Mixing Layers in Symmetric Crypto

Ko Stoffelen

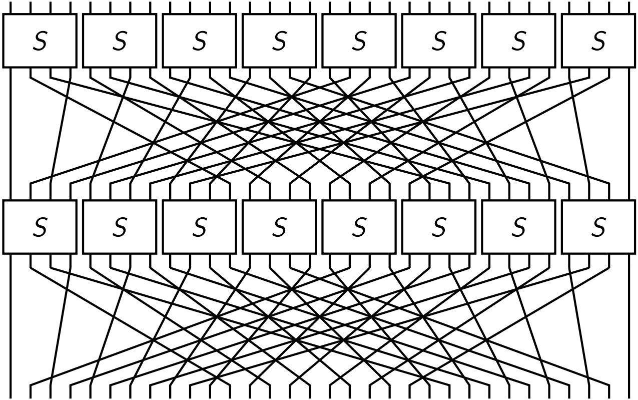
iCIS | Digital Security
Radboud University

Joint work with. . .

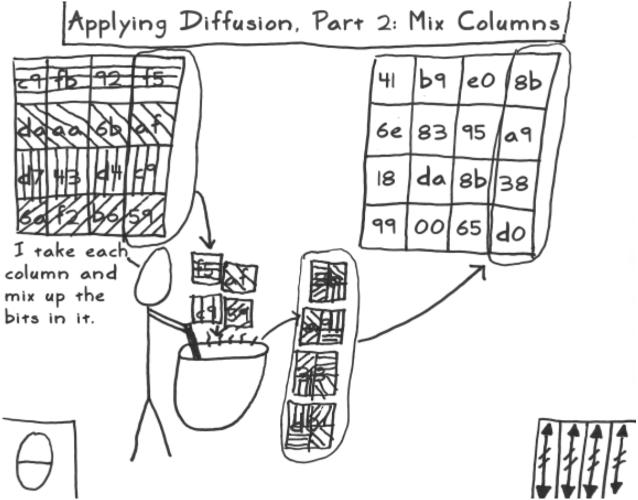
Thorsten Kranz, Gregor Leander, Ko Stoffelen, Friedrich Wiemer. **Shorter Linear Straight-Line Programs for MDS Matrices.** *ToSC 2017 Issue 4.*

Ko Stoffelen, Joan Daemen. **Column Parity Mixers.** *ToSC 2018 Issue 1.*

Diffusion



Diffusion in AES



Diffusion in AES

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

MDS matrices

- Coding theory has *maximum distance separable* (MDS) codes

MDS matrices

- Coding theory has *maximum distance separable* (MDS) codes
- "Reaches Singleton bound"

MDS matrices

- Coding theory has *maximum distance separable* (MDS) codes
- "Reaches Singleton bound"
- Given $[n, k, d]$ code over $\mathbb{F}_q$, best error correction when $d = n - k + 1$

MDS matrices

- Coding theory has *maximum distance separable* (MDS) codes
- "Reaches Singleton bound"
- Given $[n, k, d]$ code over $\mathbb{F}_q$, best error correction when $d = n - k + 1$
- For example, Reed-Solomon codes (DVD, Blu-ray)

MDS matrices

- Coding theory has *maximum distance separable* (MDS) codes
- "Reaches Singleton bound"
- Given $[n, k, d]$ code over $\mathbb{F}_q$, best error correction when $d = n - k + 1$
- For example, Reed-Solomon codes (DVD, Blu-ray)
- Use this idea for diffusion in crypto!

MDS matrices

- Coding theory has *maximum distance separable* (MDS) codes
- "Reaches Singleton bound"
- Given $[n, k, d]$ code over $\mathbb{F}_q$, best error correction when $d = n - k + 1$
- For example, Reed-Solomon codes (DVD, Blu-ray)
- Use this idea for diffusion in crypto!
- An $m \times n$ matrix A over $\mathbb{F}_q$ is MDS when the set $\{(x_1, \ldots, x_n, (Ax)_1, \ldots, (Ax)_m) | x \in \mathbb{F}_q^n\}$ forms an MDS code

MDS matrices

- Coding theory has *maximum distance separable* (MDS) codes
- "Reaches Singleton bound"
- Given $[n, k, d]$ code over $\mathbb{F}_q$, best error correction when $d = n - k + 1$
- For example, Reed-Solomon codes (DVD, Blu-ray)
- Use this idea for diffusion in crypto!
- An $m \times n$ matrix A over $\mathbb{F}_q$ is MDS when the set $\left\{(x_1, \ldots, x_n, (Ax)_1, \ldots, (Ax)_m) | x \in \mathbb{F}_q^n\right\}$ forms an MDS code
- MixColumns matrix is MDS

Lightweight MDS matrices

Lightweight Multiplication in $GF(2^n)$ with Applications to MDS Matrices

Christof Beierle(✉), Thorsten Kranz, and Gregor Leander

Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Bochum, Germany
{christof.beierle,thorsten.kranz,gregor.leander}@rub.de

Abstract. In this paper we consider the fundamental question of optimizing finite field multiplications with one fixed element. Surprisingly,

Lightweight MDS matrices

Lightweight Multiplication in $GF(2^n)$ with Applications to MDS Matrices

Christof Beierle(✉), Thorsten Kranz, and Gregor Leander

Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Bochum, Germany
{christof.beierle,thorsten.kranz

Abstract. In this paper we consider the
mizing finite field multiplications with o

Optimizing Implementations of Lightweight Building Blocks

Jérémy Jean[1], Thomas Peyrin[2], Siang Meng Sim[2] and Jade Tourteaux[1,3]

[1] ANSSI Crypto Lab, Paris, France
Jeremy.Jean@ssi.gouv.fr

[2] Nanyang Technological University, Singapore
Thomas.Peyrin@ntu.edu.sg, ssim011@e.ntu.edu.sg

[3] Paris Diderot University, Paris, France
Jade.Tourteaux@gmail.com

Abstract. We study the synthesis of small functions used as building blocks in lightweight cryptographic designs in terms of hardware implementations. This phase

Lightweight MDS matrices

Lightweight Multiplication in $GF(2^n)$ with Applications to MDS Matrices

Christof Beierle(✉), Thorsten Kranz, and Gregor Leander

Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Bochum, Germany
{christof.beierle,thorsten.kranz,gregor.leander}@rub.de

Lightweight MDS Generalized Circulant Matrices

Meicheng Liu[1,2](✉) and Siang Meng Sim[1](✉)

[1] Nanyang Technological University, Singapore, Singapore
ssim011@e.ntu.edu.sg
[2] State Key Laboratory of Information Security,
Institute of Information Engineering, Chinese Academy of Sciences,
Beijing 100093, People's Republic of China
meicheng.liu@gmail.com

Abstract. In this article, we analyze the circulant structure of generalized circulant matrices to reduce the search space for finding lightweight

Optimizing Implementations ... Building Blocks

ang Meng Sim[2] and Jade Tourteaux[1,3]

Lab, Paris, France
n@ssi.gouv.fr

ical University, Singapore
.sg, ssim011@e.ntu.edu.sg

iversity, Paris, France
eaux@gmail.com

small functions used as building blocks in
ns of hardware implementations. This phase

Lightweight MDS matrices

Lightweight Multip
with Applications

Christof Beierle[✉], Thorsten

Horst Görtz Institute for IT Security, Ruh
{christof.beierle,thorsten.k

On the Construction of Lightweight Circulant Involutory MDS Matrices

Yongqiang Li[1,2(✉)] and Mingsheng Wang[1]

[1] State Key Laboratory of Information Security,
Institute of Information Engineering, Chinese Academy of Sciences,
Beijing, China
yongq.lee@gmail.com, wangmingsheng@iie.ac.cn
[2] Science and Technology on Communication Security Laboratory,
Chengdu, China

,3

Abstract. In the present paper, we investigate the problem of constructing MDS matrices with as few bit XOR operations as possible.

Lightweight MDS
M

Meicheng Liu[1,2(✉)]

[1] Nanyang Technological University, Singapore, Singapore
ssim011@e.ntu.edu.sg
[2] State Key Laboratory of Information Security,
Institute of Information Engineering, Chinese Academy of Sciences,
Beijing 100093, People's Republic of China
meicheng.liu@gmail.com

ical University, Singapore
.sg, ssim011@e.ntu.edu.sg

iversity, Paris, France
eaux@gmail.com

small functions used as building blocks in
ns of hardware implementations. This phase

Abstract. In this article, we analyze the circulant structure of generalized circulant matrices to reduce the search space for finding lightweight

Lightweight MDS matrices

Lightweight Multip
with Applications

Christof Beierle(✉), Thorsten

Horst Görtz Institute for IT Security, Ruh

On the Construction of Lightweight Circulant Involutory MDS Matrices

Yongqiang Li[1,2](✉) and Mingsheng Wang[1]

Information Security,
, Chinese Academy of Sciences,
hina
gmingsheng@iie.ac.cn
unication Security Laboratory,
China
,3

Lightweight MDS Involution Matrices

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

[1] Nanyang Technological University, Singapore, Singapore
ssim011@e.ntu.edu.sg, {frederique,thomas.peyrin}@ntu.edu.sg
[2] DSO National Laboratories, Singapore, Singapore
kkhoongm@dso.org.sg

investigate the problem of con-
bit XOR operations as possible.

ical University, Singapore
.sg, ssim011@e.ntu.edu.sg

iversity, Paris, France
eaux@gmail.com

Abstract. In this article, we provide new methods to look for lightweight MDS matrices, and in particular involutory ones. By proving

small functions used as building blocks in
ns of hardware implementations. This phase

Abstract. In this article, we analyze the circulant structure of generalized circulant matrices to reduce the search space for finding lightweight

Lightweight MDS matrices

Lightweight Multip
with Applications

Christof Beierle(✉), Thorsten

Horst Görtz Institute for IT Security, Ruh

On the Construction of Lightweight Circulant Involutory MDS Matrices

Yongqiang Li[1,2](✉) and Mingsheng Wang[1]

Information Security,
, Chinese Academy of Sciences,

Lightweight MDS Involution Matrices

Siang Meng Sim[1](✉), Khoong
and Tho

[1] Nanyang Technological U
ssim011@e.ntu.edu.sg, {fred
[2] DSO National Labora
kkhoong

Lightweight Diffusion Layer: Importance of Toeplitz Matrices

Sumanta Sarkar[1] and Habeeb Syed[2]

[1] TCS Innovation Labs, Hyderabad, INDIA, Sumanta.Sarkar1@tcs.com
[2] TCS Innovation Labs, Hyderabad, INDIA, Habeeb.Syed@tcs.com

Abstract. MDS matrices are used as building blocks of diffusion layers in block ciphers, and XOR count is a metric that estimates the hardware implementation cost. In this paper we report the minimum value of XOR counts of 4 × 4 MDS matrices

ns of hardware implementations. This phase

Abstract. In this article, we pr
weight MDS matrices, and in p

Abstract. In this article, we analyze the circulant structure of generalized circulant matrices to reduce the search space for finding lightweight

Lightweight MDS matrices

- Cauchy and Vandermonde matrices are MDS, but don't exist for all parameters

Lightweight MDS matrices

- Cauchy and Vandermonde matrices are MDS, but don't exist for all parameters
- Add structure to reduce search space (Hadamard, circulant, Toeplitz, subfield)

Lightweight MDS matrices

- Cauchy and Vandermonde matrices are MDS, but don't exist for all parameters
- Add structure to reduce search space (Hadamard, circulant, Toeplitz, subfield)
- Have to test for MDS-ness, but the probability is higher than for a random matrix

Lightweight MDS matrices

- Cauchy and Vandermonde matrices are MDS, but don't exist for all parameters
- Add structure to reduce search space (Hadamard, circulant, Toeplitz, subfield)
- Have to test for MDS-ness, but the probability is higher than for a random matrix
- `XOR` count of a: number of `XOR`s to multiply a with an arbitrary b, $a, b \in \mathbb{F}_{2^k}$

Lightweight MDS matrices

- Cauchy and Vandermonde matrices are MDS, but don't exist for all parameters
- Add structure to reduce search space (Hadamard, circulant, Toeplitz, subfield)
- Have to test for MDS-ness, but the probability is higher than for a random matrix
- `XOR` count of a: number of `XOR`s to multiply a with an arbitrary b, $a, b \in \mathbb{F}_{2^k}$
- Assumes 'summation' has constant cost, local optimization

Lightweight MDS matrices

- Cauchy and Vandermonde matrices are MDS, but don't exist for all parameters
- Add structure to reduce search space (Hadamard, circulant, Toeplitz, subfield)
- Have to test for MDS-ness, but the probability is higher than for a random matrix
- `XOR` count of a: number of `XOR`s to multiply a with an arbitrary b, $a, b \in \mathbb{F}_{2^k}$
- Assumes 'summation' has constant cost, local optimization
- Core idea: an $n \times n$ matrix over $\mathbb{F}_{2^k}$ can be viewed as $nk \times nk$ matrix over $\mathbb{F}_2$, do global optimization

Lightweight MDS matrices

- Cauchy and Vandermonde matrices are MDS, but don't exist for all parameters
- Add structure to reduce search space (Hadamard, circulant, Toeplitz, subfield)
- Have to test for MDS-ness, but the probability is higher than for a random matrix
- `XOR` count of a: number of `XOR`s to multiply a with an arbitrary b, $a, b \in \mathbb{F}_{2^k}$
- Assumes 'summation' has constant cost, local optimization
- Core idea: an $n \times n$ matrix over $\mathbb{F}_{2^k}$ can be viewed as $nk \times nk$ matrix over $\mathbb{F}_2$, do global optimization
- Solving the shortest linear straight-line program (SLP) problem yields the optimal number of `XOR`s

Lightweight MDS matrices

- Cauchy and Vandermonde matrices are MDS, but don't exist for all parameters
- Add structure to reduce search space (Hadamard, circulant, Toeplitz, subfield)
- Have to test for MDS-ness, but the probability is higher than for a random matrix
- `XOR` count of a: number of `XOR`s to multiply a with an arbitrary b, $a, b \in \mathbb{F}_{2^k}$
- Assumes 'summation' has constant cost, local optimization
- Core idea: an $n \times n$ matrix over $\mathbb{F}_{2^k}$ can be viewed as $nk \times nk$ matrix over $\mathbb{F}_2$, do global optimization
- Solving the shortest linear straight-line program (SLP) problem yields the optimal number of `XOR`s
- Well-studied problem, known algorithms give better results

Lightweight MDS matrices

- Cauchy and Vandermonde matrices are MDS, but don't exist for all parameters
- Add structure to reduce search space (Hadamard, circulant, Toeplitz, subfield)
- Have to test for MDS-ness, but the probability is higher than for a random matrix
- `XOR` count of a: number of `XOR`s to multiply a with an arbitrary b, $a, b \in \mathbb{F}_{2^k}$
- Assumes 'summation' has constant cost, local optimization
- Core idea: an $n \times n$ matrix over $\mathbb{F}_{2^k}$ can be viewed as $nk \times nk$ matrix over $\mathbb{F}_2$, do global optimization
- Solving the shortest linear straight-line program (SLP) problem yields the optimal number of `XOR`s
- Well-studied problem, known algorithms give better results
- Remove *common subexpressions*, allow *cancellation*

SLP example

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \Rightarrow \left\{ \begin{matrix} x_0 \oplus x_1 \\ x_0 \oplus x_1 \oplus x_2 \\ x_0 \oplus x_1 \oplus x_2 \oplus x_3 \\ x_1 \oplus x_2 \oplus x_3 \end{matrix} \right\}$$

SLP example

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \Rightarrow \left\{ \begin{matrix} x_0 \oplus x_1 \\ x_0 \oplus x_1 \oplus x_2 \\ x_0 \oplus x_1 \oplus x_2 \oplus x_3 \\ x_1 \oplus x_2 \oplus x_3 \end{matrix} \right\}$$

$$v_0 := x_0 \oplus x_1$$

SLP example

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \Rightarrow \left\{ \begin{matrix} x_0 \oplus x_1 \\ x_0 \oplus x_1 \oplus x_2 \\ x_0 \oplus x_1 \oplus x_2 \oplus x_3 \\ x_1 \oplus x_2 \oplus x_3 \end{matrix} \right\}$$

$$\begin{aligned} v_0 &:= x_0 \oplus x_1 \\ v_1 &:= v_0 \oplus x_2 \end{aligned}$$

SLP example

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \Rightarrow \left\{ \begin{matrix} x_0 \oplus x_1 \\ x_0 \oplus x_1 \oplus x_2 \\ x_0 \oplus x_1 \oplus x_2 \oplus x_3 \\ x_1 \oplus x_2 \oplus x_3 \end{matrix} \right\}$$

$$\begin{aligned} v_0 &:= x_0 \oplus x_1 \\ v_1 &:= v_0 \oplus x_2 \\ v_2 &:= v_1 \oplus x_3 \end{aligned}$$

SLP example

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \Rightarrow \left\{ \begin{matrix} x_0 \oplus x_1 \\ x_0 \oplus x_1 \oplus x_2 \\ x_0 \oplus x_1 \oplus x_2 \oplus x_3 \\ x_1 \oplus x_2 \oplus x_3 \end{matrix} \right\}$$

$$\begin{aligned} v_0 &:= x_0 \oplus x_1 \\ v_1 &:= v_0 \oplus x_2 \\ v_2 &:= v_1 \oplus x_3 \\ v_3 &:= v_2 \oplus x_0 \end{aligned}$$

Alternative mixing layers

- Good diffusion over a few rounds is more important

Alternative mixing layers

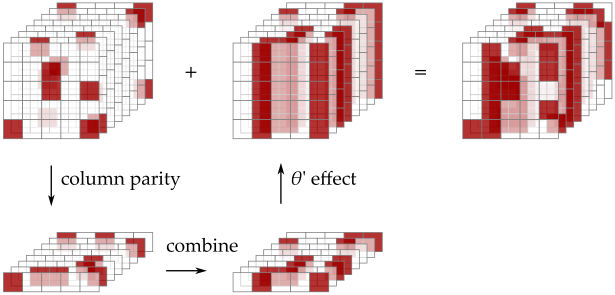
- Good diffusion over a few rounds is more important
- MDS requirement can be dropped when there are good bounds on trails (attacks)

Alternative mixing layers

- Good diffusion over a few rounds is more important
- MDS requirement can be dropped when there are good bounds on trails (attacks)
- PRIDE uses a *near-MDS* matrix, with a few more zeroes

Alternative mixing layers

- Good diffusion over a few rounds is more important
- MDS requirement can be dropped when there are good bounds on trails (attacks)
- PRIDE uses a *near-MDS* matrix, with a few more zeroes
- KECCAK-*f* uses a *column parity mixer* (CPM)



Column parity mixers

For an $m \times n$ matrix A:

$$\theta(A) = A + \quad f(A)$$

Column parity mixers

For an $m \times n$ matrix A:

$$\theta(A) = A + \quad \mathbf{1}_m^\mathsf{T} A$$

Column parity mixers

For an $m \times n$ matrix A:

$$\theta(A) = A + \quad \mathbf{1}_m^\top A Z$$

Column parity mixers

For an $m \times n$ matrix A:

$$\theta(A) = A + \mathbf{1}_m \mathbf{1}_m^\top A Z$$

Column parity mixers

For an $m \times n$ matrix A:

$$\theta(A) = A + \quad \mathbf{1}_m^m A Z$$

Column parity mixers

For an $m \times n$ matrix A:

$$\theta(A) = A + \quad \mathbf{1}_m^m A Z$$

θ fully defined by m and Z

Column parity mixers

For an $m \times n$ matrix A:

$$\theta(A) = A + \quad \mathbf{1}_m^m A Z$$

θ fully defined by m and Z

Some algebraic properties:

Column parity mixers

For an $m \times n$ matrix A:

$$\theta(A) = A + \quad \mathbf{1}_m^m A Z$$

θ fully defined by m and Z

Some algebraic properties:

- If m even, CPMs are involutions (as $(\mathbf{1}_m^m)^2 = \mathbf{0}$), commutative, $\cong (\mathbb{Z}_2^{n^2}, +)$

Column parity mixers

For an $m \times n$ matrix A:

$$\theta(A) = A + \quad \mathbf{1}_m^m A Z$$

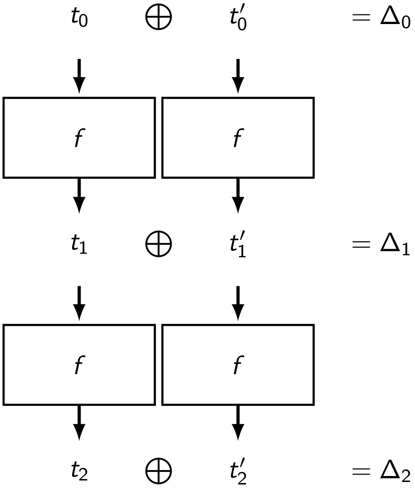
θ fully defined by m and Z

Some algebraic properties:

- If m even, CPMs are involutions (as $(\mathbf{1}_m^m)^2 = \mathbf{0}$), commutative, $\cong (\mathbb{Z}_2^{n^2}, +)$
- If m odd, CPMs invertible iff $Z + \mathbf{I}$ is invertible, non-commutative, $\cong GL(n, 2)$ if $Z + \mathbf{I}$ non-singular

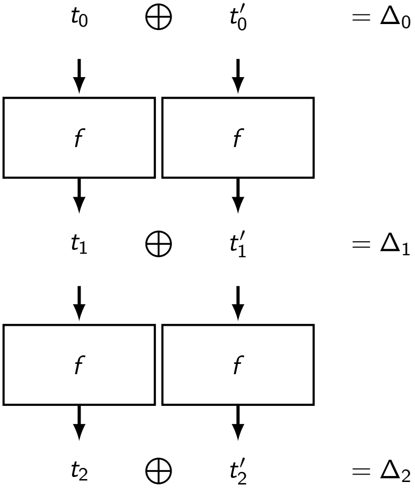
Differential cryptanalysis in a nutshell

- If $\Pr[\Delta_0, \ldots, \Delta_r]$ is high, cipher is not random



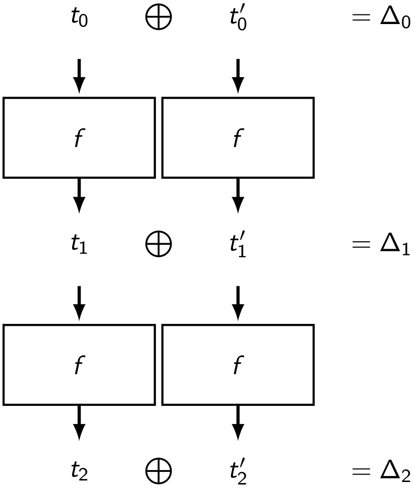
Differential cryptanalysis in a nutshell

- If $\Pr[\Delta_0, \ldots, \Delta_r]$ is high, cipher is not random
- Leads to key recovery!



Differential cryptanalysis in a nutshell

- If $\Pr[\Delta_0, \ldots, \Delta_r]$ is high, cipher is not random
- Leads to key recovery!
- Many variants exist

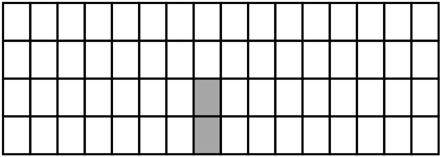


Diffusion with a CPM

- Goal: no low-weight differential trail

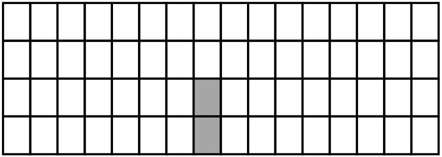
Diffusion with a CPM

- Goal: no low-weight differential trail
- How about a state like this?



Diffusion with a CPM

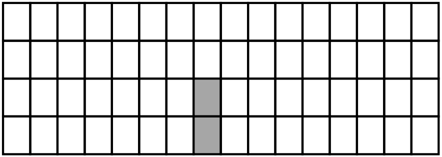
- Goal: no low-weight differential trail
- How about a state like this?



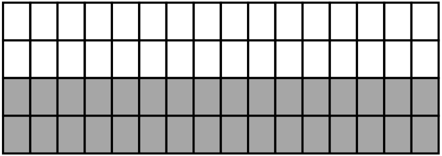
- CPM-kernel issues can be avoided by some transposition (ShiftRows-like)

Diffusion with a CPM

- Goal: no low-weight differential trail
- How about a state like this?



- CPM-kernel issues can be avoided by some transposition (ShiftRows-like)
- How about this one?



How to build a permutation with a CPM

1. Determine design goals

How to build a permutation with a CPM

1. Determine design goals
2. Pick m, n, and cell width

How to build a permutation with a CPM

1. Determine design goals
2. Pick m, n, and cell width
3. Pick ‘good’ ‘efficient’ non-linear S-box

How to build a permutation with a CPM

1. Determine design goals
2. Pick m, n, and cell width
3. Pick 'good' 'efficient' non-linear S-box
4. Consider (truncated) trails in the kernel (independent of Z)

How to build a permutation with a CPM

1. Determine design goals
2. Pick m, n, and cell width
3. Pick 'good' 'efficient' non-linear S-box
4. Consider (truncated) trails in the kernel (independent of Z)
5. Determine 'good' transposition

How to build a permutation with a CPM

1. Determine design goals
2. Pick m, n, and cell width
3. Pick 'good' 'efficient' non-linear S-box
4. Consider (truncated) trails in the kernel (independent of Z)
5. Determine 'good' transposition
6. Consider (truncated) trails outside the kernel

How to build a permutation with a CPM

1. Determine design goals
2. Pick m, n, and cell width
3. Pick ‘good’ ‘efficient’ non-linear S-box
4. Consider (truncated) trails in the kernel (independent of Z)
5. Determine ‘good’ transposition
6. Consider (truncated) trails outside the kernel
7. Determine ‘good’ Z

How to build a permutation with a CPM

1. Determine design goals
2. Pick m, n, and cell width
3. Pick 'good' 'efficient' non-linear S-box
4. Consider (truncated) trails in the kernel (independent of Z)
5. Determine 'good' transposition
6. Consider (truncated) trails outside the kernel
7. Determine 'good' Z
8. Pick 'good' round constants to beat all kinds of invariant attacks

How to build a permutation with a CPM

1. Determine design goals
2. Pick m, n, and cell width
3. Pick 'good' 'efficient' non-linear S-box
4. Consider (truncated) trails in the kernel (independent of Z)
5. Determine 'good' transposition
6. Consider (truncated) trails outside the kernel
7. Determine 'good' Z
8. Pick 'good' round constants to beat all kinds of invariant attacks
9. Do more analysis

How to build a permutation with a CPM

1. Determine design goals
2. Pick m, n, and cell width
3. Pick 'good' 'efficient' non-linear S-box
4. Consider (truncated) trails in the kernel (independent of Z)
5. Determine 'good' transposition
6. Consider (truncated) trails outside the kernel
7. Determine 'good' Z
8. Pick 'good' round constants to beat all kinds of invariant attacks
9. Do more analysis
10. Determine the number of rounds

How to build a permutation with a CPM

1. Determine design goals
2. Pick m, n, and cell width
3. Pick 'good' 'efficient' non-linear S-box
4. Consider (truncated) trails in the kernel (independent of Z)
5. Determine 'good' transposition
6. Consider (truncated) trails outside the kernel
7. Determine 'good' Z
8. Pick 'good' round constants to beat all kinds of invariant attacks
9. Do more analysis
10. Determine the number of rounds
11. Implement it

How to build a permutation with a CPM

1. Determine design goals
2. Pick m, n, and cell width
3. Pick 'good' 'efficient' non-linear S-box
4. Consider (truncated) trails in the kernel (independent of Z)
5. Determine 'good' transposition
6. Consider (truncated) trails outside the kernel
7. Determine 'good' Z
8. Pick 'good' round constants to beat all kinds of invariant attacks
9. Do more analysis
10. Determine the number of rounds
11. Implement it
12. Give it a name

Mixifer

- 16 rounds $(\iota \circ \rho \circ \pi \circ \theta \circ \gamma)$, $4 \times 16 \times 4 = 256$ bits permutation

Mixifer

- 16 rounds $(\iota \circ \rho \circ \pi \circ \theta \circ \gamma)$, $4 \times 16 \times 4 = 256$ bits permutation
- γ: rotational symmetric, $b_0 = a_1 + a_2 + a_0 a_2 + a_1 a_2 + a_1 a_2 a_3$

Mixifer

- 16 rounds $(\iota \circ \rho \circ \pi \circ \theta \circ \gamma)$, $4 \times 16 \times 4 = 256$ bits permutation
- γ: rotational symmetric, $b_0 = a_1 + a_2 + a_0 a_2 + a_1 a_2 + a_1 a_2 a_3$
- θ: Z is circulant, first row $[0, 1, 1, 0, 0, 1, 0, 0, 0, \ldots, 0]$

Mixifer

- 16 rounds $(\iota \circ \rho \circ \pi \circ \theta \circ \gamma)$, $4 \times 16 \times 4 = 256$ bits permutation
- γ: rotational symmetric, $b_0 = a_1 + a_2 + a_0 a_2 + a_1 a_2 + a_1 a_2 a_3$
- θ: Z is circulant, first row $[0, 1, 1, 0, 0, 1, 0, 0, 0, \ldots, 0]$
- π: rotate rows down

Mixifer

- 16 rounds ($\iota \circ \rho \circ \pi \circ \theta \circ \gamma$), $4 \times 16 \times 4 = 256$ bits permutation
- γ: rotational symmetric, $b_0 = a_1 + a_2 + a_0 a_2 + a_1 a_2 + a_1 a_2 a_3$
- θ: Z is circulant, first row $[0, 1, 1, 0, 0, 1, 0, 0, 0, \ldots, 0]$
- π: rotate rows down
- ρ: rotate rows cell-wise to the right by $\{14, 3, 10, 0\}$

Mixifer

- 16 rounds $(\iota \circ \rho \circ \pi \circ \theta \circ \gamma)$, $4 \times 16 \times 4 = 256$ bits permutation
- γ: rotational symmetric, $b_0 = a_1 + a_2 + a_0 a_2 + a_1 a_2 + a_1 a_2 a_3$
- θ: Z is circulant, first row $[0, 1, 1, 0, 0, 1, 0, 0, 0, \ldots, 0]$
- π: rotate rows down
- ρ: rotate rows cell-wise to the right by $\{14, 3, 10, 0\}$
- ι: add $\texttt{0xF3485763} \gg i$ in round i to even cells of top row

Mixifer

- 16 rounds $(\iota \circ \rho \circ \pi \circ \theta \circ \gamma)$, $4 \times 16 \times 4 = 256$ bits permutation
- γ: rotational symmetric, $b_0 = a_1 + a_2 + a_0 a_2 + a_1 a_2 + a_1 a_2 a_3$
- θ: Z is circulant, first row $[0, 1, 1, 0, 0, 1, 0, 0, 0, \ldots, 0]$
- π: rotate rows down
- ρ: rotate rows cell-wise to the right by $\{14, 3, 10, 0\}$
- ι: add `0xF3485763` $\gg i$ in round i to even cells of top row
- SAC after 3 rounds, full diffusion after 5

Mixifer

- 16 rounds $(\iota \circ \rho \circ \pi \circ \theta \circ \gamma)$, $4 \times 16 \times 4 = 256$ bits permutation
- γ: rotational symmetric, $b_0 = a_1 + a_2 + a_0 a_2 + a_1 a_2 + a_1 a_2 a_3$
- θ: Z is circulant, first row $[0, 1, 1, 0, 0, 1, 0, 0, 0, \ldots, 0]$
- π: rotate rows down
- ρ: rotate rows cell-wise to the right by $\{14, 3, 10, 0\}$
- ι: add $\texttt{0xF3485763} \gg i$ in round i to even cells of top row
- SAC after 3 rounds, full diffusion after 5
- In the kernel: ≥ 52 active cells after 4 rounds

Mixifer

- 16 rounds $(\iota \circ \rho \circ \pi \circ \theta \circ \gamma)$, $4 \times 16 \times 4 = 256$ bits permutation
- γ: rotational symmetric, $b_0 = a_1 + a_2 + a_0 a_2 + a_1 a_2 + a_1 a_2 a_3$
- θ: Z is circulant, first row $[0, 1, 1, 0, 0, 1, 0, 0, 0, \dots, 0]$
- π: rotate rows down
- ρ: rotate rows cell-wise to the right by $\{14, 3, 10, 0\}$
- ι: add $\texttt{0xF3485763} \gg i$ in round i to even cells of top row
- SAC after 3 rounds, full diffusion after 5
- In the kernel: ≥ 52 active cells after 4 rounds
- Outside the kernel: ≥ 46 active cells after 4 rounds (differential), DP 2^{-92}

Mixifer

- 16 rounds ($\iota \circ \rho \circ \pi \circ \theta \circ \gamma$), $4 \times 16 \times 4 = 256$ bits permutation
- γ: rotational symmetric, $b_0 = a_1 + a_2 + a_0 a_2 + a_1 a_2 + a_1 a_2 a_3$
- θ: Z is circulant, first row $[0, 1, 1, 0, 0, 1, 0, 0, 0, \ldots, 0]$
- π: rotate rows down
- ρ: rotate rows cell-wise to the right by $\{14, 3, 10, 0\}$
- ι: add $\mathtt{0xF3485763} \gg i$ in round i to even cells of top row
- SAC after 3 rounds, full diffusion after 5
- In the kernel: ≥ 52 active cells after 4 rounds
- Outside the kernel: ≥ 46 active cells after 4 rounds (differential), DP 2^{-92}
- Outside the kernel: ≥ 40 active cells after 4 rounds (linear), LP 2^{-80}

Mixifer

- 16 rounds $(\iota \circ \rho \circ \pi \circ \theta \circ \gamma)$, $4 \times 16 \times 4 = 256$ bits permutation
- γ: rotational symmetric, $b_0 = a_1 + a_2 + a_0 a_2 + a_1 a_2 + a_1 a_2 a_3$
- θ: Z is circulant, first row $[0, 1, 1, 0, 0, 1, 0, 0, 0, \ldots, 0]$
- π: rotate rows down
- ρ: rotate rows cell-wise to the right by $\{14, 3, 10, 0\}$
- ι: add $\texttt{0xF3485763} \gg i$ in round i to even cells of top row
- SAC after 3 rounds, full diffusion after 5
- In the kernel: ≥ 52 active cells after 4 rounds
- Outside the kernel: ≥ 46 active cells after 4 rounds (differential), DP 2^{-92}
- Outside the kernel: ≥ 40 active cells after 4 rounds (linear), LP 2^{-80}
- 36.69 cyc/byte on ARM Cortex-M3/M4

Thanks. . .

. . . for your attention