Shorter Linear Straight-Line Programs for MDS Matrices

Yet another XOR Count Paper

Thorsten Kranz[1], Gregor Leander[1], Ko Stoffelen[2], Friedrich Wiemer[1]

[1]Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Germany

[2]Digital Security Group, Radboud University, Nijmegen, The Netherlands

Lightweight Cryptography

Cryptographic systems might have to fulfill special constraints.

Lightweight Cryptography

Cryptographic systems might have to fulfill special constraints.

Typical Goal

Minimize the chip-area.

Linear Layers

- Matrix multiplication(s).
- Often MDS matrices.

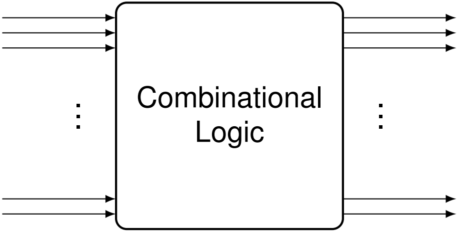
$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}, \quad x_i, y_i \in \mathbb{F}_{2^8}$$

Goal: Small round-based implementation

$$\begin{pmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}, \qquad x_i, y_i \in \mathbb{F}_{2^8}$$

Goal: Small round-based implementation

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}, \quad x_i, y_i \in \mathbb{F}_{2^8}$$



Metric: XOR count

- Implement matrix multiplication only with XOR operations.
- Use as few XORs as possible.
- Idea: Low XOR count $=$ Low chip-area

- Note: No intermediate result needs to be recomputed.

Outline

Outline

Previous Work

- **FSE 2018:** Jean, Peyrin, Sim, Tourteaux
  Optimizing Implementations of Lightweight Building Blocks
- **FSE 2017:** C. Li and Q. Wang
  Design of Lightweight Linear Diffusion Layers from Near-MDS Matrices
- **FSE 2017:** Sarkar and Syed
  Lightweight Diffusion Layer: Importance of Toeplitz Matrices
- **CRYPTO 2016:** Beierle, Kranz, Leander
  Lightweight Multiplication in $GF(2^n)$ with Applications to MDS Matrices
- **FSE 2016:** Liu and Sim
  Lightweight MDS Generalized Circulant Matrices
- **FSE 2016:** Y. Li and M. Wang
  On the Construction of Lightweight Circulant Involutory MDS Matrices
- **FSE 2015:** Sim, Khoo, Oggier, Peyrin
  Lightweight MDS Involution Matrices

Previous Work

- Searching many matrices.

Previous Work

- Searching many matrices.
  Cauchy

Previous Work

- Searching many matrices.
  Cauchy, *Vandermonde*

Previous Work

- Searching many matrices.
  Cauchy, *Vandermonde*, *Circulant*

Previous Work

- Searching many matrices.
  Cauchy, *Vandermonde*, *Circulant*, *Hadamard*

Previous Work

- Searching many matrices.
  Cauchy, *Vandermonde*, *Circulant*, *Hadamard*, *Hadamard-Cauchy*

Previous Work

- Searching many matrices.
  Cauchy, *Vandermonde*, *Circulant*, *Hadamard*, *Hadamard-Cauchy*, *Toeplitz*

Previous Work

- Searching many matrices.
  Cauchy, *Vandermonde*, *Circulant*, *Hadamard*, *Hadamard-Cauchy*, *Toeplitz*, *Arbitrary*

Previous Work

- Searching many matrices.
  Cauchy, *Vandermonde*, *Circulant*, *Hadamard*, *Hadamard-Cauchy*, *Toeplitz*, *Arbitrary*
- Optimizing element multiplication.

Counting XORs: Overhead and Fixed Cost

- The XOR count is typically split into overhead and fixed cost.

Matrix Multiplication

$$\begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & & & \\ \vdots & & \ddots & \\ \alpha_{n,1} & & & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \alpha_{i,j}, x_i, y_i \in \mathbb{F}_{2^k}$$

$$\underbrace{\sum_{i,j} \mathrm{XOR}(\alpha_{i,j})}_{\text{Overhead}} + \underbrace{n \cdot (n-1) \cdot k}_{\text{Fixed Cost}}$$

Counting XORs: Overhead and Fixed Cost

- The XOR count is typically split into overhead and fixed cost.

Matrix Multiplication

$$\begin{pmatrix} \boxed{\alpha_{1,1}} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & & & \\ \vdots & & \ddots & \\ \alpha_{n,1} & & & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} \boxed{x_1} \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \alpha_{i,j}, x_i, y_i \in \mathbb{F}_{2^k}$$

$$\underbrace{\sum_{i,j} \mathrm{XOR}(\alpha_{i,j})}_{\text{Overhead}} + \underbrace{n \cdot (n-1) \cdot k}_{\text{Fixed Cost}}$$

Counting XORs: Overhead and Fixed Cost

- The XOR count is typically split into overhead and fixed cost.

Matrix Multiplication

$$\begin{pmatrix} \alpha_{1,1} & \boxed{\alpha_{1,2}} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & & & \\ \vdots & & \ddots & \\ \alpha_{n,1} & & & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \boxed{x_2} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \alpha_{i,j}, x_i, y_i \in \mathbb{F}_{2^k}$$

$$\underbrace{\sum_{i,j} \mathrm{XOR}(\alpha_{i,j})}_{\text{Overhead}} + \underbrace{n \cdot (n-1) \cdot k}_{\text{Fixed Cost}}$$

Counting XORs: Overhead and Fixed Cost

- The XOR count is typically split into <u>overhead</u> and <u>fixed cost</u>.

Matrix Multiplication

$$\begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \dots & \boxed{\alpha_{1,n}} \\ \alpha_{2,1} & & & \\ \vdots & & \ddots & \\ \alpha_{n,1} & & & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \boxed{x_n} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \alpha_{i,j}, x_i, y_i \in \mathbb{F}_{2^k}$$

$$\underbrace{\sum_{i,j} \mathrm{XOR}(\alpha_{i,j})}_{\text{Overhead}} + \underbrace{n \cdot (n-1) \cdot k}_{\text{Fixed Cost}}$$

Counting XORs: Overhead and Fixed Cost

- The XOR count is typically split into overhead and fixed cost.

Matrix Multiplication

$$\begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & & & \\ \vdots & & \ddots & \\ \boxed{\alpha_{n,1}} & & & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} \boxed{x_1} \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \alpha_{i,j}, x_i, y_i \in \mathbb{F}_{2^k}$$

$$\underbrace{\sum_{i,j} \mathrm{XOR}(\alpha_{i,j})}_{\text{Overhead}} + \underbrace{n \cdot (n-1) \cdot k}_{\text{Fixed Cost}}$$

Counting XORs: Overhead and Fixed Cost

- The XOR count is typically split into overhead and fixed cost.

Matrix Multiplication

$$\begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & & & \\ \vdots & & \ddots & \\ \alpha_{n,1} & & & \boxed{\alpha_{n,n}} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \boxed{x_n} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \alpha_{i,j}, x_i, y_i \in \mathbb{F}_{2^k}$$

$$\underbrace{\sum_{i,j} \mathrm{XOR}(\alpha_{i,j})}_{\text{Overhead}} + \underbrace{n \cdot (n-1) \cdot k}_{\text{Fixed Cost}}$$

Counting XORs: Overhead and Fixed Cost

- The XOR count is typically split into overhead and fixed cost.

Matrix Multiplication

$$\begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & & & \\ \vdots & & \ddots & \\ \alpha_{n,1} & & & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \alpha_{i,j}, x_i, y_i \in \mathbb{F}_{2^k}$$

$$\underbrace{\sum_{i,j} \mathrm{XOR}(\alpha_{i,j})}_{\text{Overhead}} + \underbrace{n \cdot (n-1) \cdot k}_{\text{Fixed Cost}}$$

Counting XORs: Overhead and Fixed Cost

- The XOR count is typically split into overhead and fixed cost.

Matrix Multiplication

$$\begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & & & \\ \vdots & & \ddots & \\ \alpha_{n,1} & & & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \alpha_{i,j}, x_i, y_i \in \mathbb{F}_{2^k}$$

$$\underbrace{\sum_{i,j} \mathrm{XOR}(\alpha_{i,j})}_{\text{Overhead}} + \underbrace{n \cdot (n-1) \cdot k}_{\text{Fixed Cost}}$$

Previous Results

Table: Best XOR counts of previous work. Matrices in the lower half are involutory.

Dimension	S-box	XOR count
4×4	4 bit	$10 + 48$
4×4	8 bit	$10 + 96$
8×8	4 bit	$160 + 224$
8×8	8 bit	$192 + 448$
4×4	4 bit	$15 + 48$
4×4	8 bit	$30 + 96$
8×8	4 bit	$200 + 224$
8×8	8 bit	$288 + 448$

Previous Results

Table: Best XOR counts of previous work. Matrices in the lower half are involutory.

Dimension	S-box	XOR count
4×4	4 bit	58
4×4	8 bit	106
8×8	4 bit	384
8×8	8 bit	640
4×4	4 bit	63
4×4	8 bit	126
8×8	4 bit	424
8×8	8 bit	736

Outline

Local Optimization

Optimize $k \times k$ matrix over $\mathbb{F}_2$.

$$M = \begin{pmatrix} \boxed{\alpha_{1,1}} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & & & \\ \vdots & & \ddots & \\ \alpha_{n,1} & & & \alpha_{n,n} \end{pmatrix}, \quad \alpha_{i,j} \in \mathbb{F}_{2^k}$$

Global Optimization

Optimize $nk \times nk$ matrix over $\mathbb{F}_2$.

$$M = \begin{pmatrix} \boxed{\begin{matrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & & & \\ \vdots & & \ddots & \\ \alpha_{n,1} & & & \alpha_{n,n} \end{matrix}} \end{pmatrix}, \quad \alpha_{i,j} \in \mathbb{F}_{2^k}$$

Global Optimization

- **BFA 2017:** Boyar, Find, Peralta
  Low-Depth, Low-Size Circuits for Cryptographic Applications
- **ePrint 2017:** Visconti, Schiavo, Peralta
  Improved upper bounds for the expected circuit complexity of dense systems of linear equations over GF(2)
- **JoC 2013:** Boyar, Matthews, Peralta
  Logic Minimization Techniques with Applications to Cryptology
- **SAT 2010:** Fuhs, Schneider-Kamp
  Synthesizing Shortest Linear Straight-Line Programs over GF(2) Using SAT
- **IWIL 2010:** Fuhs, Schneider-Kamp
  Optimizing the AES S-Box using SAT
- **MFCS 2008:** Boyar, Matthews, Peralta
  On the Shortest Linear Straight-Line Program for Computing Linear Forms
- **ISIT 1997:** Paar
  Optimized Arithmetic for Reed-Solomon Encoders

Global Optimization

- Lots of work about implementing binary matrices with few XORs.
- Goal: Find Shortest Linear Straight-Line Programs.

Global Optimization

- Lots of work about implementing binary matrices with few XORs.
- Goal: Find Shortest Linear Straight-Line Programs.
- Equivalent to our goal!
  (Hardware implementation with lowest XOR count.)

Global Optimization

- **BFA 2017:** Boyar, Find, Peralta
  Low-Depth, Low-Size Circuits for Cryptographic Applications
- **ePrint 2017:** Visconti, Schiavo, Peralta
  Improved upper bounds for the expected circuit complexity of dense systems of linear equations over GF(2)
- **JoC 2013:** Boyar, Matthews, Peralta
  Logic Minimization Techniques with Applications to Cryptology
- **SAT 2010:** Fuhs, Schneider-Kamp
  Synthesizing Shortest Linear Straight-Line Programs over GF(2) Using SAT
- **IWIL 2010:** Fuhs, Schneider-Kamp
  Optimizing the AES S-Box using SAT
- **MFCS 2008:** Boyar, Matthews, Peralta
  On the Shortest Linear Straight-Line Program for Computing Linear Forms
- **ISIT 1997:** Paar
  Optimized Arithmetic for Reed-Solomon Encoders

Algorithm 1 (Paar 1997)

- Find most common subexpression.
- Add according computation to the program.

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_0 + a_2 + a_3 \\ a_0 + a_1 + a_2 \\ a_0 + a_1 + a_2 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

Algorithm 1 (Paar 1997)

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} {\color{red}a_0} + {\color{red}a_2} + a_3 \\ {\color{red}a_0} + a_1 + {\color{red}a_2} \\ {\color{red}a_0} + a_1 + {\color{red}a_2} + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

Algorithm 1 (Paar 1997)

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} (a_0 + a_2) + a_3 \\ (a_0 + a_2) + a_1 \\ (a_0 + a_2) + a_1 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$$x_0 = a_0 + a_2$$

Algorithm 1 (Paar 1997)

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} x_0 + a_3 \\ x_0 + a_1 \\ x_0 + a_1 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$x_0 = a_0 + a_2$

Algorithm 1 (Paar 1997)

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} x_0 + a_3 \\ x_0 + a_1 \\ x_0 + a_1 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$$x_0 = a_0 + a_2$$

Algorithm 1 (Paar 1997)

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} x_0 + a_3 \\ x_0 + a_1 \\ x_0 + a_1 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$x_0 = a_0 + a_2$

Algorithm 1 (Paar 1997)

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} x_0 + a_3 \\ (x_0 + a_1) \\ (x_0 + a_1) + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$x_0 = a_0 + a_2$

$x_1 = x_0 + a_1$

Algorithm 1 (Paar 1997)

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} x_0 + a_3 \\ x_1 \\ x_1 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$x_0 = a_0 + a_2$
$x_1 = x_0 + a_1$

Algorithm 1 (Paar 1997)

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} x_0 + a_3 \\ x_1 \\ x_1 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$$x_0 = a_0 + a_2$$
$$x_1 = x_0 + a_1$$

Algorithm 1 (Paar 1997)

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} x_3 \\ x_1 \\ x_4 \\ x_6 \end{pmatrix}$$

$$\begin{aligned} x_0 &= a_0 + a_2 \\ x_1 &= x_0 + a_1 \\ x_2 &= a_1 + a_2 \\ x_3 &= x_0 + a_3 \\ x_4 &= x_1 + a_3 \\ x_5 &= x_2 + a_3 \end{aligned}$$

Algorithm 1 (Paar 1997)

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} x_3 = a_0 + a_2 + a_3 \\ x_1 = a_0 + a_1 + a_2 \\ x_4 = a_0 + a_1 + a_2 + a_3 \\ x_5 = a_1 + a_2 + a_3 \end{pmatrix}$$

$$\begin{aligned} x_0 &= a_0 + a_2 \\ x_1 &= x_0 + a_1 = a_0 + a_1 + a_2 \\ x_2 &= a_1 + a_2 \\ x_3 &= x_0 + a_3 = a_0 + a_2 + a_3 \\ x_4 &= x_1 + a_3 = a_0 + a_1 + a_2 + a_3 \\ x_5 &= x_2 + a_3 = a_1 + a_2 + a_3 \end{aligned}$$

Algorithm 1 (Paar 1997)

Table: New XOR counts for matrices from previous work. Matrices in the lower half are involutory.

Dimension	S-box	Previously best	New results
4×4	4 bit	58	46
4×4	8 bit	106	102
8×8	4 bit	384	210
8×8	8 bit	640	464
4×4	4 bit	63	51
4×4	8 bit	126	102
8×8	4 bit	424	222
8×8	8 bit	736	620

More advanced heuristics

- There exists many follow-up work.
- More sophisticated algorithms.

More advanced heuristics

- There exists many follow-up work.
- More sophisticated algorithms.

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_0 + a_2 + a_3 \\ a_0 + a_1 + a_2 \\ a_0 + a_1 + a_2 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

More advanced heuristics

- There exists many follow-up work.
- More sophisticated algorithms.

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_0 + a_2 + a_3 \\ a_0 + a_1 + a_2 \\ a_0 + a_1 + a_2 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$x_0 = a_0 + a_1$

More advanced heuristics

- There exists many follow-up work.
- More sophisticated algorithms.

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_0 + a_2 + a_3 \\ a_0 + a_1 + a_2 \\ a_0 + a_1 + a_2 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$x_0 = a_0 + a_1$

$x_1 = x_0 + a_2 = a_0 + a_1 + a_2$

More advanced heuristics

- There exists many follow-up work.
- More sophisticated algorithms.

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_0 + a_2 + a_3 \\ a_0 + a_1 + a_2 \\ a_0 + a_1 + a_2 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$$x_0 = a_0 + a_1$$
$$x_1 = x_0 + a_2 = a_0 + a_1 + a_2$$
$$x_2 = x_1 + a_3 = a_0 + a_1 + a_2 + a_3$$

More advanced heuristics

- There exists many follow-up work.
- More sophisticated algorithms.

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_0 + a_2 + a_3 \\ a_0 + a_1 + a_2 \\ a_0 + a_1 + a_2 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$$x_0 = a_0 + a_1$$
$$x_1 = x_0 + a_2 = a_0 + a_1 + a_2$$
$$x_2 = x_1 + a_3 = a_0 + a_1 + a_2 + a_3$$
$$x_3 = x_2 + a_1 = a_0 + a_2 + a_3$$

More advanced heuristics

- There exists many follow-up work.
- More sophisticated algorithms.

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_0 + a_2 + a_3 \\ a_0 + a_1 + a_2 \\ a_0 + a_1 + a_2 + a_3 \\ a_1 + a_2 + a_3 \end{pmatrix}$$

$$\begin{aligned} x_0 &= a_0 + a_1 \\ x_1 &= x_0 + a_2 = a_0 + a_1 + a_2 \\ x_2 &= x_1 + a_3 = a_0 + a_1 + a_2 + a_3 \\ x_3 &= x_2 + a_1 = a_0 + a_2 + a_3 \\ x_4 &= x_2 + a_0 = a_1 + a_2 + a_3 \end{aligned}$$

More advanced heuristics

- There exists many follow-up work.
- More sophisticated algorithms.

Example

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} x_3 = a_0 + a_2 + a_3 \\ x_1 = a_0 + a_1 + a_2 \\ x_2 = a_0 + a_1 + a_2 + a_3 \\ x_4 = a_1 + a_2 + a_3 \end{pmatrix}$$

$$x_0 = a_0 + a_1$$
$$x_1 = x_0 + a_2 = a_0 + a_1 + a_2$$
$$x_2 = x_1 + a_3 = a_0 + a_1 + a_2 + a_3$$
$$x_3 = x_2 + a_1 = a_0 + a_2 + a_3$$
$$x_4 = x_2 + a_0 = a_1 + a_2 + a_3$$

Outline

Improved Implementations

- We applied the heuristics to

Improved Implementations

- We applied the heuristics to
  - matrices from previous work

Improved Implementations

- We applied the heuristics to
  - matrices from previous work
  - matrices known from block ciphers and hash functions

Improved Implementations

- We applied the heuristics to
  - matrices from previous work
  - matrices known from block ciphers and hash functions
- Could always find improved implementations (lower XOR count).

Improved Implementations

- We applied the heuristics to
  - matrices from previous work
  - matrices known from block ciphers and hash functions
- Could always find improved implementations (lower XOR count).
- Including AES MixColumns implementation with 97 XORs. (So far 103 was best.)

Statistical Analysis

- Analyzed different constructions
  Cauchy, Circulant, Hadamard, Toeplitz, Vandermonde, Arbitrary

Statistical Analysis

- Analyzed different constructions
  Cauchy, Circulant, Hadamard, Toeplitz, Vandermonde, Arbitrary
- No construction was superior.
- Exception: Subfield Construction

Statistical Analysis

- Analyzed different constructions
  Cauchy, Circulant, Hadamard, Toeplitz, Vandermonde, Arbitrary
- No construction was superior.
- Exception: Subfield Construction

Good strategy

Using subfield construction with best results from smaller S-box size.

New Results

Table: New best XOR counts compared to previous work. Matrices in the lower half are involutory.

Dimension	S-box	Previously best	New best
4×4	4 bit	58	36
4×4	8 bit	106	72
8×8	4 bit	384	196
8×8	8 bit	640	392
4×4	4 bit	63	42
4×4	8 bit	126	84
8×8	4 bit	424	212
8×8	8 bit	736	424

New Results

Table: New best XOR counts compared to previous work. Matrices in the lower half are involutory.

Dimension	S-box	Previously best	New best
4×4	4 bit	$10 + 48$	$-12 + 48$
4×4	8 bit	$10 + 96$	$-24 + 96$
8×8	4 bit	$160 + 224$	$-28 + 224$
8×8	8 bit	$192 + 448$	$-56 + 448$
4×4	4 bit	$15 + 48$	$-6 + 48$
4×4	8 bit	$30 + 96$	$-12 + 96$
8×8	4 bit	$200 + 224$	$-12 + 224$
8×8	8 bit	$288 + 448$	$-24 + 448$

Conclusion

Take Home Messages

- Optimize globally rather than locally.
- Stop thinking in *overhead* and *fixed cost*.
- Use the existing heuristics.
- Not necessary to restrict to matrices over finite fields.

https://github.com/pfasante/shorter_linear_slps_for_mds_matrices

Conclusion

Take Home Messages

- Optimize globally rather than locally.
- Stop thinking in *overhead* and *fixed cost*.
- Use the existing heuristics.
- Not necessary to restrict to matrices over finite fields.

`https://github.com/pfasante/shorter_linear_slps_for_mds_matrices`

Any Questions?