = 1$ and $m = 2$: so a single affected column overlapping with the odd column. But this is exactly the case that is excluded in the lemma. As for even m we have $\theta^{-1} = \theta$, that case is proven too.
- Odd m: if $x = 1$, i.e., there is one affected column, then it cannot overlap. Namely, if that is the case, it implies a single 1 in the corresponding row that is on the main diagonal. But for θ to be invertible, $Z + \mathbf{I}$ must be invertible and if Z contains a row with a single 1 and that is on the main diagonal, that row is 0 in $Z + \mathbf{I}$ and hence it is not invertible. The same reasoning applies to θ^{-1} as $Z' + \mathbf{I}$ with Z' the parity-folding matrix of θ^{-1} must be invertible.

For the linear branch number it suffices to replace θ by θ^{T} and rows of Z become columns of Z. □

Example 2. The simplest possible CPM that satisfies the conditions of Lemma 4 operates on two rows and two columns and has

$$Z = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

If the 4 bits of the state are arranged in a 4-bit column vector, θ can be expressed as multiplying it with the following familiar matrix:

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}.$$

This matrix was first used in the block cipher MMB [DGV93] and is still used in many modern lightweight ciphers (sometimes modulo some row and/or column permutations), where it is often referred to as a near-MDS matrix. See for instance Minalpher [Sas+15], L_0 and L_3 in PRIDE [Alb+14], or Midori [Ban+15]. It has an implementation cost of 1.5 additions per bit. To see that the CPM with the given Z and the matrix are very similar, let

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

Then, using a CPM:

$$\theta(A) = A + \mathbf{1}_2^2 AZ = \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} a+b+d & a+b+c \\ b+c+d & a+c+d \end{bmatrix}.$$

Similarly:

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} a+b+d \\ a+b+c \\ b+c+d \\ a+c+d \end{bmatrix}.$$

We now consider the size of the column parity kernel by counting the number of states with a given number of orbitals. The number of single-orbital states, i.e., that have in total 4 active bits before and after θ, is given in following lemma.

Lemma 5. *The number of single-orbital states is* $n\binom{m}{2}$.

Proof. The orbital can be in one of n columns and the number of possible positions of the bits of the orbital in the column is $\binom{m}{2}$. □

If we divide the number of 2-bit states in the kernel by the state size, we obtain $\frac{m-1}{2}$. In this respect, it is advantageous for a given state size to have a small number of rows and a large number of columns.

We can similarly compute the number of states in the kernel with 2, 3, 4, etc. orbitals. The number of such states increases exponentially with the number of active bits. For example, we have the following lemma on the number of two-orbital states.

Lemma 6. *The number of two-orbital states is* $\binom{n}{2}\binom{m}{2}^2 + n\binom{m}{4}$ *where the second term disappears for* $m < 4$.

Proof. We can partition the set of states with two orbitals into two subsets: those where the orbitals are in different columns and those where they are in the same column. For the first set of states, one can position the 2 orbitals in the n columns in $\binom{n}{2}$ ways and each of the two orbitals can take $\binom{m}{2}$ positions. Their product accounts for the first term. For the second set of states, the 4 active bits can be in one of n columns and the number of possible positions among the m bits of that column is $\binom{m}{4}$. If $m < 4$, of course no 4 bits can be stuffed in a column and the second term disappears. □

4.2 Propagation of isolated bits

Often one tries to determine the minimum number of rounds such that each output bit depends on each input bit, typically called *full diffusion*. One way of measuring full diffusion is propagating single-bit input differences or a single-bit output mask through the rounds. In this context it is interesting to see how single-bit differences propagate.

A single-bit difference at the input of θ propagates to on average $1 + |Z|m$ bits, with $|Z|$ the Hamming weight of Z divided by its dimension. For the circulant case, this is exactly $1 + |z(x)|m$ with $|z(x)|$ the Hamming weight of the parity-folding polynomial, assuming its constant term is 0. The same holds for propagation of single-bit masks at the output of θ to the input. For a single-bit difference at the output (or single-bit mask at the input) of θ we have to distinguish between two cases. If m is even, the column parity mixer is an involution and the same properties hold as in the forward direction. For odd m, we have to consider the column parity mixer defined by $Z' = (Z + \mathbf{I})^{-1} + \mathbf{I}$. The matrix Z' may have much higher Hamming weight than Z, as is illustrated by θ in Keccak [Ber+11b] and even for large states, full diffusion in the backward direction can be immediate.

4.3 Comparison to other mixing layers

Mixing layers can be compared by their cost, where cost can mean the number of additions, but also the number of cycles on a specific CPU microarchitecture or other quantities.

Table 1: Comparison of mixing layers.

Cipher/permutation	Type	Dimensions	Additions per bit	Branch number
AES	MDS	4×4, $GF(2^8)$	3.03	5
Joltik [JNP15]	MDS	4×4, $GF(2^4)$	3	5
PHOTON [GPP11]	MDS	6×6, $GF(2^4)^*$	$5^\dagger$	7
Prøst [Kav+14]	MDS	16×16, $GF(2)$	$4.5^\dagger$	5
Midori [Ban+15]	Not MDS‡	4×4, $GF(2^4)$ or $GF(2^8)$	1.5	4
Minalpher [Sas+15]	Not MDS‡	4×4, $GF(2^4)$	1.5	4
Prince [Bor+12]	Not MDS	64×64, $GF(2)$	1.5	4
SKINNY [Bei+16]	Not MDS	4×4, $GF(2^4)$ or $GF(2^8)$	0.75	2
Keccak-f [Ber+11b]	CPM	$5 \times 5 \times w$,§ $GF(2)$	2	4
Circulant CPM	CPM	$m \times n$	$2 + \frac{h-2}{m}\|$	4

* Dimensions are for P_{100}, P_{144}, and P_{196}.
† Unknown whether it can be computed with less additions.
‡ Can also be considered to be a CPM, following Example 2.
§ Where $w \in \{8, 16, 32, 64\}$, depending on which variant of Keccak-f.
$\|$ Where h is the Hamming weight of $z(x)$. Additions/bit $\in [2 - 1/m, 2 + (n-2)/m]$.

Here we consider the number of binary additions or two-input XOR gates, as that is independent from CPUs or standard cell libraries.

Mixing layers should also be compared by their 'mixing power'. This is best quantified by bounds on linear and differential trails, but that requires to study a full permutation. We will do so in Section 6.3, but for the moment we restrict ourselves to only consider the mixing layer itself. This implies that we are limited to use the linear and differential branch numbers that should be high relative to the dimensions of the state.

Ever since SHARK [Rij+96] and AES [DR02], MDS matrices have been a common choice for a mixing layer as their branch numbers are optimal. The past few years have seen a lot of work on searching for more efficient MDS matrices. A noteworthy development is that of serialization of the mixing layer, first put forward by PHOTON [GPP11] and LED [Guo+11]. This decreases the hardware area that is required to implement matrix multiplication. Another line of work focussed more directly on searching MDS matrices that require less additions [BKL16; LS16; LW16; LW17].

While there has been a lot of improvement, it was always assumed that there was a fixed cost of $n(n-1)m$ additions [BKL16], derived from summing all the multiplication results. Recently, it has been shown that this lower bound does not really hold when global optimizations are taken into account [Kra+17]. In fact, there exist MDS matrices that can be implemented with less additions.

Other ciphers such as PRIDE [Alb+14], Midori [Ban+15], Minalpher [Sas+15], and SKINNY [Bei+16] have dropped the MDS requirement to achieve a better trade-off between performance and security.

Table 1 provides a comparison of mixing layers. Most cost numbers are computed using the results of [Kra+17]. We list the number of additions per bit, in order to normalize for the different dimensions. While more lightweight mixing layers exist, CPMs offer a good trade-off. It should be noted, however, that bounds over multiple rounds for a concrete scheme are much more meaningful than just a pair of branch numbers.

5 A general design strategy

In this section, we present a design strategy for the round function of nibble- or byte-oriented iterative block ciphers or permutations, viewed as a substitution-permutation network (SPN), where a CPM is embedded as its mixing layer. We also outline a search strategy for truncated linear and differential trails. In Section 6, we instantiate this design strategy with a specific permutation and give security bounds and implementation benchmarks.

5.1 Structure of the round function

We take the b-bit state to be a rectangle with m rows and n columns. The state consists of mn *cells* of size ℓ, so $b = \ell mn$. The cells are typically nibbles ($l = 4$) or bytes ($l = 8$) for software performance reasons, although other values are also possible. The cells can be seen as elements of $\mathrm{GF}(2^\ell)$.

The round function consists of four layers:

- A nonlinear S-box layer γ, that applies the same invertible S-box to every cell.
- A column parity mixer θ. Although it operates on elements of $\mathrm{GF}(2^\ell)$, the parity-folding matrix only contains 0 and 1.
- A transposition layer. This consists of two steps: one that permutes the rows called π and one that performs cyclic shift operations (rotations) on the rows, called ρ.
- The addition of round constants.

In the case of a block cipher, round keys can be added between the rounds that are derived from a cipher key (and a possible tweak in the case of tweakable block ciphers) by means of a key schedule. In this paper we consider the design of key schedules out of scope.

The roles of the different layers are the same as in the well-known wide trail strategy.

The role of the round constants is to remove invariants of the round function:

- Invariance of the round function with respect to simple transformations of the mappings of the state, such as rotations. Attacks relevant in this context are rotational cryptanalysis [KN10].
- Invariance of the round function across the different rounds. Relevant in this context are slide attacks [BW99].
- Existence of sets that are invariant under the round function. Relevant attacks are here invariant subspace attacks [Lea+11].

5.2 Outline of the steps in our design approach

We assume the designers are faced with the request to design a permutation or block cipher with some given width b, that must be efficient on some given set of platforms. They also know whether side-channel attacks are a concern leading to the requirement of constant-time code or the ability to mask efficiently, and whether the inverse of the mapping must be efficient.

The first step in the design process consists of choosing ℓ, m, and n with the constraint $\ell mn = b$. As we have showed in Section 4.1, choosing m large compared to n implies that there will be more states in the kernel of θ. Diffusion of isolated bits, however, benefits from choosing m large, as shown in Section 4.2.

Once ℓ is determined, one can design the γ S-box quasi-independently of the other components of the round function. Important metrics are its algebraic degree, the maximum input-output correlation, the maximum differential probability, but also its implementation cost. For the latter, one should take into account which are the main implementation targets, such as ASICs, bitsliced software, or software using table look-ups. This is not different from most design strategies.

The design of θ, π, and ρ can be performed in two phases, taking advantage of the kernel property of CPMs.

In a first phase, propagation is investigated of difference patterns and masks inside the kernel and that are *truncated* [Knu95]. The latter means that at cell level we do not consider the actual values but only whether the cell in the pattern is passive (all-zero) or active (non-zero). For such patterns, θ acts as the identity function, as we impose that they are in the kernel. Moreover, γ acts as the identity, because the S-box is invertible and hence a non-zero input difference leads to a non-zero output difference, and any sum of bits at the output is balanced and hence a non-zero mask at the output cannot propagate to an all-zero mask at the input. It follows that the propagation of such patterns is fully determined by the transposition steps π and ρ. As these steps just move cells around and do not mix them, this propagation is fully *deterministic*. Moreover, transpositions are their own transpose and hence masks and difference patterns propagate in the same way. Hence one can take a pattern A_0 in the kernel and propagate it forward through $\rho \circ \pi$ resulting in A_1. If the resulting pattern can be in the kernel, one can propagate one more round and so on. A pattern that has at least one column with a single active cell cannot be in the kernel, all other patterns can. In a column with two active cells, these must have the same value (2^ℓ possibilities). In a column with three active cells, if two of the cells have value x and y, the third has value $x + y$ and hence it follows that $x \neq y$ ($2^\ell(2^\ell - 1)$ possibilities). As the number of active cells grows, more cases can be distinguished. In any case, a pattern can be propagated through the rounds until it has a column with a single active cell.

This first phase can be used to select the rotation distances of ρ and the π permutation. This allows to reduce all possible values to a reasonably-sized set of candidate parameter values. For instance, if ρ_0 and ρ_1 are two rotation distances and $\rho_0 = \rho_1$, it makes it easier to remain longer in the kernel: a single orbital in some column will, after $\rho \circ \pi$, still be a single orbital. Hence this should be avoided. Similarly, when $\rho_i \neq \rho_j$ but $\rho_i = c\rho_j$ for all $i, j \in [0, m-1], i \neq j$ where c is some integer, there exist patterns with multiple orbitals that remain in the kernel after applying the transposition layer.

The obtained bounds of trails in the kernel can then be used in the second phase to decide on the amount of work one wants to do in the parity folding matrix. A further selection of Z candidates can be made by means of a full trail search and diffusion criteria such as the number of rounds to reach full diffusion or to meet the (strict) avalanche criterion.

Finally, the round constants can be designed quasi-independently from the other parts of the round function, such that various invariant attacks are not a concern [Bei+17].

5.3 Searching linear and differential trails

Given a candidate design that follows the outlined approach, its resistance against truncated linear and differential cryptanalysis should be quantified by scanning a large search space of trails. For most ciphers nowadays, this search is performed using Mixed Integer Linear Programming (MILP). However, recently a very different technique was used to find trails for Keccak, which appeared to be quite promising [MDA17]. There, a unique decomposition of a difference pattern is defined, allowing a tree-based generation of 'two-round trail cores', where many branches can be pruned and symmetry can be used to skip over large parts in the search space. We will explain this in more detail.

We use techniques that are based on that work to find trails for our design. However, to make it work for our design, we have to make a number of modifications. In this section we will also highlight the most interesting differences, that are due to the different dimensions of the states and the choice of a bit-oriented design versus a design with an almost arbitrary cell size ℓ.

5.3.1 Trails

A trail Q is a sequence over r rounds. In the case of linear cryptanalysis, it is a sequence of masks. In the case of differential cryptanalysis, it is a sequence of difference patterns. This section mostly uses the neutral term *state pattern*. A cell in a state pattern is called *passive* when it is 0 and *active* when it is any non-zero value.

We barely distinguish between linear and differential cryptanalysis, because we have shown in Section 3 that masks propagate through θ^{T} in the same way that differences propagate through θ. Furthermore, the γ step acts as the identity function as it does not alter the activity of cells: what is active remains active and what is passive remains passive. The same holds for the addition of round constants. Linear masks also propagate the same as differences through row permutations and rotations. This means that the only difference between searching truncated differential and truncated linear trails is that in differential trails, forward propagation through θ is governed by Z, and in linear trails backward propagation through θ is governed by Z^{T}.

Because we only consider truncated trails, we only look at cells and not at individual bits. The *weight* W of a trail Q is defined as $W = \sum_{i=0}^{r-1} |q_i|$, where $|q_i|$ denotes the number of active cells in q_i.

5.3.2 Trivial trails in the kernel

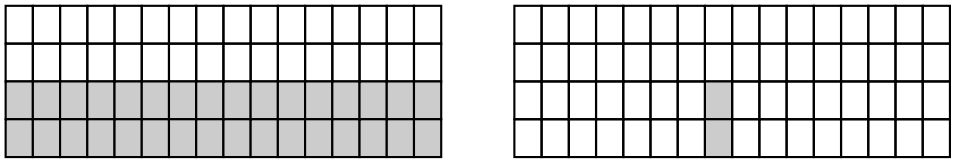
The possibility to remain in the kernel leads to some trivial trails. One example of a starting state pattern is given in Figure 1a, for dimensions $m = 4$ and $n = 16$, with two full rows of active cells. In general, for all dimensions, if the column parity is the all-0 vector, θ will just be the identity function. Furthermore, a transposition layer consisting of row permutations and rotations will be unable to move the pattern out of the kernel. This leads to a trivial trail over an arbitrary number of rounds. However, this trail has a high weight of $W = 2nr$ active cells.

When Z is circulant, when there is only one full row of active cells and when the column parity is therefore the all-1 vector, θ is the identity function when the number of affected columns for a single active cell is even. Then there is a trail of weight $W = nr$ active cells. Recall that a column is affected when its bit in the θ-effect is one. When the number of affected columns is odd, θ will be the complement function. The trail (with even length) will then have weight $W = n\frac{r}{2} + (m-1)n\frac{r}{2} = mnr/2$ active cells.

For reasonable dimensions, n is high enough such that these do not pose a problem. Other state patterns with less active cells might be able to stay in the kernel for one or two rounds, but the transposition layer should always move them out of the kernel soon. An example is shown in Figure 1b. This means that the property that a CPM has a kernel will not pose a serious threat.

5.3.3 Generating two-round trails

In an r-round trail with weight W the average weight per round differential is $\frac{W}{r}$ and hence it always contains a round differential with weight $L \leq \left\lfloor \frac{W}{r} \right\rfloor$ active cells. When one wants to find all r-round trails up to weight W, one can therefore generate round differentials up to weight L and extend all of them forward and backward $r-1$ rounds. However, it was observed in [MDA17] that there are less two-round trail cores with weight



(a) Remains in the kernel indefinitely. **(b)** Should quickly move out of the kernel.

Figure 1: Examples of state patterns in the kernel.

below $2L$ than differentials of weight below L. We therefore generate two-round trail cores up to weight $2L + 1$ active cells and extend them $r - 2$ rounds forward and backward.

Similarly to [MDA17], generating the two-round trails is modelled as the traversal of a tree, where each node represents a pair of state patterns at the input and output of θ. The root is formed by the empty state patterns. The idea is that the weight monotonically increases when visiting a node's children. Branches where it is known that all descendants will have a weight higher than $2L$ can then be pruned.

As with Keccak-f, we can uniquely decompose every state pattern into a *parity-bare state* and a list of *free orbitals*. A parity-bare state is a state pattern that does not have free orbitals. A free orbital is a pair of two active cells in an unaffected column, as was already mentioned in Section 4.1. Removing a free orbital from a state pattern also removes two active cells after θ, so adding a free orbital to a state pattern will always increase the weight of a trail by 4 active cells. This decomposition reduces the problem to that of generating all parity-bare states in a weight-increasing manner, because we can construct all states up to some weight $2L + 1$ by generating all parity-bare states up to that weight and adding free orbitals to those where there is still budget.

A parity-bare state consists of a list of odd columns, i.e., columns with non-zero parity. Initially, at the root node, this list is empty. Then one odd column is added to the list. This means that the children of the root node are all the state patterns with a single odd column. At every level of the tree, new columns are added.

On top of the tree of parity-bare states, all the nodes also form the root of their own tree, now maintaining a list of free orbitals, such that at each level free orbitals are added up to the limit weight. Together, this guarantees that all two-round trails will be found.

Moreover, the rotation-symmetric nature of the state in the horizontal direction and of all operations allow us to prune many more branches of the trees. There is only need to consider *canonical* state patterns, where a state pattern is defined to be canonical if and only if it is minimal under all horizontal translations given a total ordering over state patterns. In this case, the total ordering is just the lexicographical ordering on $[z, y]$, where z is the horizontal axis and y the vertical axis, to keep the notation similar to [MDA17]. The lower left has coordinates $(z, y) = (0, 0)$.

5.3.4 Propagation of cells and trail extension

After having generated two-round trail cores, they need to be extended forwards and backwards by $r - 2$ rounds. With Keccak-f, extension implies that one needs to check which patterns are compatible trough the nonlinear χ step. Given a non-zero state pattern at the end of a trail, there are multiple possibilities after χ and each case needs to be explored. This branching behaviour causes an exponential increase of the search space when extending to more rounds.

However, in our case we are considering truncated trails and γ has no effect on state patterns. This is very different from the situation in [MDA17]. Instead, this branching behaviour is now caused by θ.

To see this, consider a state pattern before and after θ. Assume a column has ≥ 2

active cells before θ. To apply θ, we start by computing the parity of that column. This means that we sum over non-zero differences (in the case of differential cryptanalysis). It is possible that this results in a non-zero difference (i.e. the column is odd), but it is also possible that the non-zero differences cancel each other out and the column becomes even. Both cases should be explored and this causes the search space to branch.

There is another source of branching. We distinguish between four types of columns: even affected, even unaffected, odd affected, and odd unaffected.

When a column of the state pattern is unaffected, θ will not add anything to the cells in that column. Therefore, cells that were active before θ will remain active after θ. Passive cells will also remain passive. We therefore only need to consider affected columns. In affected columns, every cell that is passive before θ becomes active after θ, as a non-zero difference is added to a difference of zero in the case of differential trails. However, if a cell is active cell before θ, it is unknown whether it will remain active or will cancel out after θ, as some non-zero difference is added to some non-zero difference. A search for truncated trails would need to consider both cases for all active cells in affected columns.

However, one can do slightly better. How an active cell propagates depends on the number of active cells in that column. As an example, let $\#_x$ denote the number of active cells in column x (and let $m \geq 4$). We will make a case distinction on whether x is even or odd, and on $\#_x$, up to $\#_x = 4$.

$\#_x = 0$, x **even.** Nothing can cancel out. Of course, all cells become active (with the same difference value) after θ.

$\#_x = 0$, x **odd.** Impossible.

$\#_x = 1$, x **even.** Impossible.

$\#_x = 1$, x **odd.** That active cell may cancel out or not after θ.

$\#_x = 2$, x **even.** This can only happen when the two active cells have the same difference value. When x is affected, the two active cells either both cancel out after θ, or none of them, but never only one.

$\#_x = 2$, x **odd.** This can only happen when the two active cells have a different difference value. Therefore, either no active cells cancel out after θ, or one of them, but never both.

$\#_x = 3$, x **even.** This can only happen when all three active cells are different. At most one of the active cells cancels out after θ.

$\#_x = 3$, x **odd.** Now there are more possibilities. The active cells can be all the same, all different, or can consist of one pair and one other value. This implies that any in $\{0, 1, 2, 3\}$ active cells cancel out after θ.

$\#_x = 4$, x **even.** The active cells are all the same, all different, or consist of two pairs. So any in $\{0, 1, 2, 4\}$ active cells cancel out after θ.

$\#_x = 4$, x **odd.** All combinations are possible, except that all active cells have the same difference value. Therefore any in $\{0, 1, 2, 3\}$ active cells may cancel out after θ.

This can be extended further to higher values for $\#_x$, but the relative amount of cases that can be skipped will then decrease. It is clear that the search space of differential trails branches heavily with a CPM.

We wrote dedicated software to traverse the search space of truncated linear and differential trails for designs following our approach. The software is freely available at https://github.com/Ko-/cpm.

6 The Mixifer permutation

6.1 Design goals

To show an instance of this design strategy, in this section we introduce a concrete permutation and study its security and efficiency. We aim for a suitable permutation for lightweight applications, such as in the Internet of Things. In this case, with 'lightweight' we mean that it should be fast in constant-time code for microcontrollers such as the ARM Cortex-M series and that it should require little area in ASICs.

This could be used in a simple (single-key) Even-Mansour construction [EM93] to build a block cipher or in a construction such as XPX [Men16] to build a tweakable block cipher. Based on this, one could even build other primitives such as authenticated encryption. Depending on the mode, we may or may not need the inverse permutation. To make the permutation more flexible, we therefore require that the inverse also needs to be efficient.

For efficient full-width processing in software, we would like a state size that is a power of two. A potential problem with a state size b of 128 bits is that this gives a birthday bound of only 64 bits, which may or may not be an issue depending on the mode in which this permutation is to be used. Again, we design this permutation to be useful in many scenarios, which is why we choose our state size to be equal to the next power of two, $b = 256$ bits. This gives a birthday bound of 128 bits, which should be more than enough.

To achieve fast constant-time code, we aim for a round function that can be efficiently implemented with bitwise Boolean instructions and cyclic shifts. This has implications for the way we arrange the bits of the cells in CPU words.

6.2 The construction

First we have to select ℓ, m and n. To have an efficient inverse of θ, we choose n to be even, such that θ becomes an involution. For an efficient bitsliced implementation of γ, we choose $\ell = 4$. Now both 8 rows and 8 columns, and 4 rows and 16 columns are viable options. We set $m = 4$ and $n = 16$, based on the trivial in-kernel trails discussed in Section 5.3.2.

For γ, applying a 4-bit S-box to every nibble separately can quickly become expensive. We make sure that, in software, the S-box can be applied efficiently on a full row, without using lookup tables or other operations that might lead to timing attacks. We suggest a bitsliced state representation and explore one such possibility in Section 6.5. This works when the S-box is rotation-symmetric.

An ℓ-bit S-box $S : \mathrm{GF}(2^\ell) \to \mathrm{GF}(2^\ell)$ is *rotation-symmetric* if and only if $S(a \ggg d) = S(a) \ggg d$ for all $a \in \mathrm{GF}(2^\ell)$, $d \in [0, \ell - 1]$, where $\ggg$ denotes bitwise rotation. Properties of rotation-symmetric S-boxes (RSSBs) have been explored in, e.g., [RBG08] and [Kav12]. Rotation-symmetric S-boxes are determined by a single coordinate function. There are therefore only 2^{2^4} 4-bit rotation-symmetric functions, which makes them efficiently enumerable. We search exhaustively through these candidate S-boxes to find an example with nice cryptographic properties and that also has an efficient implementation. Out of the 65536 functions, 1536 are invertible. Of course, many are linear and/or affine equivalent to each other [LP07]. Out of these 1536 candidates, 512 are 'optimal' S-boxes, as defined in [LP07]. This means that they have a maximum differential probability of 1/4 and a maximum input-output correlation of 1/2. All of these have 3 as their algebraic degree. We then selected the S-box based on the number of binary Boolean operations that are required to compute it and its inverse. It can be defined in algebraic normal form by the coordinate function $b_0 = a_1 + a_2 + a_0a_2 + a_1a_2 + a_1a_2a_3$. More details and representations can be found in Appendix A.

For the first phase of selecting θ, π, and ρ, we consider trails in the kernel of θ. Because of all the symmetry, we can arbitrarily set one rotation distance to zero, shaving off a few

more operations. After some experiments, we set π to be the permutation that rotates all rows down. ρ rotates rows nibble-wise to the right by the distances 14, 3, 10, and 0, respectively, from top to bottom. This yielded a best trail with a weight of 52 active cells over 4 rounds. For implementation efficiency, we also make ρ compute an 'intra-nibble' rotation to the left for all nibbles that are wrapped to the other side. In Section 6.5 it will become clear why this 'extra' work is actually more efficient and not less. Note that this does not affect our truncated trail search.

For the second phase, the parity-folding matrix Z should be chosen such that the truncated trail search yields bounds that are not much better than the in-kernel trails. We end up with a circulant matrix where a column can affect three other columns. Over 4 rounds, the best differential trail then has a weight of 46 active cells

The steps of the round function have high symmetry. All four γ, θ, π, and ρ are invariant to rotation along the rows and rotation of bits within the cells. Moreover, γ and θ are even invariant in rotation along the columns. We therefore add round constants, that should achieve the following goals:

1. Applying a few rounds to a state that has some symmetry $A \oplus (A \ggg d) = 0$ shall result in a state B where $B \oplus (B \ggg d)$ has high Hamming weight, for all values of d.
2. Applying a few rounds to two states A and A^* with $A' = A \oplus A^*$ and $A' \oplus (A' \ggg d) = 0$ shall result in a difference B' where $B' \oplus (B' \ggg d)$ has high Hamming weight, for all values of d.
3. There shall be no invariant subspaces in the linear part of the round function, including the addition of the round constant.
4. Round constant addition shall be cheap.

The minimum cost of round constant addition is a single bitwise addition per round. Therefore, we have round constants that are non-zero in a single word. We choose the word that contains the nibbles of the top row in even positions. We use a simple scheme to achieve asymmetry: all round constants are obtained by performing a (non-cyclic) shift of a single master round constant, where the shift offset is simply the round index. We believe criteria 1 and 2 listed above are satisfied for the following reasons. First, the round constants are outside the kernel and hence their influence will spread very quickly. Second, they do not have symmetry along the rows or within the nibbles.

As for criterion 3, we investigated the algebraic properties of the linear part of the round function, as explained in [Bei+17]. It can be seen as 4 identical mappings, each operating on the bits in a specific position of the nibbles. So each one operates on a matrix of 4 rows and 16 columns. The characteristic polynomial of this mapping is $1 + x^{64} = (1 + x)^{64}$. This suggests that it has non-trivial invariant subspaces of dimension 1, 2, 4, 8, 16, 32. These are easy to find, knowing the symmetry. The ones of dimension above 2 simply consist of states that have periodic rows. The subspace of dimension 32 is the set of all states with period 8, dimension 16 with period 4, dimension 8 with period 2 and dimension 4 with period 1 (rows are all-1 or all-0). The subspace with dimension 2 are the 4 states with an even number of all-1 states and an even number of all-0 rows. Finally, the subspace with dimension 1 are the all-0 and the all-1 state.

The 32 active bits in the round constants are spread evenly over these 4 mappings. It would be problematic if the differences between the round constants were all periodic. We made sure that this is not the case.

The master round constant has the following value expressed as a bit sequence: `11000110111010100001001011001111`, starting with the least significant bit. This has been generated with the 5-bit LFSR with feedback polynomial $1 + x^3 + x^5$. While there are linear recurrences among bits due to a short LFSR, this has no relation with the symmetry in the steps of the round function and we therefore believe that this is not a problem.

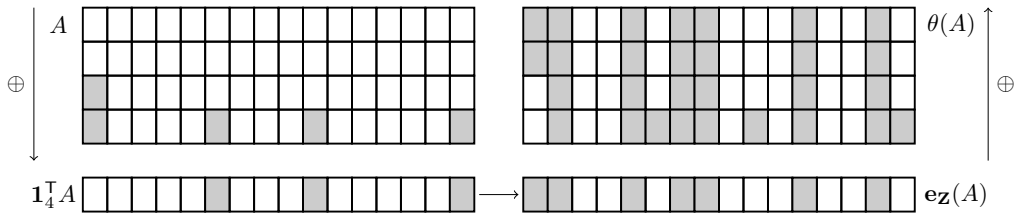


Figure 2: The diffusion layer θ.

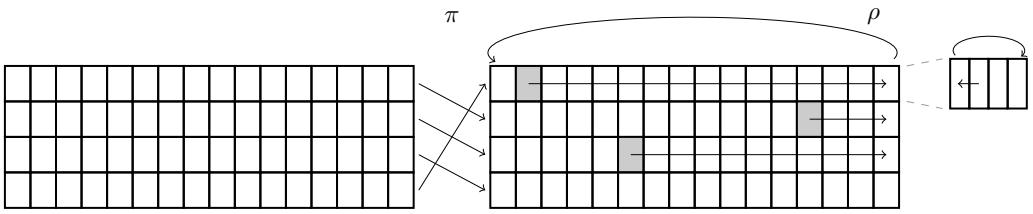


Figure 3: The transposition layer $\rho \circ \pi$.

The round constant for round i is this sequence shifted to the left by i bits.

Summary. To summarize this and to make this more precise, we introduce a 256-bit iterated permutation called Mixifer. Mixifer consists of 16 rounds. The 256-bit state is arranged as $m = 4$ rows with $n = 16$ columns of 4-bit nibbles. Every round consists of applying $\iota \circ \rho \circ \pi \circ \theta \circ \gamma$.

- γ is a non-linear mapping, the S-box. It is a 4-bit S-box that is applied to each nibble. We choose the rotational symmetric S-box defined by the following equation (over $GF(2)$) for the first coordinate: $b_0 = a_1 + a_2 + a_0a_2 + a_1a_2 + a_1a_2a_3$. Appendix A provides other descriptions of this S-box and its inverse.

- θ is our column parity mixer, defined by dimensions $m = 4$, $n = 16$, and by the following circulant parity-folding matrix Z:

$$Z = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}.$$

  Alternatively, the parity-folding polynomial is $z(x) = x + x^2 + x^5$.

- π permutes the rows as follows: $0 \mapsto 1, 1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 0$, where $\{0, 1, 2, 3\}$ are row indices, starting from the top.

- ρ rotates the 4 rows nibble-wise to the right by the distances 14, 3, 10, and 0, respectively, from top to bottom. Additionally, all the nibbles that are wrapped to the other side are rotated to the left by 1 bit within that nibble.

- ι adds a round constant into the even cells of the top row, starting to count at 0, as highlighted in Figure 4. In round i this is `0xF3485763` shifted to the right by i, again starting to count at 0.

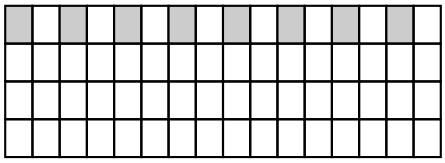


Figure 4: Locations where ι adds the round constant.

6.3 Evaluation

6.3.1 Avalanche effect

We first study the diffusion of our CPM by considering the avalanche effect for Mixifer. Good diffusion implies that a single bit will quickly affect as many other bits as possible. Avalanche testing is a quick and easy test to quantify how quickly diffusion takes place.

For each of the 256 bits, we generate 10000 random states. A round-reduced Mixifer is then applied to that state and to that state where one bit is flipped. Each time, we look at the differences between the outputs. We consider the amount of bits that are flipped, but also in how many different cells bits are flipped. It can be seen in Table 2 that a single bit flip in a random state has very rapid diffusion. Already after 3 rounds, each bit of the output is flipped with 50.0% probability. This is also known as the strict avalanche criterion.

Table 2: Avalanche test flipping a single bit.

Rounds	1	2	3
Average bit flip probability	10.2%	47.1%	50.0%
Average cell change probability	20.2%	88.83%	93.8%
Worst-case bits flipped	13	44	89
Worst-case cells changed	13	32	48

Another way to test the avalanche effect is by starting with a state where only a single bit is set, and to measure how many rounds it takes before every bit in the state is touched. Typically, operations can be changed to `OR`s and it can be checked when all bits in the state are set. The results are listed in Table 3. After 5 rounds, 'full' diffusion is achieved from every starting bit.

Table 3: Avalanche test starting with single bit set.

Rounds	1	2	3	4	5
Worst-case bits set	71	196	243	255	256
Worst-case cells set	25	58	63	64	64

6.3.2 Linear and differential cryptanalysis

A strategy for searching truncated linear and differential trails has been outlined in Section 5.3. Here, we only summarize the trails and bounds that we computed for Mixifer.

First it should be verified that the kernel of the CPM does not pose a serious concern and that it is infeasible to stay in the kernel for many consecutive rounds, except for trivial trails where two full rows are active, as was discussed in Section 5.3.2. We generated trails where the state pattern remains in the kernel until the last round. The number of active

cells per state pattern in the trail is therefore constant. Table 4 summarizes the bounds, while Appendix C contains the actual trails.

Table 4: Minimum weight for trails that remain in the kernel until the last round.

Number of rounds r	2	3	4
Minimum W active cells	4	18	52

Outside of the kernel, we first use our software to generate two-round trails. Figure 5 shows the number of two-round differential trails that need to be considered and their weights.

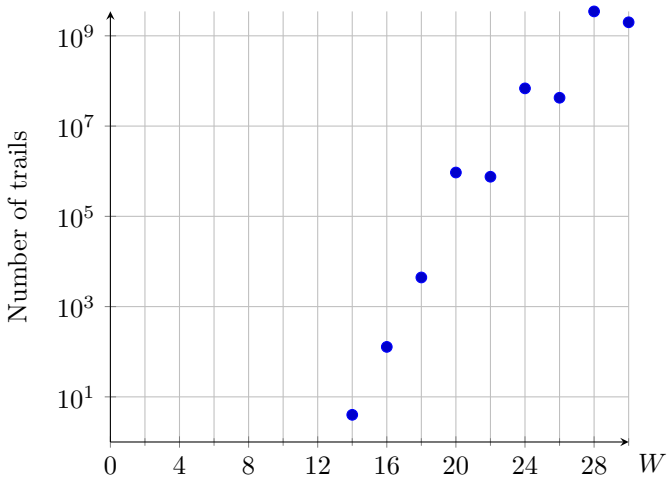


Figure 5: Number of active cells of two-round differential trails.

After extending these two-round trails, we find the minimum weight trails for three and four rounds, both for linear and for differential trails. For three rounds, both the best linear and the best differential trail have a weight of 27 active cells. Table 5 shows the overall minimum weights (combing trails in the kernel and outside the kernel), while the trails themselves can be found in Appendix C. It can be seen that after four rounds, the trails in the kernel are no longer the minimum weight trails. We expect the weight to again increase significantly for subsequent rounds, but we were not able to cover the full search space anymore.

Table 5: Minimum weight in number of active cells for trails.

Number of rounds r	1	2	3	4
Minimum W differential	1	4	18	46
Minimum W linear	1	4	18	40

6.3.3 Clustering of differential trails

So far we have considered truncated trails. However, as the round function is cell-oriented, there may be significant cases of trail clustering. In particular, there may be multi-round differentials with many differential trails in the same truncated differential trail.

For two-round AES, this was investigated in [DR06] and it turned out that the two-round differential over two rounds has a maximum differential probability (DP) of 13.25×2^{-32},

while the best two-round trail has DP 4×2^{-32}. The differential is the result of 75 trails, all in the same truncated trail. These investigations were done on an interesting sub-structure of AES, the so-called *AES super-box*. This structure is a permutation operating on a 4-byte array and consists of the SubBytes S-box layer, MixColumns, round key addition, and again a SubBytes S-box layer, all restricted to a 4-byte array. The best differentials over that structure have in total 5 active S-boxes over the two SubBytes layers and those are the ones that are investigated.

For Mixifer, we analyze an analogous case, namely the best differential trails over two rounds. Those are the ones with two active cells that form an orbital in θ of the first round. So we have two nibbles in the same column with difference values d_0 and d_1 at the input of γ, that maps them both to the same difference d_Δ. The subsequent θ acts as the identity and ρ and π move them to different positions before they arrive at the next γ layer. There, the two active nibbles map to output differences d_2 and d_3. The remainder of the second round does not modify the differential probability. So the super-box structure we consider consists of two γ S-boxes, followed by two γ S-boxes. There is no mapping in between, only the requirement that in the intermediate difference the two active cells are equal. So the trails look like this: $(d_0, d_1) \xrightarrow{\gamma} (d_\Delta, d_\Delta) \xrightarrow{\gamma} (d_2, d_3)$. This means that their differential probability is $\mathrm{DP}(d_0, d_\Delta)\mathrm{DP}(d_1, d_\Delta)\mathrm{DP}(d_\Delta, d_2)\mathrm{DP}(d_\Delta, d_3)$. They contribute to the super-box differential in the following way:

$$\mathrm{DP}((d_0, d_1) \to (d_2, d_3)) = \sum_{d_\Delta} \mathrm{DP}(d_0, d_\Delta)\mathrm{DP}(d_1, d_\Delta)\mathrm{DP}(d_\Delta, d_2)\mathrm{DP}(d_\Delta, d_3)\,.$$

A single trail can have DP at most 2^{-8}, as the maximum DP of our S-box is 2^{-2}. We have exhaustively checked for all 2^{16} possible combinations of (d_0, d_1, d_2, d_3), and the differential with the highest DP turned out to be the one where all d_i are all-1 (so `F` in hexadecimal). It has DP $2^{-8} + 6 \times 2^{-12}$ that is the contribution of 7 trails, one dominant with DP 2^{-8} and 6 with DP each 2^{-12}. This gives a DP for the differential that is only a factor 1.375 higher than that of its dominating trail.

As a preliminary conclusion, it appears that the effect of clustering appears to be smaller than for AES.

6.3.4 Impossible differential cryptanalysis

It is well-known that every permutation has many impossible differentials [BBS05]. For any input difference, there are exactly 2^{b-1} pairs and hence there can be only 2^{b-1} output differences. However, to the best of our knowledge no attacks have been reported that exploit that aspect. Impossible differential attacks exploit large classes of impossible differentials, i.e. differentials with zero probability. Here we will discuss whether this can be applied to Mixifer.

Let us consider a single-cell difference at the input of a round (called *first round*):

- First round: θ maps the single active cell to a difference with three additional affected columns and the subsequent ρ moves the active cells to different columns.
- Second round: γ transforms the difference values of the 13 active cells to possibly different values. The subsequent θ adds affected columns. We found that at this stage, *all* columns can be affected. The subsequent ρ and π steps just move the active cells around.
- Third round: γ transforms the difference values of the active cells to possibly different values. Within each column, all possibilities can occur: in the kernel or out of the kernel, all four the same 4, three the same $3+1$, $2+2$, $2+1+1$ and all four different $1+1+1+1$. The subsequent θ computes column parities over all columns and adds

three affected columns for each one. This leads to a state where each cell can be active or passive, with the exception of an all-passive state. This remains to be the case after ρ and π.

So a difference with a single active cell may lead to any (non-zero) truncated difference after three rounds. If we consider fully specified patterns taking into account the cell difference values, we have verified that, when we apply a difference a to the serial composition of two S-boxes with the addition of an offset in between, that this may lead to all possible non-zero differences for any non-zero value a. So, even though for any stage of the computation there will certainly exist (non-truncated) difference patterns that cannot occur, after χ of the fourth round, it will be very hard to exploit them.

We treated the case of a difference pattern with a single active cell. If there is more than one active cell, one may try to slow down diffusion by having the pattern at the input of θ of the first round in the kernel. In that case, one would apply a pattern at the input consisting of one or more orbitals. However, after γ of the first round, the difference values of the active cells belonging to the same orbital depend on the absolute value of the state, and hence they are only equal in a subset of all cases. So for the other cases that may occur and are hence not impossible, there are actually more active cells present after θ and hence diffusion is even faster.

We see that Mixifer has no exploitable impossible differentials that start from a single-cell difference over more than 3 rounds. This is actually as good as AES despite the larger number of cells, and we attribute it to the large average diffusion of the column parity mixer that we use.

6.3.5 Invariant attacks

Under invariant attacks, we consider rotational cryptanalysis [KN10], slide attacks [BW99], invariant subspace attacks [Lea+11], and nonlinear invariant attacks [TLS16]. One property that these attacks share, is that they can be defeated by appropriately choosing round constants [Bei+17].

We explicitly designed our round constants to fulfill all the relevant criteria, which is why we conclude that there are no exploitable invariant attacks against Mixifer.

6.4 The number of rounds

Recall that we aim for use cases such as XPX [Men16], where an adversary can apply difference patterns across the complete input and can do inverse permutation queries, and that we target a security level of 128 bits. Note that this is different than for block ciphers where by default a security level equal to the block length is claimed, leading to the absurd situation that attacks requiring an adversary to query almost the full codebook can *break* a cipher, at least from an academic perspective. Our security claim is compatible with the fact that most modes do not achieve security above the birthday bound, that is at a comfortable 128 bits for our permutation width.

Based on our investigations, we fix the number of rounds to 16 and believe this gives a comfortable safety margin for the following reasons:

- The best differential trails over 4 rounds have DP 2^{-92}, so the best ones over 12 rounds have an approximated DP of 2^{-276}. In our best 2-round differential trail, the DP is 2^{-8}, while the best 2-round differential has DP $1.375 \times 2^{-8} \approx 2^{-7.54}$. So clustering may lead to some loss, but even dividing the exponent of the best trail DP by two would give 2^{-138}, an unexploitable value.

- For linear trails we have a maximum linear probability (square of the correlation) over 4 rounds of 2^{-80}, resulting in 2^{-240} over 12 rounds. This is slightly less comfortable

than the case of differential trails, but it is still very large, even if a huge degeneration due to clustering would occur.

- The number of rounds for which structural attacks, such as integral cryptanalysis and impossible differentials, still work, is strongly correlated with the number of rounds that it takes to achieve full diffusion. We have illustrated this explicitly with impossible differentials. Specifically, we claim that despite its larger state and more lightweight round function, Mixifer achieves full diffusion about as fast as AES.
- The algebraic degree of the round function is 3, as is that of its inverse. This already starts saturating after 5 rounds, so we don't expect there to be any problems.

Of course every concrete cipher can only be considered secure after intensive public scrutiny so we invite all members of the cryptographic community to attack Mixifer.

6.5 Implementation cost

Mixifer is designed for efficient constant-time implementations in hardware and in software on 32-bit and 64-bit architectures. We provide two reference implementations in C and an optimized assembly implementation for the (32-bit) ARM Cortex-M3 and M4 microcontrollers.

The first C implementation uses 4 `uint64_t`s to represent the state and implements the round function in a very straightforward way. In a naive software implementation like this, γ can be relatively computationally expensive. The S-box needs to be computed for each cell individually. Even when one applies the S-box to an entire row in parallel, bitmasks are typically required to separate the four bits of a cell over separate registers. By selecting a rotation-symmetric S-box and assuming a bitsliced state, this can be made more efficient, especially on architectures such as those by ARM where shifts and rotations are cheap. After loading our state, we take it to be organized as in Figure 6.

The ith byte of a 32-bit word stores the ith bit for 8 cells, as the S-box can then be computed in just 4 instructions per register. We also choose to interleave adjacent cells over two words, as a word-rotation by one nibble is now only a bytewise rotation by one bit and a swap of registers, which is free, instead of having to do this bytewise rotation on two words. The second C implementation uses this approach and represents the state with 8 `uint32_t`s.

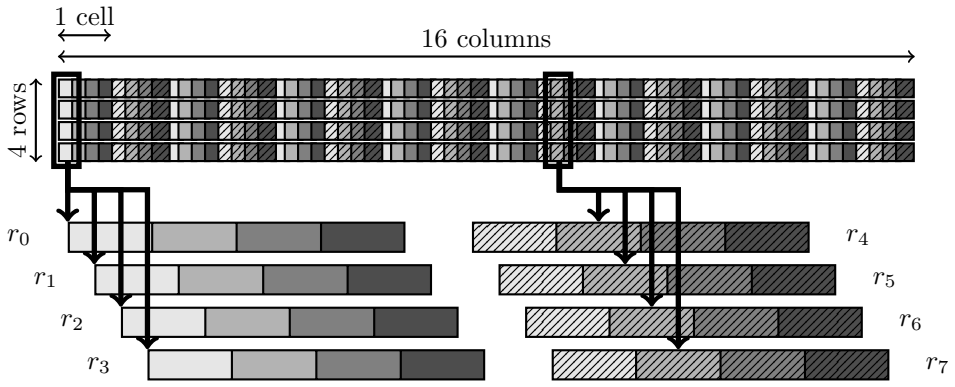


Figure 6: Bitsliced representation for 32-bit architectures. At the top is the 256-bits state, where every rectangle is a bit. At the bottom are 8 32-bit registers labeled r_0 to r_7, where every colored rectangle is a byte.

On the ARM Cortex-M3 or M4, we can compute γ with this representation in 32 single-cycle instructions. On a 64-bit architecture, this could naturally be twice as efficient.

Computing θ takes 6 cycles for computing the parity, 17 cycles for calculating the effect, and 8 cycles for adding it back to the state. π can be implemented by just renaming registers, and ρ takes us 2 cycles per row, except for the last row which is not rotated, so only 6 cycles in total. ι takes a single cycle.

All combined, our optimized assembly implementations needs 70 cycles for a single round. Together with some overhead for a function call and for loading and storing data, the full unrolled 16-round permutation runs in 1174 cycles, or 36.69 cycles per byte on the ARM Cortex-M4, as measured on an STM32F407 microcontroller. For ARM Cortex-M3, we measure 1175 cycles on an STM32L100 microcontroller. The implementation uses only 44 bytes on the stack, excluding the input. They are mostly to store callee-save registers.

Notably, the inverse permutation has the same memory requirements and even a slightly higher speed, measuring 1126 cycles on the STM32F407.

All source code is put in the public domain and available at `https://github.com/Ko-/mixifer`.

6.6 Comparing to other ciphers

To put our bounds and benchmarks into context, we compare the results to some other permutations and ciphers. We aim to make this comparison fair by only considering schemes for which there is an optimized implementation for the ARM Cortex-M3 or M4. Aside from the usual speed and size numbers, we would also like to somehow incorporate the 'quality' of a round function into the comparison.

Some schemes have many lightweight rounds while others have more computationally demanding rounds, but need less of them to reach a certain level of security. Some schemes are also more conservative than others with regard to the number of rounds compared to the best known attacks. Moreover, schemes target different security levels. All of that makes it hard to quantify the notion of the quality of a round function in a meaningful way. Intuitively, however, we believe that it would be interesting to look at the amount of 'security' that is gained per amount of 'work' that has to be done.

Table 6 lists some of the results. Regarding 'security', we consider the weight that a single round adds to the best known differential trail. In the table, the weight is expressed as the $-\log$ of the maximum differential probability (DP). 'Work' is then quantified as the number of cycles per byte per round on these Cortex-M microarchitectures. Of course these numbers do not reflect security against different types of attacks, performance on different architectures, the security margin, and so on.

AES [DR02] is a prime example of a cipher using an MDS matrix and with strong bounds. For fair comparison, we also mention performance numbers for a bitsliced implementation that does not use any secret-data dependent lookup tables and is not vulnerable to cache attacks. Gimli [Ber+17] is a 384-bit permutation with a much lighter round function which shows in its speed, but also in the best known trails. We compare to Salsa20/20 [Ber08b] because it is known for being very fast. The speed that is mentioned, is actually that of ChaCha20 [Ber08a], which has comparable speed. However, to the best of our knowledge, no bounds are known for ChaCha20 and Salsa20/20 does not have an optimized implementation for the Cortex-M3 and M4. We compare to Keccak-f[400] and Keccak-f[800] [Ber+11b] because they are known to have very strong security guarantees against linear and differential cryptanalysis in the form of a good bound on the weight of trails [MDA17]. The big gap in performance is because Keccak-f[800] is the smallest Keccak-f that can fully use 32-bit registers.

It is clear that Mixifer may not be as fast as Salsa20/20 or Gimli, but their currently available bounds leave something to be desired, although they may improve in the future. When one considers the amount of 'security' that one gets per amount of 'work' that needs to be done, Mixifer stands out.

Table 6: Comparison of performance on the Cortex-M3/M4 and bounds.

Name	Width (bits)	r	Speed (cycles/byte) Full	$/r$	r	Bound trails $-\log(\max DP)$	$/r$
AES bitsliced	128	10	50.52 [SS16]	5.05	4	150 [DR02]	37.5
AES tables			39.97 [SS16]	4.00			
Gimli	384	24	21.81 [Ber+17]	0.91	8	52 [Ber+17]	6.5
Keccak-f[400]	400	20	106 [Ber+14]	5.3	6	92 [MDA17]	15.3
Keccak-f[800]	800	22	48.02 [Ber+14]	2.18	6	92 [MDA17]	15.3
Salsa20/20	512	20	13.88 [HRS16]	0.69	3	18 [MP13]	6
Mixifer	256	16	36.69	2.33	4	92	23

7 Conclusions and future work

We have generalized the mixing layer of the permutation of Keccak and showed that column parity mixers are an interesting alternative to MDS matrices. They can be very lightweight mixing layers and they lend themselves well to bitsliced implementations, even for nibble-oriented ciphers and permutations. Additionally, we formulated a strategy to obtain strong bounds with respect to truncated linear and differential cryptanalysis and demonstrated its effectiveness by an example.

CPMs look promising, but there are still some interesting aspects that are worth investigating. For instance, intuitively it makes sense that a sparser parity-folding matrix means less diffusion and a cheaper implementation, but it is less obvious what would be an ideal trade-off and how this relates to the trail search space that needs to be covered. One can imagine that after some point, there is not so much to be gained by making a parity-folding matrix denser, especially when considering trails over many rounds.

For circulant parity-folding matrices, there exist interesting patterns after one selects which columns should be affected given a single isolated active cell. For example, consider the case where $z(x) = x + x^2 + x^3 + x^4$. Then a single active cell leads to four affected columns. However, with two active cells in adjacent columns, there would be some destructive interference in the θ-effect leading to only two affected columns. In general, something similar can happen with other parity-folding matrices and multiple odd columns. It is worth investigating what is an optimal choice.

As another example, it would be interesting to know whether there exist better transposition layers than a combination of row permutations and rotations. Or, to learn what is the optimal ratio between ℓ, m, and n.

For Mixifer, we considered truncated trails and while we touched upon the subject of trail clustering, more can be done. It would be interesting to investigate this behaviour for more rounds, both for AES and for MDS-based ciphers in general, as for Mixifer. We consider further research on clustering of trails to be future work.

We are looking forward to see any future work on the subject of column-parity mixers.

References

[77] *Data Encryption Standard*. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce. Jan. 1977.

[Alb+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. "Block Ciphers - Focus on the Linear Layer (feat. PRIDE)". In: *CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario

Gennaro. Vol. 8616. LNCS. Springer, Heidelberg, Aug. 2014, pp. 57–76. DOI: 10.1007/978-3-662-44371-2_4.

[Ban+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. "Midori: A Block Cipher for Low Energy". In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, Nov. 2015, pp. 411–436. DOI: 10.1007/978-3-662-48800-3_17.

[BBS05] Eli Biham, Alex Biryukov, and Adi Shamir. "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials". In: *Journal of Cryptology* 18.4 (Sept. 2005), pp. 291–311.

[Bei+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS". In: *CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, Heidelberg, Aug. 2016, pp. 123–153. DOI: 10.1007/978-3-662-53008-5_5.

[Bei+17] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. "Proving Resistance Against Invariant Attacks: How to Choose the Round Constants". In: *CRYPTO 2017, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. LNCS. Springer, Heidelberg, Aug. 2017, pp. 647–678.

[Ber+11a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *Cryptographic sponge functions*. Jan. 2011. URL: https://keccak.team/files/CSF-0.1.pdf.

[Ber+11b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *The* KECCAK *reference*. Jan. 2011. URL: http://keccak.noekeon.org/.

[Ber+14] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. *Keccak code package*. Oct. 2014. URL: https://github.com/gvanas/KeccakCodePackage.

[Ber+17] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. "Gimli : A Cross-Platform Permutation". In: *CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. LNCS. Springer, Heidelberg, Sept. 2017, pp. 299–320.

[Ber08a] Daniel J. Bernstein. "ChaCha, a variant of Salsa20". In: *Workshop Record of SASC*. 2008. URL: http://cr.yp.to/chacha/chacha-20080120.pdf.

[Ber08b] Daniel J. Bernstein. "The Salsa20 Family of Stream Ciphers". In: *New Stream Cipher Designs: The eSTREAM Finalists*. Ed. by Matthew Robshaw and Olivier Billet. Vol. 4986. LNCS. Springer, 2008, pp. 84–97. ISBN: 978-3-540-68351-3. DOI: 10.1007/978-3-540-68351-3_8.

[BKL16] Christof Beierle, Thorsten Kranz, and Gregor Leander. "Lightweight Multiplication in GF(2^n) with Applications to MDS Matrices". In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 625–653. DOI: 10.1007/978-3-662-53018-4_23.

[Bor+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. "PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract". In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, Dec. 2012, pp. 208–225. DOI: 10.1007/978-3-642-34961-4_14.

[BW99] Alex Biryukov and David Wagner. "Slide Attacks". In: *FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. LNCS. Springer, Heidelberg, Mar. 1999, pp. 245–259.

[DF04] David S. Dummit and Richard M. Foote. *Abstract algebra*. John Wiley and Sons, Inc., 2004.

[DGV93] Joan Daemen, René Govaerts, and Joos Vandewalle. "Block Ciphers Based on Modular Arithmetic". In: *Proceedings of the 3rd Symposium on the State and Progress of Research in Cryptography*. Ed. by W. Wolfowicz. 1993, pp. 80–89.

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael – AES, The Advanced Encryption Standard*. Springer, 2002. URL: http://jda.noekeon.org/JDA_VRI_Rijndael_2002.pdf.

[DR06] Joan Daemen and Vincent Rijmen. "Understanding Two-Round Differentials in AES". In: *SCN 06*. Ed. by Roberto De Prisco and Moti Yung. Vol. 4116. LNCS. Springer, Heidelberg, Sept. 2006, pp. 78–94.

[EM93] Shimon Even and Yishay Mansour. "A Construction of a Cipher From a Single Pseudorandom Permutation". In: *ASIACRYPT'91*. Ed. by Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto. Vol. 739. LNCS. Springer, Heidelberg, Nov. 1993, pp. 210–224.

[GPP11] Jian Guo, Thomas Peyrin, and Axel Poschmann. "The PHOTON Family of Lightweight Hash Functions". In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 222–239.

[Guo+11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. "The LED Block Cipher". In: *CHES 2011*. Ed. by Bart Preneel and Tsuyoshi Takagi. Vol. 6917. LNCS. Springer, Heidelberg, Sept. 2011, pp. 326–341.

[HRS16] Andreas Hülsing, Joost Rijneveld, and Peter Schwabe. "ARMed SPHINCS - Computing a 41 KB Signature in 16 KB of RAM". In: *PKC 2016, Part I*. Ed. by Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang. Vol. 9614. LNCS. Springer, Heidelberg, Mar. 2016, pp. 446–470. DOI: 10.1007/978-3-662-49384-7_17.

[JNP15] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. *Joltik v1.3*. Submission to the CAESAR submission. Aug. 2015. URL: http://www1.spms.ntu.edu.sg/~syllab/m/images/c/ca/V1.3_Joltik.pdf.

[Kav+14] Elif Bilge Kavun, Martin M. Lauridsen, Gregor Leander, Christian Rechberger, Peter Schwabe, and Tolga Yalçın. *Prøst v1.1*. Submission to the CAESAR competition. 2014. URL: https://competitions.cr.yp.to/round1/proestv11.pdf.

[Kav12] Selçuk Kavut. "Results on rotation-symmetric S-boxes". In: *Information Sciences* 201 (2012), pp. 93–113. DOI: 10.1016/j.ins.2012.02.030.

[KN10] Dmitry Khovratovich and Ivica Nikolic. "Rotational Cryptanalysis of ARX". In: *FSE 2010*. Ed. by Seokhie Hong and Tetsu Iwata. Vol. 6147. LNCS. Springer, Heidelberg, Feb. 2010, pp. 333–346.

[Knu95] Lars R. Knudsen. "Truncated and Higher Order Differentials". In: *FSE'94*. Ed. by Bart Preneel. Vol. 1008. LNCS. Springer, Heidelberg, Dec. 1995, pp. 196–211.

[Kra+17] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. "Shorter Linear Straight-Line Programs for MDS Matrices". In: *IACR Trans. Symm. Cryptol.* 2017.4 (2017), pp. 188–211. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i4.188-211.

[Lea+11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. "A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack". In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 206–221.

[LLM16] David C. Lay, Steven R. Lay, and Judi J. McDonald. *Linear Algebra and Its Applications*. 5th ed. Pearson, 2016.

[LP07] Gregor Leander and Axel Poschmann. "On the Classification of 4 Bit S-Boxes". In: *Arithmetic of Finite Fields — WAIFI 2007*. Ed. by Claude Carlet and Berk Sunar. Vol. 4547. LNCS. Springer, 2007, pp. 159–176. DOI: 10.1007/978-3-540-73074-3_13.

[LS16] Meicheng Liu and Siang Meng Sim. "Lightweight MDS Generalized Circulant Matrices". In: *FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. LNCS. Springer, Heidelberg, Mar. 2016, pp. 101–120. DOI: 10.1007/978-3-662-52993-5_6.

[LW16] Yongqiang Li and Mingsheng Wang. "On the Construction of Lightweight Circulant Involutory MDS Matrices". In: *FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. LNCS. Springer, Heidelberg, Mar. 2016, pp. 121–139. DOI: 10.1007/978-3-662-52993-5_7.

[LW17] Chaoyun Li and Qingju Wang. "Design of Lightweight Linear Diffusion Layers from Near-MDS Matrices". In: *IACR Trans. Symm. Cryptol.* 2017.1 (2017), pp. 129–155. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.129-155.

[Mat94] Mitsuru Matsui. "Linear Cryptanalysis Method for DES Cipher". In: *EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. LNCS. Springer, Heidelberg, May 1994, pp. 386–397.

[MDA17] Silvia Mella, Joan Daemen, and Gilles Van Assche. "New techniques for trail bounds and application to differential trails in Keccak". In: *IACR Trans. Symm. Cryptol.* 2017.1 (2017), pp. 329–357. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.329-357.

[Men16] Bart Mennink. "XPX: Generalized Tweakable Even-Mansour with Improved Security Guarantees". In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 64–94. DOI: 10.1007/978-3-662-53018-4_3.

[MP13] Nicky Mouha and Bart Preneel. *Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20*. Cryptology ePrint Archive, Report 2013/328. http://eprint.iacr.org/2013/328. 2013.

[NIS15] NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. FIPS 202. Aug. 2015. DOI: 10.6028/NIST.FIPS.202.

[RBG08] Vincent Rijmen, Paulo SLM Barreto, and Décio L Gazzoni Filho. "Rotation symmetry in algebraically generated cryptographic substitution tables". In: *Information Processing Letters* 106.6 (2008), pp. 246–250. DOI: 10.1016/j.ipl.2007.09.012.

[Rij+96] Vincent Rijmen, Joan Daemen, Bart Preneel, Anton Bossalaers, and Erik De Win. "The Cipher SHARK". In: *FSE'96*. Ed. by Dieter Gollmann. Vol. 1039. LNCS. Springer, Heidelberg, Feb. 1996, pp. 99–111.

[Sas+15] Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, Mitsuru Matsui, and Shoichi Hirose. *Minalpher v1.1*. Submission to the CAESAR competition. 2015. URL: https://info.isl.ntt.co.jp/crypt/minalpher/files/minalpherv11.pdf.

[SS16] Peter Schwabe and Ko Stoffelen. "All the AES You Need on Cortex-M3 and M4". In: *SAC 2016*. Ed. by Roberto Avanzi and Howard M. Heys. Vol. 10532. LNCS. Springer, Heidelberg, Aug. 2016, pp. 180–194.

[TLS16] Yosuke Todo, Gregor Leander, and Yu Sasaki. "Nonlinear Invariant Attack - Practical Attack on Full SCREAM, iSCREAM, and Midori64". In: *ASIACRYPT 2016, Part II*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10032. LNCS. Springer, Heidelberg, Dec. 2016, pp. 3–33. DOI: 10.1007/978-3-662-53890-6_1.

A S-box

The S-box S in algebraic normal form.

$$\begin{aligned}
b_0 &= a_1 + a_2 + a_0a_2 + a_1a_2 + a_1a_2a_3 \\
b_1 &= a_2 + a_3 + a_1a_3 + a_2a_3 + a_2a_3a_0 \\
b_2 &= a_3 + a_0 + a_2a_0 + a_3a_0 + a_3a_0a_1 \\
b_3 &= a_0 + a_1 + a_3a_1 + a_0a_1 + a_0a_1a_2
\end{aligned}$$

S can be computed more efficiently. This takes 4 instructions in ARMv7-M, assuming the bit lay-out that was sketched in Section 6.5. In the assembly implementation, a_i are stored in `s` and `t` is a temporary register.

$$\begin{aligned}
b_0 &= a_1 + a_2(a_0 + (\overline{a_1}\ \texttt{OR}\ a_3)) \\
b_1 &= a_2 + a_3(a_1 + (\overline{a_2}\ \texttt{OR}\ a_0)) \\
b_2 &= a_3 + a_0(a_2 + (\overline{a_3}\ \texttt{OR}\ a_1)) \\
b_3 &= a_0 + a_1(a_3 + (\overline{a_0}\ \texttt{OR}\ a_2))
\end{aligned}$$

```
.macro gammapart s t
    orn \t, \s, \s, ror #16
    eor \t, \s, \t, ror #8
    and \t, \t, \s, ror #16
    eor \s, \t, \s, ror #24
.endm
```

The inverse S-box S^{-1} in algebraic normal form.

$$\begin{aligned}
a_0 &= b_0 + b_1 + b_3 + b_0b_2 + b_1b_2 + b_1b_3 + b_1b_2b_3 \\
a_1 &= b_1 + b_2 + b_0 + b_1b_3 + b_2b_3 + b_2b_0 + b_2b_3b_0 \\
a_2 &= b_2 + b_3 + b_1 + b_2b_0 + b_3b_0 + b_3b_1 + b_3b_0b_1 \\
a_3 &= b_3 + b_0 + b_2 + b_3b_1 + b_0b_1 + b_0b_2 + b_0b_1b_2
\end{aligned}$$

S^{-1} can be computed more efficiently. This takes 4 instructions in ARMv7-M.

$$a_0 = b_3 + (\overline{b_2}(b_0 + (b_1\overline{b_3})))$$
$$a_1 = b_0 + (\overline{b_3}(b_1 + (b_2\overline{b_0})))$$
$$a_2 = b_1 + (\overline{b_0}(b_2 + (b_3\overline{b_1})))$$
$$a_3 = b_2 + (\overline{b_1}(b_3 + (b_0\overline{b_2})))$$

```
.macro invgammapart s t
    bic \t, \s, \s, ror #16
    eor \t, \s, \t, ror #24
    bic \t, \t, \s, ror #16
    eor \s, \t, \s, ror #8
.endm
```

The S-box as hexadecimal lookup table.

a	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(a)$	0	c	9	d	3	a	b	2	6	e	5	1	7	8	4	f

Below are the difference distribution table and linear approximation table of the S-box S.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	4	0	0	0	2	4	0	2	2	0	0	2
2	0	2	0	4	0	0	0	2	4	2	0	0	0	0	0	2
3	0	4	0	0	0	2	0	2	2	0	0	2	2	2	0	0
4	0	4	2	2	0	0	4	0	0	0	0	0	0	0	2	2
5	0	0	0	0	0	2	2	0	0	2	4	2	0	0	2	2
6	0	2	4	0	0	0	0	2	0	2	2	2	0	0	2	0
7	0	0	2	2	0	0	2	2	0	2	2	0	0	2	2	0
8	0	0	4	0	2	0	2	0	0	0	0	0	4	2	0	2
9	0	0	0	0	2	0	2	0	4	0	2	2	0	2	2	0
a	0	0	0	2	0	4	0	2	0	0	2	0	2	2	0	2
b	0	2	0	2	0	2	0	2	0	2	0	2	2	0	2	0
c	0	0	2	2	4	2	0	2	0	0	0	0	0	2	2	0
d	0	0	0	0	0	0	2	2	2	2	2	2	2	2	0	0
e	0	0	0	2	2	2	2	0	0	0	0	2	2	2	2	0
f	0	2	2	0	2	2	0	0	2	0	2	0	0	0	0	4

DDT

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	-2	0	2	0	-2	0	2	4	2	0	2	-4	2	0	2
2	0	4	-2	2	0	0	2	2	0	-4	-2	2	0	0	2	2
3	0	-2	-2	0	0	2	2	0	4	2	-2	0	4	-2	2	0
4	0	0	4	-4	-2	-2	2	2	0	0	0	0	2	2	2	2
5	0	2	0	-2	2	4	2	0	0	2	0	-2	-2	0	-2	4
6	0	4	-2	2	-2	-2	0	0	0	4	2	-2	2	2	0	0
7	0	2	2	0	2	0	0	2	0	2	2	0	-2	-4	4	-2
8	0	0	0	0	4	0	-4	0	-2	2	-2	2	2	2	2	2
9	0	-2	0	2	4	-2	4	2	-2	0	2	0	2	0	-2	0
a	0	0	2	2	0	0	-2	-2	2	-2	4	0	2	-2	0	4
b	0	2	2	0	0	2	-2	4	2	0	0	2	2	0	-4	-2
c	0	0	4	4	-2	2	2	-2	-2	2	-2	2	0	0	0	0
d	0	2	0	-2	2	0	2	-4	2	0	2	4	0	2	0	-2
e	0	0	2	2	2	2	0	0	2	-2	0	-4	0	4	2	-2
f	0	-2	-2	0	-2	4	0	2	-2	0	4	2	0	2	2	0

LAT

B Test vectors

```
Input:  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000.
Output: 82f0778a 1696c72c 5cd10f7e f3e2b448 dc8c6a7d 2d753216 43d11823 14f5d1f7.

Input:  00010203 04050607 08090a0b 0c0d0e0f 10111213 14151617 18191a1b 1c1d1e1f.
Output: e41eeafb 1c57ee87 a835b347 2ddef9cd 27ab6bff 95fd6df4 548db2bd 065d19b1.

Input:  fffefdfc fbfaf9f8 f7f6f5f4 f3f2f1f0 efeeedec ebeae9e8 e7e6e5e4 e3e2e1e0.
Output: aefa559b 3a7a5781 6b10507d 5c68429a 0183a7cb 09575707 985a20c3 71f2d6ac.
```

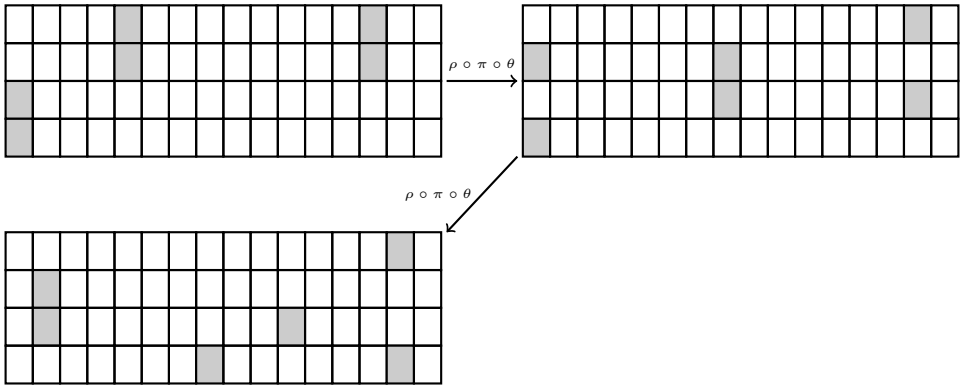
C Minimum weight truncated trails

C.1 In the kernel

The following trails attempt to stay in the kernel for all rounds except the last.

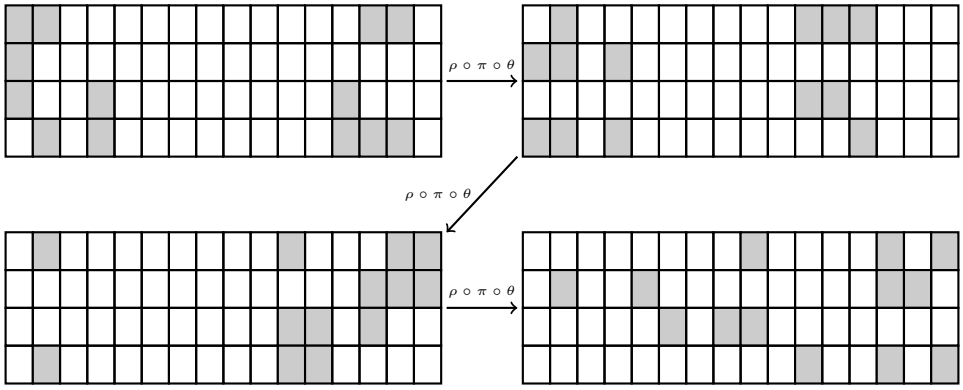
Three rounds

The following trail has $W = 18$.



Four rounds

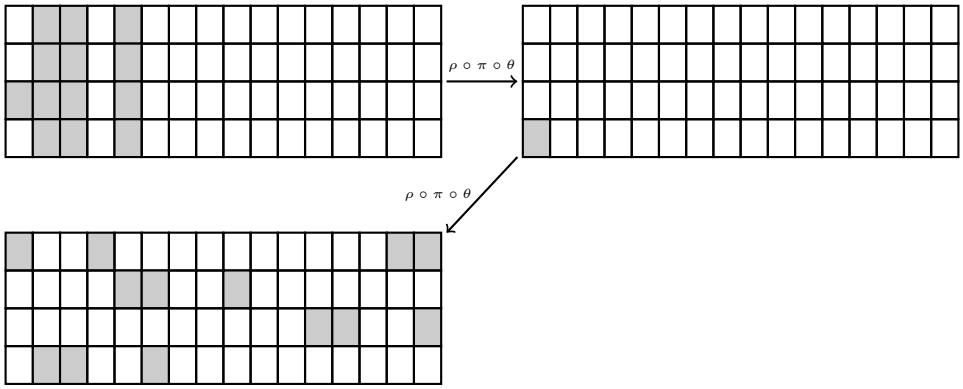
The following trail has $W = 52$.



C.2 Extending two-round differential trails outside the kernel

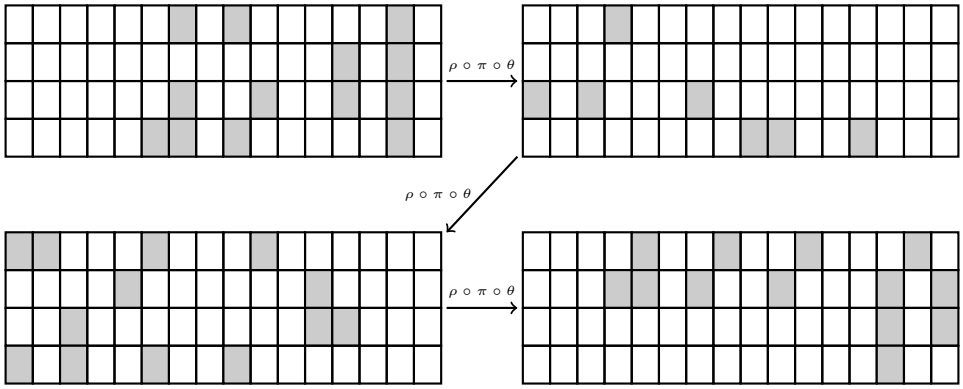
Three rounds

The following trail has $W = 27$.



Four rounds

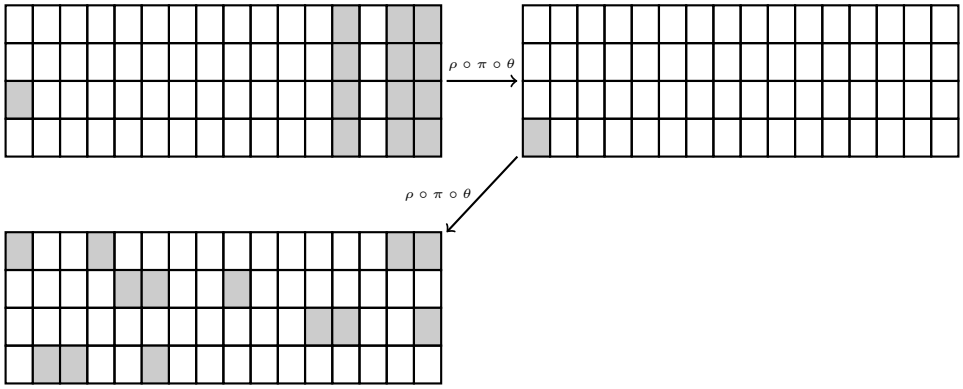
The following trail has $W = 46$.



C.3 Extending two-round linear trails outside the kernel

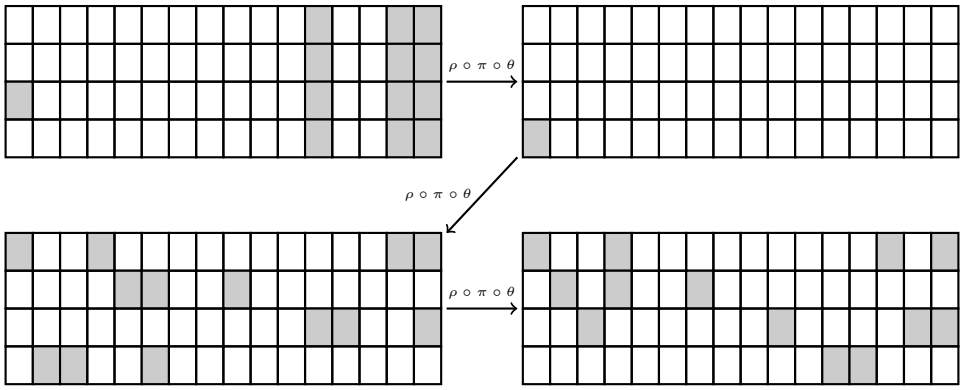
Three rounds

The following trail has $W = 27$.



Four rounds

The following trail has $W = 40$.



D Effect distribution

The following table shows the relation between the Hamming weight of the parity (horizontal axis) and the Hamming weight of the θ-effect (vertical axis) for Mixifer. The numbers show the number of possibilities. For example, there is only one possibility where there is a single odd column. Then there will be three affected columns. With two odd columns, it is possible that six columns become affected, but it is also possible that a few cancel each other out and there are only four affected columns.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	5	0	2	0	0	0	0	0	0
3	1	0	1	0	3	0	11	0	16	0	3	0	0	0	0	0
4	0	3	0	8	0	23	0	40	0	40	0	2	0	0	0	0
5	0	0	9	0	30	0	81	0	111	0	42	0	0	0	0	0
6	0	5	0	31	0	98	0	213	0	134	0	23	0	0	0	0
7	0	0	19	0	96	0	237	0	258	0	99	0	6	0	0	0
8	0	0	0	52	0	206	0	294	0	206	0	52	0	0	0	0
9	0	0	6	0	99	0	258	0	237	0	96	0	19	0	0	0
10	0	0	0	23	0	134	0	213	0	98	0	31	0	5	0	0
11	0	0	0	0	42	0	111	0	81	0	30	0	9	0	0	0
12	0	0	0	2	0	40	0	40	0	23	0	8	0	3	0	0
13	0	0	0	0	3	0	16	0	11	0	3	0	1	0	1	0
14	0	0	0	0	0	2	0	5	0	1	0	0	0	0	0	0
15	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1