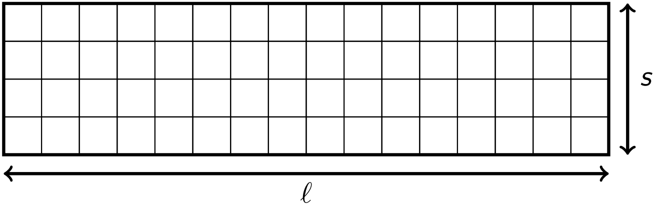
Design of a Linear Layer Optimised for Bitsliced 32-bit Implementation

Gaëtan Leurent[1], **Clara Pernot**[1,2]
[1] Inria, Paris
[2] Hensoldt France

Thursday, 28th March 2024

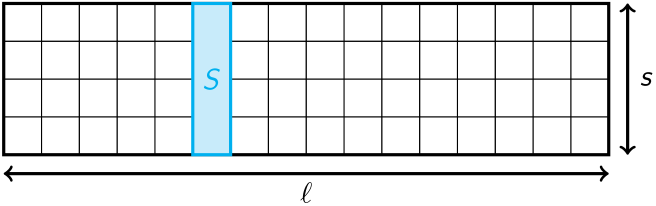
LS-designs [GLS+15]

LS-designs: a family of ciphers optimized for **bitsliced implementation**.
The state is considered as an $s \times \ell$ matrix of bits:



LS-designs [GLS+15]

LS-designs: a family of ciphers optimized for **bitsliced implementation**.
The state is considered as an $s \times \ell$ matrix of bits:

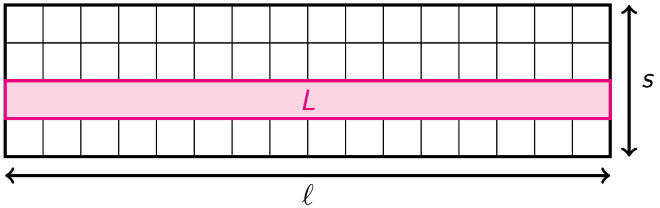


Round function:

- SBox layer: S applied ℓ times

LS-designs [GLS+15]

LS-designs: a family of ciphers optimized for **bitsliced implementation**.
The state is considered as an $s \times \ell$ matrix of bits:

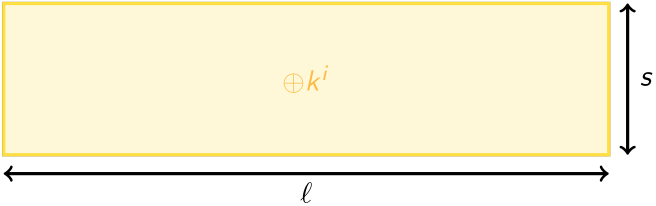


Round function:

- SBox layer: S applied ℓ times
- Linear layer Λ: L applied s times

LS-designs [GLS+15]

LS-designs: a family of ciphers optimized for **bitsliced implementation**.
The state is considered as an $s \times \ell$ matrix of bits:

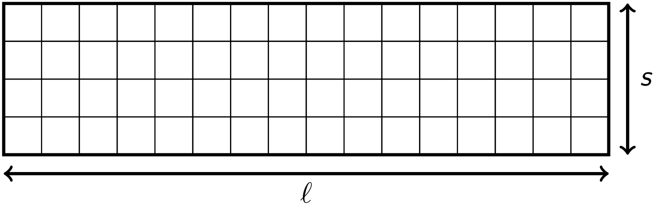


Round function:

- SBox layer: S applied ℓ times
- Linear layer Λ: L applied s times
- Key addition

LS-designs [GLS+15]

LS-designs: a family of ciphers optimized for **bitsliced implementation**.
The state is considered as an $s \times \ell$ matrix of bits:



Round function:

- SBox layer: S applied ℓ times
- Linear layer Λ: L applied s times
- Key addition

Here: $s = 4$ and $\ell = 32$.

Wide-Trail Strategy

Wide-Trail Strategy [DR01]

It's a design strategy proposed by Daemen and Rijmen:

- select SBoxes with good cryptographic properties
- design a linear layer that guarantees a large number of active SBoxes

Wide-Trail Strategy

Wide-Trail Strategy [DR01]

It's a design strategy proposed by Daemen and Rijmen:

- select SBoxes with good cryptographic properties
- design a linear layer that guarantees a large number of active SBoxes

To measure the diffusion of a linear layer Λ, we define the **branch number**:

$$\mathcal{B}(\Lambda) = \min_{x \neq 0} \left(|x| + |\Lambda(x)| \right)$$

Wide-Trail Strategy

Wide-Trail Strategy [DR01]

It's a design strategy proposed by Daemen and Rijmen:

- select SBoxes with good cryptographic properties
- design a linear layer that guarantees a large number of active SBoxes

To measure the diffusion of a linear layer Λ, we define the **branch number**:

$$\mathcal{B}(\Lambda) = \min_{x \neq 0} \left(|x| + |\Lambda(x)| \right)$$

0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0

$\xrightarrow{\Lambda}$

0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	1	0	0	0	0	0	1	0

Wide-Trail Strategy

Wide-Trail Strategy [DR01]

It's a design strategy proposed by Daemen and Rijmen:

- select SBoxes with good cryptographic properties
- design a linear layer that guarantees a large number of active SBoxes

To measure the diffusion of a linear layer Λ, we define the **branch number**:

$$\mathcal{B}(\Lambda) = \min_{x \neq 0} \left(|x| + |\Lambda(x)| \right)$$

0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0

$$\xrightarrow{\Lambda}$$

0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	1	0	0	0	0	0	1	0

Wide-Trail Strategy

Wide-Trail Strategy [DR01]

It's a design strategy proposed by Daemen and Rijmen:

- select SBoxes with good cryptographic properties
- design a linear layer that guarantees a large number of active SBoxes

To measure the diffusion of a linear layer Λ, we define the **branch number**:

$$\mathcal{B}(\Lambda) = \min_{x \neq 0} \left(|x| + |\Lambda(x)| \right)$$

$$\underbrace{\begin{array}{|cccccccccccccccc|} 0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&0 \\ 0&0&1&1&0&0&0&1&0&0&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0 \end{array}}_{|x| = 7} \xrightarrow{\Lambda} \underbrace{\begin{array}{|cccccccccccccccc|} 0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&1 \\ 1&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&0&1&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&1&1&0&0&0&0&0&1&0 \end{array}}_{|\Lambda(x)| = 10}$$

Wide-Trail Strategy

Wide-Trail Strategy [DR01]

It's a design strategy proposed by Daemen and Rijmen:

- select SBoxes with good cryptographic properties
- design a linear layer that guarantees a large number of active SBoxes

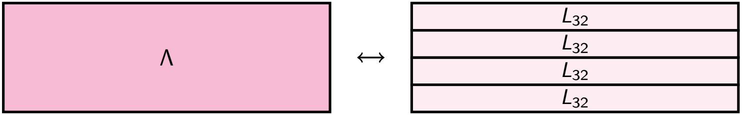
To measure the diffusion of a linear layer Λ, we define the **branch number**:

$$\mathcal{B}(\Lambda) = \min_{x \neq 0} \left(|x| + |\Lambda(x)| \right)$$

- Any non-trivial differential characteristics in two consecutive rounds has at least $\mathcal{B}(\Lambda)$ active SBoxes.
- It allows to derive security bounds.

Linear layers

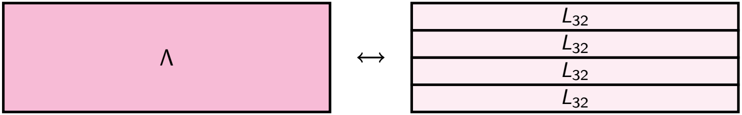
LS-designs:



- $\mathcal{B}(\Lambda) = \mathcal{B}(L_{32})$

Linear layers

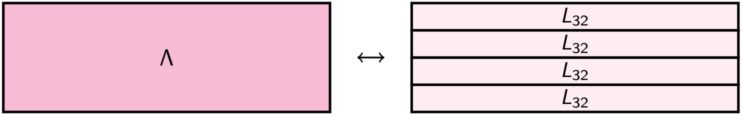
LS-designs:



- $\mathcal{B}(\Lambda) = \mathcal{B}(L_{32})$
- $\mathcal{B}(L_{32}) = 12$ with the best known code

Linear layers

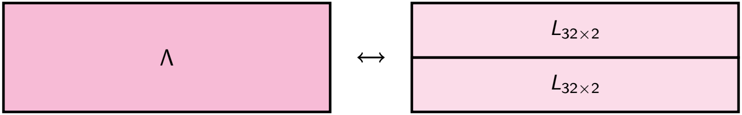
LS-designs:



- $\mathcal{B}(\Lambda) = \mathcal{B}(L_{32})$
- $\mathcal{B}(L_{32}) = 12$ with the best known code

Spook:

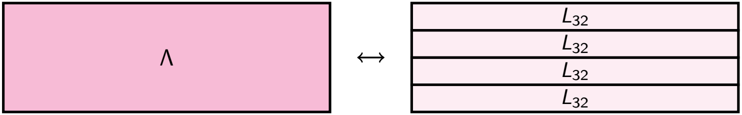
The linear transformation is defined on **two words simultaneously**:



- $\mathcal{B}(\Lambda) = \mathcal{B}(L_{32\times 2})$

Linear layers

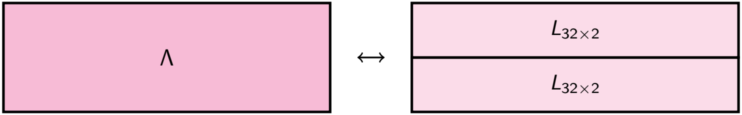
LS-designs:



- $\mathcal{B}(\Lambda) = \mathcal{B}(L_{32})$
- $\mathcal{B}(L_{32}) = 12$ with the best known code

Spook:

The linear transformation is defined on **two words simultaneously**:



- $\mathcal{B}(\Lambda) = \mathcal{B}(L_{32\times 2})$
- $\mathcal{B}(L_{32\times 2}) =$ **16** in Spook

Efficient Implementation using Rotations and XORs

Algorithm L_{32} LBox

Input: x

$a \leftarrow x \oplus \mathrm{rot}(x, 1)$

$b \leftarrow a \oplus \mathrm{rot}(a, 4)$

$a \leftarrow b \oplus \mathrm{rot}(a, 9)$

$b \leftarrow a \oplus \mathrm{rot}(x, 3)$

Return $b \oplus \mathrm{rot}(a, 6)$

$$L_{32} = \begin{bmatrix}
1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1 \\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1 \\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0 \\
0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0 \\
0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0 \\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1 \\
1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0 \\
0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1 \\
1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0 \\
0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1 \\
1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1 \\
1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1 \\
1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1 \\
1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1 \\
1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0 \\
0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1 \\
1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix}$$

- $\mathcal{B}(L_{32}) = 12$
- Corresponds to circulant matrices

Efficient Implementation using Rotations and XORs

Algorithm L_{32} LBox

Input: x

$a \leftarrow x \oplus \mathrm{rot}(x, 1)$

$b \leftarrow a \oplus \mathrm{rot}(a, 4)$

$a \leftarrow b \oplus \mathrm{rot}(a, 9)$

$b \leftarrow a \oplus \mathrm{rot}(x, 3)$

Return $b \oplus \mathrm{rot}(a, 6)$

$$L_{32} = \begin{bmatrix}
1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0\\
0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1\\
1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0\\
0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1\\
1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0\\
0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1\\
1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1\\
1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1\\
1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1\\
1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1\\
1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0\\
0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix}$$

- $\mathcal{B}(L_{32}) = 12$
- Corresponds to circulant matrices
- All circulant matrices can be implemented using Rot and XOR
  $\rightarrow$ Goal: minimize the number of Rot and XOR

Efficient Implementation using Rotations and XORs

Algorithm L_{32} LBox

Input: x

$a \leftarrow x \oplus \mathrm{rot}(x, 1)$

$b \leftarrow a \oplus \mathrm{rot}(a, 4)$

$a \leftarrow b \oplus \mathrm{rot}(a, 9)$

$b \leftarrow a \oplus \mathrm{rot}(x, 3)$

Return $b \oplus \mathrm{rot}(a, 6)$

$$L_{32} = \begin{bmatrix}
1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1 \\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1 \\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0 \\
0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0 \\
0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0 \\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1 \\
1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0 \\
0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1 \\
1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0 \\
0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1 \\
1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1 \\
1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1 \\
1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1 \\
1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1 \\
1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0 \\
0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1 \\
1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix}$$

- $\mathcal{B}(L_{32}) = 12$
- Corresponds to circulant matrices
- All circulant matrices can be implemented using Rot and XOR
  $\rightarrow$ Goal: minimize the number of Rot and XOR
- The inverse can also be implemented using Rot and XOR

Efficient Implementation using Rotations and XORs

Algorithm L_{32} LBox

Input: x

$a \leftarrow x \oplus \mathrm{rot}(x, 1)$

$b \leftarrow a \oplus \mathrm{rot}(a, 4)$

$a \leftarrow b \oplus \mathrm{rot}(a, 9)$

$b \leftarrow a \oplus \mathrm{rot}(x, 3)$

Return $b \oplus \mathrm{rot}(a, 6)$

$$L_{32} = \begin{bmatrix}
1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0&1\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&0\\
0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0&1\\
1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1&0\\
0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0&1\\
1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&0\\
0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1\\
1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1\\
1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1\\
1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1\\
1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1\\
1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0\\
0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
1&0&1&1&1&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix}$$

- $\mathcal{B}(L_{32}) = 12$
- Corresponds to circulant matrices
- All circulant matrices can be implemented using Rot and XOR
  $\rightarrow$ Goal: minimize the number of Rot and XOR
- The inverse can also be implemented using Rot and XOR
- $\mathcal{B} = \mathcal{B}_{diff} = \mathcal{B}_{lin}$

Linear layer in Spook [BBB+20]

Algorithm $L_{2\times 32}$ LBox

Input: (x, y)

$a \leftarrow x \oplus \mathrm{rot}(x, 12)$
$b \leftarrow y \oplus \mathrm{rot}(y, 12)$
$a \leftarrow a \oplus \mathrm{rot}(a, 3)$
$b \leftarrow b \oplus \mathrm{rot}(b, 3)$
$a \leftarrow a \oplus \mathrm{rot}(x, 17)$
$b \leftarrow b \oplus \mathrm{rot}(y, 17)$
$c \leftarrow a \oplus \mathrm{rot}(a, 31)$
$d \leftarrow b \oplus \mathrm{rot}(b, 31)$
$a \leftarrow a \oplus \mathrm{rot}(d, 26)$
$b \leftarrow b \oplus \mathrm{rot}(c, 25)$
$a \leftarrow a \oplus \mathrm{rot}(c, 15)$
$b \leftarrow b \oplus \mathrm{rot}(d, 15)$
$b \leftarrow \mathrm{rot}(b, 1)$

Return (b, a)

Algorithm $L_{2\times 32}$ LBox inverse

Input: (x, y)

$a \leftarrow x \oplus \mathrm{rot}(x, 25)$
$b \leftarrow y \oplus \mathrm{rot}(y, 25)$
$c \leftarrow x \oplus \mathrm{rot}(a, 31)$
$d \leftarrow y \oplus \mathrm{rot}(b, 31)$
$c \leftarrow c \oplus \mathrm{rot}(a, 20)$
$d \leftarrow d \oplus \mathrm{rot}(b, 20)$
$a \leftarrow c \oplus \mathrm{rot}(c, 31)$
$b \leftarrow d \oplus \mathrm{rot}(d, 31)$
$c \leftarrow c \oplus \mathrm{rot}(b, 26)$
$d \leftarrow d \oplus \mathrm{rot}(a, 25)$
$a \leftarrow a \oplus \mathrm{rot}(c, 17)$
$b \leftarrow b \oplus \mathrm{rot}(d, 17)$
$a \leftarrow \mathrm{rot}(a, 15)$
$b \leftarrow \mathrm{rot}(b, 16)$

Return (b, a)

Linear layer in Spook [BBB+20]

Algorithm $L_{2\times 32}$ LBox

Input: (x, y)

$a \leftarrow x \oplus \text{rot}(x, 12)$
$b \leftarrow y \oplus \text{rot}(y, 12)$
$a \leftarrow a \oplus \text{rot}(a, 3)$
$b \leftarrow b \oplus \text{rot}(b, 3)$
$a \leftarrow a \oplus \text{rot}(x, 17)$
$b \leftarrow b \oplus \text{rot}(y, 17)$
$c \leftarrow a \oplus \text{rot}(a, 31)$
$d \leftarrow b \oplus \text{rot}(b, 31)$
$a \leftarrow a \oplus \text{rot}(d, 26)$
$b \leftarrow b \oplus \text{rot}(c, 25)$
$a \leftarrow a \oplus \text{rot}(c, 15)$
$b \leftarrow b \oplus \text{rot}(d, 15)$
$b \leftarrow \text{rot}(b, 1)$

Return (b, a)

1 step = 1 Rot/XOR per word

Algorithm $L_{2\times 32}$ LBox inverse

Input: (x, y)

$a \leftarrow x \oplus \text{rot}(x, 25)$
$b \leftarrow y \oplus \text{rot}(y, 25)$
$c \leftarrow x \oplus \text{rot}(a, 31)$
$d \leftarrow y \oplus \text{rot}(b, 31)$
$c \leftarrow c \oplus \text{rot}(a, 20)$
$d \leftarrow d \oplus \text{rot}(b, 20)$
$a \leftarrow c \oplus \text{rot}(c, 31)$
$b \leftarrow d \oplus \text{rot}(d, 31)$
$c \leftarrow c \oplus \text{rot}(b, 26)$
$d \leftarrow d \oplus \text{rot}(a, 25)$
$a \leftarrow a \oplus \text{rot}(c, 17)$
$b \leftarrow b \oplus \text{rot}(d, 17)$
$a \leftarrow \text{rot}(a, 15)$
$b \leftarrow \text{rot}(b, 16)$

Return (b, a)

Efficient implementation of L and L^{-1} in Spook

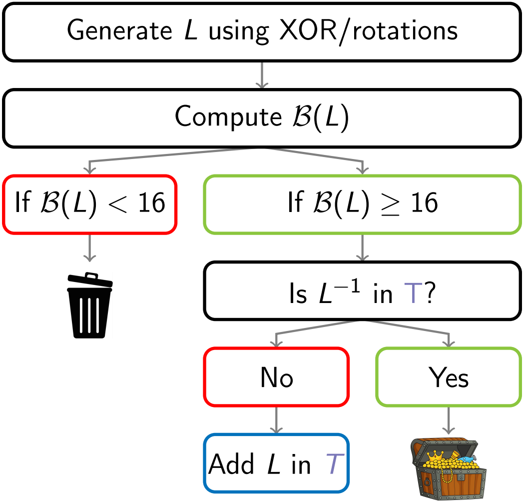
Idea: find L_1, L_2 such that $L_1 = L_2^{-1}$

L_1, L_2: linear layers with efficient implementations

Efficient implementation of L and L^{-1} in Spook

Idea: find L_1, L_2 such that $L_1 = L_2^{-1}$

L_1, L_2: linear layers with efficient implementations



Efficient implementation of L and L^{-1} in Spook

Idea: find L_1, L_2 such that $L_1 = L_2^{-1}$

L_1, L_2: linear layers with efficient implementations

Generate L using XOR/rotations

Compute $\mathcal{B}(L)$

If $\mathcal{B}(L) < 16$

If $\mathcal{B}(L) \geq 16$

Is L^{-1} in T?

No

Add L in T

Yes

T

001...110	$(r_i, x_i)_{i=0,\dots,6}$
010...011	$(r_i, x_i)_{i=0,\dots,6}$
...	...
101...000	$(r_i, x_i)_{i=0,\dots,6}$
111...101	$(r_i, x_i)_{i=0,\dots,6}$

Our work

Goal: obtain a linear layer operating on **3 or 4 32-bit words** with:

Our work

Goal: obtain a linear layer operating on **3 or 4 32-bit words** with:

- a **higher branch number**
- an **efficient implementation** of L and L^{-1}

Our work

Goal: obtain a linear layer operating on **3 or 4 32-bit words** with:

- a **higher branch number**
  - Naive computation of a 128-bit linear transformation: 2^{128} operations.
  - Is there a more efficient method?

- an **efficient implementation** of L and L^{-1}

Our work

Goal: obtain a linear layer operating on **3 or 4 32-bit words** with:

- a **higher branch number**
  - Naive computation of a 128-bit linear transformation: 2^{128} operations.
  - Is there a more efficient method?
- an **efficient implementation** of L and L^{-1}
  - collisions are used in Spook: the search space is too big for 128 bits!
  - Is there a more efficient method?

Table of contents

How to compute efficiently the branch number?

Reminder:

$$\mathcal{B}(\Lambda) = \min_{x \neq 0} \left(|x| + |\Lambda(x)| \right)$$

How to compute efficiently the branch number?

Reminder:

$$\mathcal{B}(\Lambda) = \min_{x \neq 0} \left(|x| + |\Lambda(x)|\right)$$

Property

$\mathcal{B}(\Lambda)$ is equal to the **minimal distance** of the code with **codewords** $x \| \Lambda(x)$ for $x \in (\{0,1\}^s)^\ell$.

How to compute efficiently the branch number?

Reminder:

$$\mathcal{B}(\Lambda) = \min_{x \neq 0} \left(|x| + |\Lambda(x)| \right)$$

Property

$\mathcal{B}(\Lambda)$ is equal to the **minimal distance** of the code with **codewords** $x \| \Lambda(x)$ for $x \in (\{0,1\}^s)^\ell$.

We use an **Information Set Decoding algorithm** to compute $\mathcal{B}(\Lambda)$:

- Derived from Prange's algorithm [Pra62]
- Find the non-zero codeword with the lowest possible weight.
- Probabilistic algorithm.

The Information Set Decoding algorithm

Small example on $(\mathbb{F}_2)^8$ corresponding to L_8:

1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1
0	1	0	0	0	0	0	0	1	1	0	0	1	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1
0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0
0	0	0	0	1	0	0	0	1	1	0	1	1	1	0	0
0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	1
0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0
0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	1

Input of L_8 Output of L_8

The Information Set Decoding algorithm

Small example on $(\mathbb{F}_2)^8$ corresponding to L_8:

We bet that there is a weight 1 on these columns.

1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1
0	1	0	0	0	0	0	0	1	1	0	0	1	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1
0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0
0	0	0	0	1	0	0	0	1	1	0	1	1	1	0	0
0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	1
0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0
0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	1

Input of L_8 Output of L_8

Repeat:

1. Select an information set

The Information Set Decoding algorithm

Small example on $(\mathbb{F}_2)^8$ corresponding to L_8:

1	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0
0	1	0	0	0	1	0	1	0	0	0	0	0	0	1	1
0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	1	0	0	1	1	1	0
0	0	0	0	1	0	0	1	0	0	1	0	1	0	0	1
0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0
0	0	0	1	1	1	0	1	0	0	0	0	1	0	0	1

Information set

Repeat:

1. Select an information set
2. Put the columns of the information set at the left

The Information Set Decoding algorithm

Small example on $(\mathbb{F}_2)^8$ corresponding to L_8:

1	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1
0	1	0	0	0	0	0	0	1	1	0	1	0	1	0	1
0	0	1	0	0	0	0	0	0	1	0	0	1	1	1	0
0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	1
0	0	0	0	1	0	0	0	1	0	1	0	0	1	1	0
0	0	0	0	0	1	0	0	0	1	0	1	1	0	0	0
0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0

Information set

Repeat:

1. Select an information set
2. Put the columns of the information set at the left
3. Do a Gauss reduction

The Information Set Decoding algorithm

Small example on $(\mathbb{F}_2)^8$ corresponding to L_8:

1	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1
0	1	0	0	0	0	0	0	1	1	0	1	0	1	0	1
0	0	1	0	0	0	0	0	0	1	0	0	1	1	1	0
0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	1
0	0	0	0	1	0	0	0	1	0	1	0	0	1	1	0
0	0	0	0	0	1	0	0	0	1	0	1	1	0	0	0
0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0

Information set

Repeat:

1. Select an information set
2. Put the columns of the information set at the left
3. Do a Gauss reduction
4. Look at the weight of the lines

The Information Set Decoding algorithm

We assume that there is a word W of weight **w**

We find W if it has weight 1 in the information set

W =	1	1	0	1	0	0	0	0	0	1	0	0	1	0	1	0

The Information Set Decoding algorithm

We assume that there is a word W of weight **w**
We find W if it has weight 1 in the information set

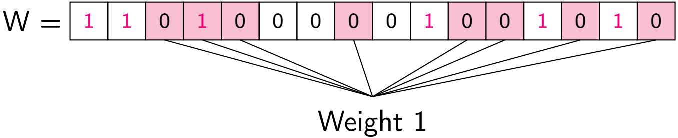
$$W = \boxed{1\;1\;0\;1\;0\;0\;0\;0\;0\;1\;0\;0\;1\;0\;1\;0}$$

Weight $\mathbf{w}/2 = 3$

The Information Set Decoding algorithm

We assume that there is a word W of weight **w**

We find W if it has weight 1 in the information set



The Information Set Decoding algorithm

We assume that there is a word W of weight $\mathbf{w}$
We find W if it has weight 1 in the information set

W =	1	1	0	1	0	0	0	0	0	1	0	0	1	0	1	0

This happens with probability:

$$p = \frac{\binom{\ell}{\mathbf{w}-1} \times \binom{\ell}{1}}{\binom{2\ell}{\mathbf{w}}}$$

The Information Set Decoding algorithm

We assume that there is a word W of weight **w**
We find W if it has weight 1 in the information set

W =

1	1	0	1	0	0	0	0	0	1	0	0	1	0	1	0

This happens with probability:

$$p = \frac{\binom{\ell}{\mathbf{w}-1} \times \binom{\ell}{1}}{\binom{2\ell}{\mathbf{w}}}$$

$\Rightarrow$ We can also detect a weight of 2 by considering all the pairs of 2 lines:

- p : $\nearrow$
- Time complexity : $\approx$
  (because the time complexity is dominated by the Gaussian Reduction)

The Information Set Decoding algorithm

We need to adapt this algorithm to our context:

$$\mathbf{w} = 20 \qquad \ell = 32 \qquad \text{4 words}$$

With 2^{25} iterations that costs $2^{16.8}$ the probability of failing to find W if it exists is 2^{-604}:

Method	Time Complexity	Success Probability
Naive	2^{128}	1
ISD	$2^{41.8}$	1 - 2^{-604}

Table of contents

Efficient implementation of L and L^{-1}

- The method used in Spook uses collisions
- Here, the search space is too big...
  - Sometimes, we only require an efficient implementation of L
    Example: CounTeR Mode
  - Otherwise, we use a heuristic algorithm to find an efficient implementation of the inverse

Efficient implementation of L and L^{-1}

Random circulant matrix:

For 4 32-bit words, after 2^{18} tests: the best $\mathcal{B}$ is **21**

Efficient implementation of L and L^{-1}

Random circulant matrix:

For 4 32-bit words, after 2^{18} tests: the best $\mathcal{B}$ is **21**

Efficient matrix:

Our strategy:

1. generate candidates based on 6 steps of XOR and rotations
2. keep only candidates with $\mathcal{B} = 21$
3. look for an efficient implementation of the inverse
4. keep the candidate whose inverse has the most efficient implementation

Table of contents

Results

L	w	Branch number	$c(L)$	$c(L^{-1})$	Ref
L_{32}	1	12	5	5	LS-designs [Leu19]
$L_{32\times 2}$	2	16	6	6	Spook [BBB+20]
$L_{32\times 3}$	3	19	6	13	New
$L_{32\times 4}$	4	21	6	18	New

Linear transformations based on XORs and rotations

$c(L)$: number of XORs per 32-bit word in our implementation

Conclusion

→ Extension of the work done in the LS-designs and Spook

→ Linear layer with branch number 21 over 128 bits with the same cost as Spook (whose branch number is 16)

→ Illustration of the interactions between different fields of cryptography: use algorithm from coding theory

Thank you!