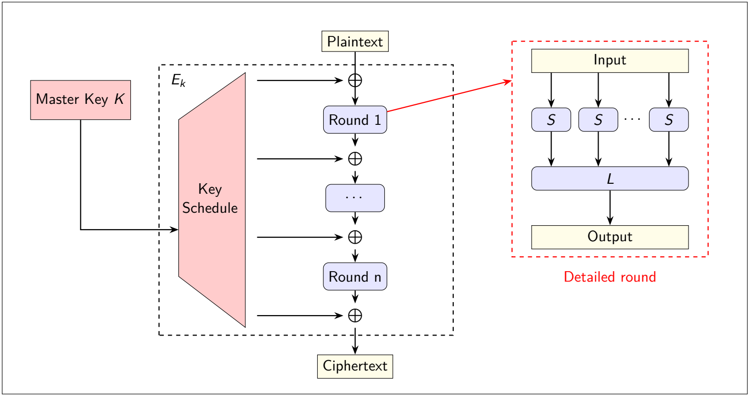
# New Algorithms to Find Lightweight (AND,XOR) Implementations of S-boxes

**Marie Bolzer**, Sébastien Duval, Marine Minier

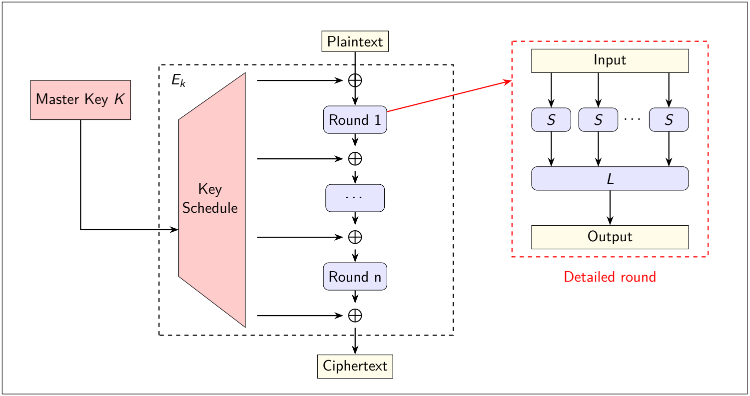
March 20, 2026

Séminaire de cryptograhie de l'université de Rennes

# Context: Substitution-Permutation Network

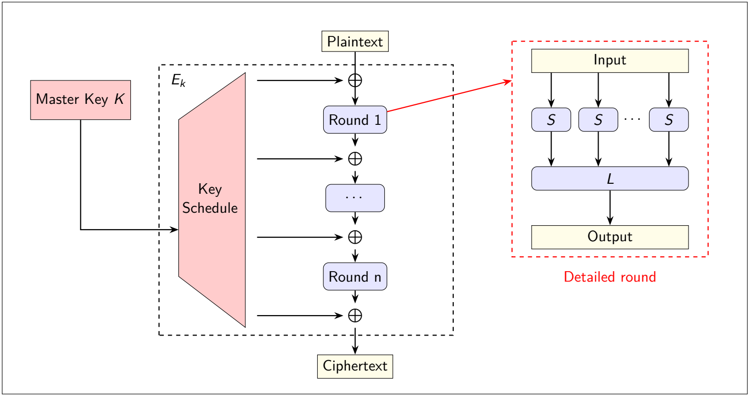


# Context: Substitution-Permutation Network



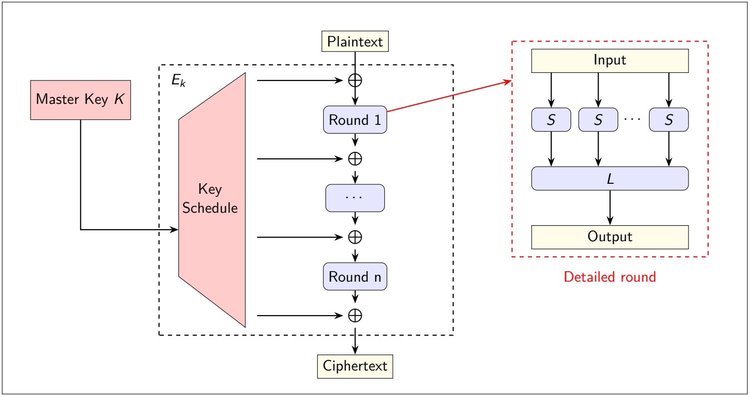

Cryptographic implementations must be **protected** against physical attacks.

# Context: Substitution-Permutation Network




Cryptographic implementations must be **protected** against physical attacks. A usual countermeasure is **masking**.

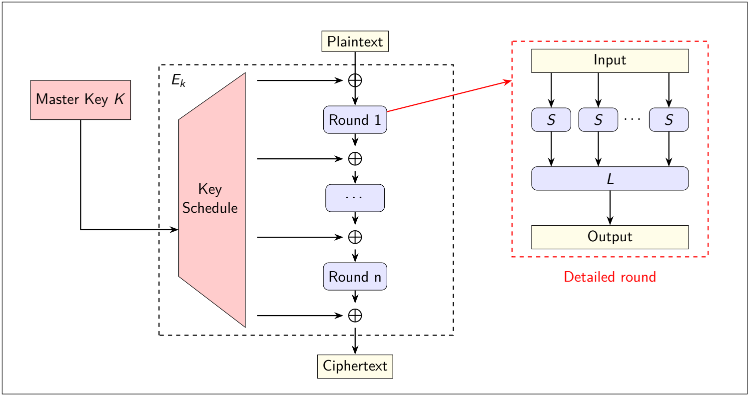
# Context: Substitution-Permutation Network




Cryptographic implementations must be **protected** against physical attacks. A usual countermeasure is **masking**.

Problem: Masking is very expensive for non-linear operations (AND, OR).

# Context: Substitution-Permutation Network




Cryptographic implementations must be **protected** against physical attacks. A usual countermeasure is **masking**.

Problem: Masking is very expensive for non-linear operations (AND, OR).

→ The S-box is then the costliest part of the cipher.

# Context: choice of metrics

We focused on two main cost metrics for our implementation:

- **multiplicative depth**, largest number of multiplication operations on any path from input variables to output variables of the circuit

# Context: choice of metrics

We focused on two main cost metrics for our implementation:

- **multiplicative depth**, largest number of multiplication operations on any path from input variables to output variables of the circuit

  $\Rightarrow$ main factor of the **latency** of the circuit,

# Context: choice of metrics

We focused on two main cost metrics for our implementation:

- **multiplicative depth**, largest number of multiplication operations on any path from input variables to output variables of the circuit

  $\Rightarrow$ main factor of the **latency** of the circuit,

- **multiplicative complexity**, total number of multiplications in the implementation.

# Context: choice of metrics

We focused on two main cost metrics for our implementation:

- **multiplicative depth**, largest number of multiplication operations on any path from input variables to output variables of the circuit

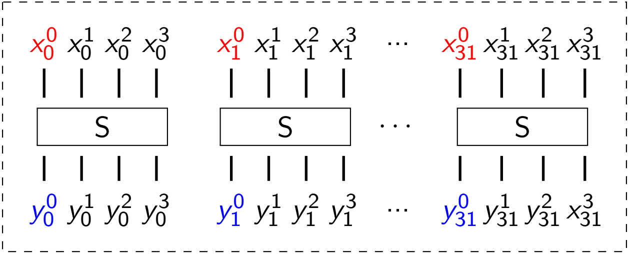
  $\Rightarrow$ main factor of the **latency** of the circuit,

- **multiplicative complexity**, total number of multiplications in the implementation.

  $\Rightarrow$ main factor of the circuit **size**.

# Bitsliced Implementation

**Idea** : Use only bitwise operations.
**Interest** : Process the same S-box several times in parallel




$$x_0 = (x_0^0\ x_1^0\ x_2^0\ \ldots\ x_{31}^0) \rightarrow \mathbf{S} \rightarrow y_0 = (y_0^0\ y_1^0\ y_2^0\ \ldots\ y_{31}^0)$$

# S-boxes and Algebraic Normal Form (ANF)

$$S : \mathbb{F}_2^n \to \mathbb{F}_2^m, \quad S(x_1, \ldots, x_n) = (y_1, \ldots, y_m),$$

can be described by its **Algebraic Normal Form** (ANF):

$$y_j = \bigoplus_{I \subseteq \{1,\ldots,n\}} a_I \prod_{i \in I} x_i, \quad a_I \in \mathbb{F}_2.$$

**Look-Up-Table (LUT)**:

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $S(x)$ | 0 | 1 | 3 | 4 | 5 | 6 | 7 | 2 |

**Multivariate representation (ANF):**

$$S : \mathbb{F}_2^3 \to \mathbb{F}_2^3$$
$(x_0, x_1, x_2) \mapsto (y_0, y_1, y_2)$ where
$$y_0 = x_0 \oplus x_1 \oplus x_2 \oplus x_1 x_2$$
$$y_1 = x_1 \oplus x_0 x_1 \oplus x_0 x_2$$
$$y_2 = x_0 x_1 \oplus x_2$$

$\to$ ANF is already a bitsliced implementation.

# S-boxes and Algebraic Normal Form (ANF)

$$S : \mathbb{F}_2^n \to \mathbb{F}_2^m, \quad S(x_1, \ldots, x_n) = (y_1, \ldots, y_m),$$

can be described by its **Algebraic Normal Form** (ANF):

$$y_j = \bigoplus_{I \subseteq \{1,\ldots,n\}} a_I \prod_{i \in I} x_i, \quad a_I \in \mathbb{F}_2.$$

**Polynomial interpretation:**

- $y_j$ can be seen as a polynomial in the ring $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_i^2 + x_i \rangle$.
- Then:

$$\text{XOR} \Leftrightarrow \text{addition in } \mathbb{F}_2, \qquad \text{AND} \Leftrightarrow \text{multiplication in } \mathbb{F}_2.$$

# S-boxes and Algebraic Normal Form (ANF)

$$S : \mathbb{F}_2^n \to \mathbb{F}_2^m, \quad S(x_1, \ldots, x_n) = (y_1, \ldots, y_m),$$

can be described by its **Algebraic Normal Form** (ANF):

$$y_j = \bigoplus_{I \subseteq \{1,\ldots,n\}} a_I \prod_{i \in I} x_i, \quad a_I \in \mathbb{F}_2.$$

**Polynomial interpretation:**

- $y_j$ can be seen as a polynomial in the ring $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_i^2 + x_i \rangle$.
- Then:

$$\text{XOR} \Leftrightarrow \text{addition in } \mathbb{F}_2, \qquad \text{AND} \Leftrightarrow \text{multiplication in } \mathbb{F}_2.$$

$\Rightarrow$ **Minimising AND gates = minimising polynomial multiplications.**

# Formalisation of the problem

## Specifications required in the implementation

- Use only bitwise operations (AND, XOR, NOT)
- Minimise multiplicative depth and complexity (AND gates)

# Formalisation of the problem

**Specifications required in the implementation**

- Use only bitwise operations (AND, XOR, NOT)
- Minimise multiplicative depth and complexity (AND gates)

We start from:

$$\boxed{\begin{array}{l} y_0 = x_0x_1 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \\ y_1 = x_0x_1 \oplus x_1x_3 \end{array}}$$

6 AND - 4 XOR

# Formalisation of the problem

**Specifications required in the implementation**

- Use only bitwise operations (AND, XOR, NOT)
- Minimise multiplicative depth and complexity (AND gates)

We start from:

$$\begin{array}{|l|}\hline y_0 = x_0x_1 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \\ y_1 = x_0x_1 \oplus x_1x_3 \\ \hline\end{array} \longrightarrow$$

6 AND - 4 XOR

We want to obtain:

$$\begin{array}{|l|}\hline l_0 = x_0 \oplus x_3 \\ q_0 = x_1 \times l_0 \\ l_1 = x_1 \oplus x_3 \\ q_1 = x_2 \times l_1 \\ y_1 = q_0 \\ y_0 = y_1 \oplus q_1 \\ \hline\end{array}$$

2 AND - 3 XOR

# Formalisation of the problem

**Specifications required in the implementation**

- Use only bitwise operations (AND, XOR, NOT)
- Minimise multiplicative depth and complexity (AND gates)

We start from:

$$\boxed{\begin{aligned} y_0 &= x_0x_1 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \\ y_1 &= x_0x_1 \oplus x_1x_3 \end{aligned}}$$

6 AND - 4 XOR

$\longrightarrow$

We want to obtain:

$$\boxed{\begin{aligned} l_0 &= x_0 \oplus x_3 \\ q_0 &= x_1 \times l_0 \\ l_1 &= x_1 \oplus x_3 \\ q_1 &= x_2 \times l_1 \\ y_1 &= q_0 \\ y_0 &= y_1 \oplus q_1 \end{aligned}}$$

2 AND - 3 XOR

**Objective:** To obtain a tool that automates this process and gives an implementation of a given S-box having the **best multiplicative depth** and **few multiplications**.

# Existing tools

Several tools exist to implement small functions. There are 2 major algorithmic approaches:

**Depth-First-Search in graphs:**

• Defined in 2011 by Ullrich.
• Used in LIGHTER in 2017 and then in PEIGEN in 2019.

**SAT-solvers:**

• Defined in 2016 by Stoffelen.
• Refined in 2020, 2023 then 2024.

# Existing tools

Several tools exist to implement small functions. There are 2 major algorithmic approaches:

**Depth-First-Search in graphs:**

• Defined in 2011 by Ullrich.
• Used in LIGHTER in 2017 and then in PEIGEN in 2019.

**SAT-solvers:**

• Defined in 2016 by Stoffelen.
• Refined in 2020, 2023 then 2024.

**Limitations of these tools:** Functions operating on less than 5 bits, sometimes 6 bits for quadratic functions, or to very simple functions. (High computing time)

# Our Contributions

**Objectives with regards to existing tools:**

- Design a *dedicated algorithm* for finding implementations with **optimal multiplicative depth** and **low multiplicative complexity**.
- *Reduce execution time* for more complex functions in order to *extend applicability* to functions on a larger number of bits.

# Our Contributions

**Objectives with regards to existing tools:**

- Design a *dedicated algorithm* for finding implementations with **optimal multiplicative depth** and **low multiplicative complexity**.
- *Reduce execution time* for more complex functions in order to *extend applicability* to functions on a larger number of bits.

**Achieved results:**

- Development of an efficient approach for the *quadratic case* — results published in IEEE TCAS.
- Extension of the method to *higher-degree functions* — accepted to CHES.

# Some intuitions

**First idea:** Try to find a lightweight implementation with pen, paper and coffee.

$$\boxed{\begin{aligned} y_0 &= x_0x_1 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \\ y_1 &= x_0x_1 \oplus x_1x_3 \end{aligned}} \longrightarrow \boxed{\begin{aligned} l_0 &= x_0 \oplus x_3 \\ q_0 &= x_1 \times l_0 \\ l_1 &= x_1 \oplus x_3 \\ q_1 &= x_2 \times l_1 \\ y_1 &= q_0 \\ y_0 &= y_1 \oplus q_1 \end{aligned}}$$

$\rightarrow$ It seems natural to start from the ANF, look for patterns which appear in several output bits and iteratively factorise the ANF.

# Some intuitions

**First idea:** Try to find a lightweight implementation with pen, paper and coffee.

$$\boxed{\begin{aligned} y_0 &= x_0x_1 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \\ y_1 &= x_0x_1 \oplus x_1x_3 \end{aligned}} \longrightarrow \boxed{\begin{aligned} l_0 &= x_0 \oplus x_3 \\ q_0 &= x_1 \times l_0 \\ l_1 &= x_1 \oplus x_3 \\ q_1 &= x_2 \times l_1 \\ y_1 &= q_0 \\ y_0 &= y_1 \oplus q_1 \end{aligned}}$$

$\rightarrow$ It seems natural to start from the ANF, look for patterns which appear in several output bits and iteratively factorise the ANF.

**Objective:** Make this algorithm *feasible* for a computer.

# PART 2: DESCRIPTION OF THE ALGORITHM FOR THE QUADRATIC CASE

# Notations: Useful polynomial subsets

**Degree of a monomial:** number of variables that appear in the monomial.
**Degree of a polynomial:** maximum degree of its monomials.

- $\mathcal{L}_n$: set of **linear** polynomials (degree $\leq 1$)
- $\mathcal{Q}_n$: set of **quadratic** polynomials (degree $\leq 2$)
- $\mathcal{Q}_{1,n}$: non-linear truncations of $\ell_1 \times \ell_2$, $\ell_i \in \mathcal{L}_n$

  Example: $(x_0 \oplus x_1) \times (x_1 \oplus x_2) = x_0x_1 \oplus x_0x_2 \oplus x_1 \oplus x_1x_2$
  $\Rightarrow p = x_0x_1 \oplus x_0x_2 \oplus x_1x_2 \in \mathcal{Q}_{1,n}$.

- $\mathcal{Q}^L_{1,n} = \mathcal{Q}_{1,n} \oplus \mathcal{L}_n$
- $\mathcal{Q}_{2,n} = \mathcal{Q}_{1,n} \oplus \mathcal{Q}_{1,n}$

# Notations: Useful polynomial subsets

**Degree of a monomial:** number of variables that appear in the monomial.
**Degree of a polynomial:** maximum degree of its monomials.

- $\mathcal{L}_n$: set of **linear** polynomials (degree $\leq 1$)
- $\mathcal{Q}_n$: set of **quadratic** polynomials (degree $\leq 2$)
- $\mathcal{Q}_{1,n}$: non-linear truncations of $\ell_1 \times \ell_2$, $\ell_i \in \mathcal{L}_n$

  Example: $(x_0 \oplus x_1) \times (x_1 \oplus x_2) = x_0x_1 \oplus x_0x_2 \oplus x_1 \oplus x_1x_2$
  $\Rightarrow p = x_0x_1 \oplus x_0x_2 \oplus x_1x_2 \in \mathcal{Q}_{1,n}$.

- $\mathcal{Q}^L_{1,n} = \mathcal{Q}_{1,n} \oplus \mathcal{L}_n$
- $\mathcal{Q}_{2,n} = \mathcal{Q}_{1,n} \oplus \mathcal{Q}_{1,n}$

**Cost interpretation:** A polynomial in $\mathcal{Q}_{x,n}$ or $\mathcal{Q}^L_{x,n}$ requires exactly $x$ AND gates.

# Reformulation of the problem in the quadratic case

## General principle:

- Consider non-linear truncation of the polynomials in the ANF. (Y)
  If $y = x_0x_1 \oplus x_3 \oplus x_1x_2$, we only look at $x_0x_1 \oplus x_1x_2$.

## Formalisation in matrix form

$$Y = \begin{pmatrix} x_1x_2 \\ x_0x_1 \oplus x_0x_2 \\ x_0x_1 \end{pmatrix}$$

# Reformulation of the problem in the quadratic case

## General principle:

- Consider non-linear truncation of the polynomials in the ANF. (Y)
  If $y = x_0x_1 \oplus x_3 \oplus x_1x_2$, we only look at $x_0x_1 \oplus x_1x_2$.
- Store all possible quadratic factorisation schemes.
  $\rightarrow$ corresponds to the set $\mathcal{Q}_{1,n}$

## Formalisation in matrix form

$$\mathcal{Q}_{1,3} = \begin{pmatrix} x_0x_1 \\ x_0x_2 \\ x_1x_2 \\ x_0x_1 \oplus x_0x_2 \\ x_0x_1 \oplus x_1x_2 \\ x_0x_2 \oplus x_1x_2 \\ x_0x_1 \oplus x_0x_2 \oplus x_1x_2 \end{pmatrix} Y = \begin{pmatrix} x_1x_2 \\ x_0x_1 \oplus x_0x_2 \\ x_0x_1 \end{pmatrix}$$

# Reformulation of the problem in the quadratic case

## General principle:

- Consider non-linear truncation of the polynomials in the ANF. (Y)
  If $y = x_0x_1 \oplus x_3 \oplus x_1x_2$, we only look at $x_0x_1 \oplus x_1x_2$.
- Store all possible quadratic factorisation schemes.
  $\rightarrow$ corresponds to the set $\mathcal{Q}_{1,n}$

## Formalisation in matrix form

$$\mathcal{Q}_{1,3} = \begin{pmatrix} x_0x_1 \\ x_0x_2 \\ x_1x_2 \\ x_0x_1 \oplus x_0x_2 \\ x_0x_1 \oplus x_1x_2 \\ x_0x_2 \oplus x_1x_2 \\ x_0x_1 \oplus x_0x_2 \oplus x_1x_2 \end{pmatrix} Y = \begin{pmatrix} x_1x_2 \\ x_0x_1 \oplus x_0x_2 \\ x_0x_1 \end{pmatrix}$$

We search for a binary matrix $S$ such that:

$$S\mathcal{Q}_{1,3} = Y$$

# Reformulation of the problem in the quadratic case

To obtain the best implementation, we look for a solution $S$ having **as many zero columns as possible**.

The matrix

$$S_1 = \begin{array}{c} \begin{array}{ccccccc} q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

is a solution of the equation and induces the implementation :

$y_0 = q_1 \oplus q_5 = x_0x_2 \oplus x_0x_2 \oplus x_1x_2$

$y_1 = q_3 = x_0x_1 \oplus x_0x_2$

$y_2 = q_5 \oplus q_6 = x_0x_2 \oplus x_1x_2 \oplus x_0x_1 \oplus x_0x_2 \oplus x_1x_2$

$\rightarrow$ Each non-zero column implies that the operation is used in the implementation and then costs 1 AND.

# The algorithm chosen for the non-linear part:

**Objective**: Express each output bit as a XOR-sum of elements of $\mathcal{Q}_{1,n}$ by constructing iteratively a set **Op_selec** containing polynomials in $\mathcal{Q}_{1,n}$.
$\Rightarrow$ At the end, this set will contain all the polynomials to implement the S-box.

**Adding polynomials to Op_selec**:

**Principle function**

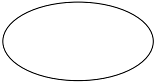
**Inputs**: An output bit $y$, Op_selec
**Outputs**: Op_selec $\bigcup$
$\{v_0, ..., v_b \in \mathcal{Q}_{1,n} \| \exists u_i \in Op\_selec, \bigoplus u_i \oplus \bigoplus v_i = y\}$, with $b$ minimal.

# Illustration for the non-linear part

Consider: $y_0 = x_0 x_2$

$y_1 = x_0 x_1 \oplus x_2 x_3$

$y_2 = x_0 x_3 \oplus x_1 x_2 \oplus x_2 x_3$



Op_selec

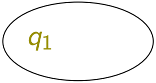
# Illustration for the non-linear part

Consider: $y_0 = x_0x_2$
$y_1 = x_0x_1 \oplus x_2x_3$
$y_2 = x_0x_3 \oplus x_1x_2 \oplus x_2x_3$

**First step:** $y_0$

- We cannot get $y_0$ adding 0 polynomials.
- $y_0 = x_0 \times x_2$ then $y_0 \in \mathcal{Q}_{1,4}$

$\rightarrow$ We add $q_1 = x_0x_2$ in Op_selec.




Op_selec

# Illustration for the non-linear part

Consider: $y_0 = x_0 x_2$
$y_1 = x_0 x_1 \oplus x_2 x_3$
$y_2 = x_0 x_3 \oplus x_1 x_2 \oplus x_2 x_3$

**First step:** $y_0$

- We cannot get $y_0$ adding 0 polynomials.
- $y_0 = x_0 \times x_2$ then $y_0 \in \mathcal{Q}_{1,4}$

$\rightarrow$ We add $q_1 = x_0 x_2$ in Op_selec.

$q_1$

Op_selec

**Second step:** $y_1$

- We cannot get $y_1$ adding 0 polynomials.
- We cannot get $y_1$ adding 1 polynomial.
- We can get $y_1$ by adding only 2 polynomials of $\mathcal{Q}_{1,4}$. We get several possibilities that all need to be tested:

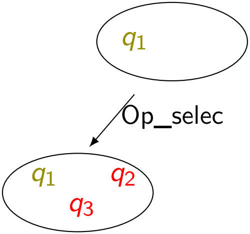
# Illustration for the non-linear part

Consider: $y_0 = x_0x_2$
$y_1 = x_0x_1 \oplus x_2x_3$
$y_2 = x_0x_3 \oplus x_1x_2 \oplus x_2x_3$

**Second step:** $y_1$

- $y_1 = (x_0 \times x_1) \oplus (x_2 \times x_3)$

$\rightarrow$ We add $q_2 = x_0x_1$ and $q_3 = x_2x_3$ in Op_selec.



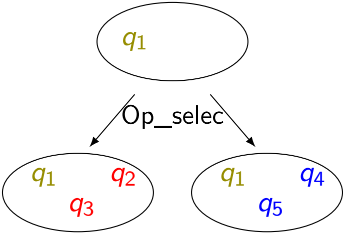
# Illustration for the non-linear part

Consider: $y_0 = x_0x_2$
$y_1 = x_0x_1 \oplus x_2x_3$
$y_2 = x_0x_3 \oplus x_1x_2 \oplus x_2x_3$

**Second step:** $y_1$

- $y_1 = (x_0 \times x_1) \oplus (x_2 \times x_3)$
$\rightarrow$ We add $q_2 = x_0x_1$ and $q_3 = x_2x_3$ in Op_selec.

- $y_1 = x_1 \times (x_0 \oplus x_2) \oplus x_2 \times (x_1 \oplus x_3)$
$\rightarrow$ We add $q_4 = x_0x_1 \oplus x_1x_2$ and $q_5$ $= x_1x_2 \oplus x_2x_3$ in Op_selec.



# Illustration for the non-linear part

Consider: $y_0 = x_0 x_2$
$y_1 = x_0 x_1 \oplus x_2 x_3$
$y_2 = x_0 x_3 \oplus x_1 x_2 \oplus x_2 x_3$

**Second step:** $y_1$

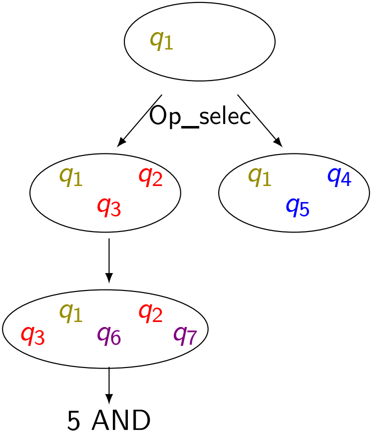
- $y_1 = (x_0 \times x_1) \oplus (x_2 \times x_3)$
$\rightarrow$ We add $q_2 = x_0 x_1$ and $q_3 = x_2 x_3$ in Op_selec.

- $y_1 = x_1 \times (x_0 \oplus x_2) \oplus x_2 \times (x_1 \oplus x_3)$
$\rightarrow$ We add $q_4 = x_0 x_1 \oplus x_1 x_2$ and $q_5 = x_1 x_2 \oplus x_2 x_3$ in Op_selec.

**Third step:** $y_2$

- We add $q_6 = x_0 x_3$ and $q_7 = x_1 x_2$.

Then $y_2 = q_3 \oplus q_6 \oplus q_7$



# Illustration for the non-linear part

Consider: $y_0 = x_0 x_2$
$y_1 = x_0 x_1 \oplus x_2 x_3$
$y_2 = x_0 x_3 \oplus x_1 x_2 \oplus x_2 x_3$

**Second step:** $y_1$

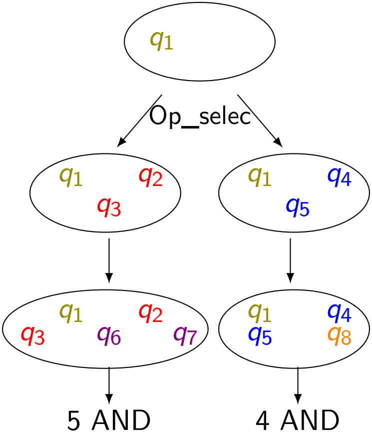
- $y_1 = (x_0 \times x_1) \oplus (x_2 \times x_3)$
$\rightarrow$ We add $q_2 = x_0 x_1$ and $q_3 = x_2 x_3$ in Op_selec.

- $y_1 = x_1 \times (x_0 \oplus x_2) \oplus x_2 \times (x_1 \oplus x_3)$
$\rightarrow$ We add $q_4 = x_0 x_1 \oplus x_1 x_2$ and $q_5 = x_1 x_2 \oplus x_2 x_3$ in Op_selec.

**Third step:** $y_2$

- We add $q_6 = x_0 x_3$ and $q_7 = x_1 x_2$.
Then $y_2 = q_3 \oplus q_6 \oplus q_7$

- We add $q_8 = x_0 x_3$. Then $y_2 = q_5 \oplus q_8$



# Order of treatment of the output bits

Suppose we use the order ($y_3$ - $y_0$ - $y_2$ - $y_1$) and obtain the resulting implementation with 7 non-linear operations $q_i$:

$$y_3 = q_0 \oplus q_4 \oplus q_5$$
$$y_0 = q_5 \oplus q_6$$
$$y_2 = q_2 \oplus q_3$$
$$y_1 = q_0 \oplus q_1$$

# Order of treatment of the output bits

Suppose we use the order ($y_3$ - $y_0$ - $y_2$ - $y_1$) and obtain the resulting implementation with 7 non-linear operations $q_i$:

$$
\begin{aligned}
y_3 &= q_0 \oplus q_4 \oplus q_5 \\
y_0 &= q_5 \oplus q_6 \\
y_2 &= q_2 \oplus q_3 \\
y_1 &= q_0 \oplus q_1
\end{aligned}
$$

If $y_0$ is also equal to $q_1 \oplus q_3 \oplus q_4$,

**what happens if we change the order ?**

# Order of treatment of the output bits

Suppose we use the order ($y_3$ - $y_0$ - $y_2$ - $y_1$) and obtain the resulting implementation with 7 non-linear operations $q_i$:

$$
\begin{aligned}
y_3 &= q_0 \oplus q_4 \oplus q_5 \\
y_0 &= q_5 \oplus q_6 \\
y_2 &= q_2 \oplus q_3 \\
y_1 &= q_0 \oplus q_1
\end{aligned}
$$

If $y_0$ is also equal to $q_1 \oplus q_3 \oplus q_4$,

**what happens if we change the order ?** If we use ($y_1$ - $y_2$ - $y_0$ - $y_3$), we obtain:

$$
\begin{aligned}
y_1 &= q_0 \oplus q_1 \\
y_2 &= q_2 \oplus q_3 \\
y_0 &= q_1 \oplus q_3 \oplus q_4 \\
y_3 &= q_0 \oplus q_4 \oplus q_5
\end{aligned}
$$

# Results for the quadratic case

| S-box | Implementation(**AND** - XOR) | | | | Timings (in seconds) | | |
|---|---|---|---|---|---|---|---|
| | [BDD+20] | [ZH23] | [FWZ+24] | Ours | [ZH23] | [FWZ+24] | Ours |
| $\chi_5$ | - | **5** - | **5** - 5 | **5** - 10 | 1.2 | 19.81 | 0.05 |
| ASCON | - | **5** - | **5** - | **5** - 15 | 1.4 | 26.48 | 0.1 |
| SYCON | - | - | **5** - 17 | **5** - 15 | | 26.72 | 0.1 |
| FIDES | **7** - 29 | **7** - | **7** - 27 | **7** - 20 | 3.6 | 140.31 | 0.2 |
| $X^3$ | **7** - 29 | - | - | **7** - 19 | | | $< 1$ |
| $X^5$ | **7** - 26 | - | - | **7** - 21 | | | $< 1$ |

Table: Results for 5-bit S-boxes.

# Results for the quadratic case

| | Implementation (**AND** - XOR) | |
|---|---|---|
| Size (on bits) | [BDD+20] | Ours |
| 5 | **7** - 29 | **7** - 19 |
| 6 | **9** - 43 | **8** - 37 |
| 7 | **15** - 79 | **11** - 45 |
| 8 | - | **14** - 67 |
| 9 | - | **19** - 94 |

Table: Results for the power function $X^3$

Our tool is able to handle any quadratic functions up to 9 bits.

# PART 3: DESCRIPTION OF THE ALGORITHM FOR HIGHER-DEGREE FUNCTIONS

# What about higher degrees ?

**An example for degree 3:**

$\mathcal{C}_{2,n}$ v1 $= \{\ell_1 \times \ell_2 \times \ell_3\}$, $\ell_i \in \mathcal{L}_n$

| $n$ | $\#\mathcal{Q}_{1,n}$ | $\#\mathcal{C}_{2,n}$ v1 | | |
|---|---|---|---|---|
| 4 | 35 | 141 | | |
| 5 | 155 | 1241 | | |
| 6 | 651 | 10 417 | | |

Table: Cardinalities of polynomial sets for degree 3

# What about higher degrees ?

**An example for degree 3:**

$\mathcal{C}_{2,n}$ v1 $= \{\ell_1 \times \ell_2 \times \ell_3\}$, $\ell_i \in \mathcal{L}_n$
$\mathcal{C}_{2,n}$ v2 $= \{((\ell_1 \times \ell_2) \oplus \ell_3) \times \ell_4\}$, $\ell_i \in \mathcal{L}_n$

| $n$ | $\#\mathcal{Q}_{1,n}$ | $\#\mathcal{C}_{2,n}$ v1 | $\#\mathcal{C}_{2,n}$ v2 |
|---|---|---|---|
| 4 | 35 | 141 | 561 |
| 5 | 155 | 1241 | 14 261 |
| 6 | 651 | 10 417 | 283 387 |

Table: Cardinalities of polynomial sets for degree 3

# What about higher degrees ?

**An example for degree 3:**

$\mathcal{C}_{2,n}$ v1 $= \{\ell_1 \times \ell_2 \times \ell_3\}$, $\ell_i \in \mathcal{L}_n$
$\mathcal{C}_{2,n}$ v2 $= \{((\ell_1 \times \ell_2) \oplus \ell_3) \times \ell_4\}$, $\ell_i \in \mathcal{L}_n$
$\mathcal{C}_{3,n} = \{((\ell_1 \times \ell_2) \oplus (\ell_3 \times \ell_4) \oplus \ell_5) \times \ell_6\}$, $\ell_i \in \mathcal{L}_n$

| $n$ | $\#\mathcal{Q}_{1,n}$ | $\#\mathcal{C}_{2,n}$ v1 | $\#\mathcal{C}_{2,n}$ v2 | $\#\mathcal{C}_{3,n}$ |
|---|---|---|---|---|
| 4 | 35 | 141 | 561 | 561 |
| 5 | 155 | 1241 | 14 261 | 28 149 |
| 6 | 651 | 10 417 | 283 387 | 2 033 725 |

Table: Cardinalities of polynomial sets for degree 3

# What about higher degrees ?

**An example for degree 3:**

$\mathcal{C}_{2,n}$ v1 $= \{\ell_1 \times \ell_2 \times \ell_3\}$, $\ell_i \in \mathcal{L}_n$
$\mathcal{C}_{2,n}$ v2 $= \{((\ell_1 \times \ell_2) \oplus \ell_3) \times \ell_4\}$, $\ell_i \in \mathcal{L}_n$
$\mathcal{C}_{3,n} = \{((\ell_1 \times \ell_2) \oplus (\ell_3 \times \ell_4) \oplus \ell_5) \times \ell_6\}$, $\ell_i \in \mathcal{L}_n$

| $n$ | $\#\mathcal{Q}_{1,n}$ | $\#\mathcal{C}_{2,n}$ v1 | $\#\mathcal{C}_{2,n}$ v2 | $\#\mathcal{C}_{3,n}$ |
|---|---|---|---|---|
| 4 | 35 | 141 | 561 | 561 |
| 5 | 155 | 1241 | 14 261 | 28 149 |
| 6 | 651 | 10 417 | 283 387 | 2 033 725 |

Table: Cardinalities of polynomial sets for degree 3

$\rightarrow$ A more relevant approach would be to use the different formulas given above to decompose a given function.

# From the quadratic case to higher-degree functions

$$\mathcal{C}_{2,n}\ \mathsf{v1} = \{\ell_1 \times \ell_2 \times \ell_3\}, \quad \ell_i \in \mathcal{L}_n$$

$$\mathcal{C}_{2,n}\ \mathsf{v2} = \{((\ell_1 \times \ell_2) \oplus \ell_3) \times \ell_4\}, \quad \ell_i \in \mathcal{L}_n$$

$$\mathcal{C}_{3,n} = \{((\ell_1 \times \ell_2) \oplus (\ell_3 \times \ell_4) \oplus \ell_5) \times \ell_6\}, \quad \ell_i \in \mathcal{L}_n$$

# From the quadratic case to higher-degree functions

$$y = q_1 \times \ell_3, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$

$$\mathcal{C}_{2,n} \text{ v2} = \{((\ell_1 \times \ell_2) \oplus \ell_3) \times \ell_4\}, \quad \ell_i \in \mathcal{L}_n$$

$$\mathcal{C}_{3,n} = \{((\ell_1 \times \ell_2) \oplus (\ell_3 \times \ell_4) \oplus \ell_5) \times \ell_6\}, \quad \ell_i \in \mathcal{L}_n$$

# From the quadratic case to higher-degree functions

$$y = q_1 \times \ell_3, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$

$$y = (q_1 \oplus \ell_3) \times \ell_4, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$

$$y = q_2 \times \ell_4, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}^L_{1,n}$$

$$\mathcal{C}_{3,n} = \{((\ell_1 \times \ell_2) \oplus (\ell_3 \times \ell_4) \oplus \ell_5) \times \ell_6\}, \quad \ell_i \in \mathcal{L}_n$$

# From the quadratic case to higher-degree functions

$$y = q_1 \times \ell_3, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$

$$y = (q_1 \oplus \ell_3) \times \ell_4, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$
$$y = q_2 \times \ell_4, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}^L_{1,n}$$

$$y = (q_1 \oplus q_2 \oplus \ell_5) \times \ell_6, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$
$$y = q_3 \times \ell_6, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}^L_{2,n}$$

# From the quadratic case to higher-degree functions

$$y = q_1 \times \ell_3, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$

$$y = (q_1 \oplus \ell_3) \times \ell_4, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$

$$y = q_2 \times \ell_4, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}^L_{1,n}$$

$$y = (q_1 \oplus q_2 \oplus \ell_5) \times \ell_6, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$

$$y = q_3 \times \ell_6, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}^L_{2,n}$$

Idea: Test if an output bit admits such a representation.

# From the quadratic case to higher-degree functions

$$y = q_1 \times \ell_3, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$

$$y = (q_1 \oplus \ell_3) \times \ell_4, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$
$$y = q_2 \times \ell_4, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}^L_{1,n}$$

$$y = (q_1 \oplus q_2 \oplus \ell_5) \times \ell_6, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$
$$y = q_3 \times \ell_6, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}^L_{2,n}$$

Idea: Test if an output bit admits such a representation.

$\rightarrow$ **Divide** by $\ell_i$ and check whether the quotient and remainder fit the target form.

# From the quadratic case to higher-degree functions

$$y = q_1 \times \ell_3, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$

$$y = (q_1 \oplus \ell_3) \times \ell_4, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$
$$y = q_2 \times \ell_4, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}^L_{1,n}$$

$$y = (q_1 \oplus q_2 \oplus \ell_5) \times \ell_6, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}_{1,n}$$
$$y = q_3 \times \ell_6, \qquad \ell_i \in \mathcal{L}_n,\ q_i \in \mathcal{Q}^L_{2,n}$$

Idea: Test if an output bit admits such a representation.

→ **Divide** by $\ell_i$ and check whether the quotient and remainder fit the target form.

**Previous approach:**
Start from the inputs and rely on *multiplications tables*.

**New approach:** Start from the outputs and use *division tables*.

# High-level description of the algorithm

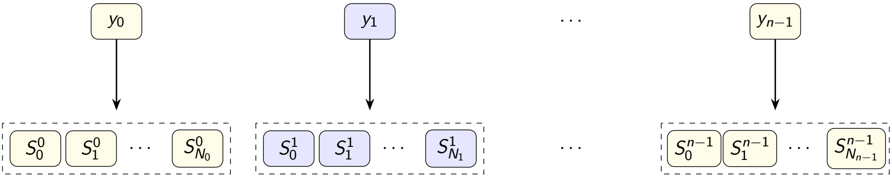
**2 different phases**

# High-level description of the algorithm

**2 different phases**

Phase 1: AND depth $\geq 2 \rightarrow 1$ Reduce high-degree functions into compositions of quadratic functions based on recognising factorisation patterns (*using formulas*).

Objective: Find efficient implementations (*decomposition sets*) expressed in $\{\mathcal{L}_n \bigcup \mathcal{Q}_{1,n}\}$ of each output bit $y_i$.

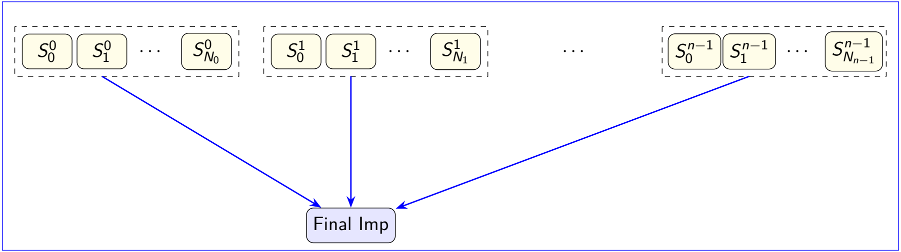



$\boxed{S_j^i}$ : j-th decomposition set for output $y_i$,
set of polynomials in $\mathcal{L}_n$ and $\mathcal{Q}_{1,n}$ enabling the implementation of output bit $y_i$

# High-level description of the algorithm

Phase 2: AND depth $1 \to 0$ Computational phase based on algebraic calculations, different from the previous quadratic algorithm.

Objective: Find a combination of these parallel implementations to minimise cost.



# First phase: Construction of the decomposition sets

**Decomposition**: A decomposition of a polynomial is a formal expression representing the polynomial using multiplications, additions and polynomials of degree at most 2.

**Cost of a decomposition** = Number of multiplications in the expression.

**Example:**

$y = (q_1 \times l_1) \oplus l_2$ is a decomposition of a polynomial of degree 3 and its cost is 1.

# First phase: Construction of the decomposition sets

**Decomposition**: A decomposition of a polynomial is a formal expression representing the polynomial using multiplications, additions and polynomials of degree at most 2.

**Cost of a decomposition** = Number of multiplications in the expression.

**Example:**

$y = (q_1 \times l_1) \oplus l_2$ is a decomposition of a polynomial of degree 3 and its cost is 1.

**Total cost of an implementation:**

$$\underbrace{\text{Cost of the decompositions}}_{\#\text{ AND in formal expression}} + \underbrace{\text{Cost of the quadratic polynomials used}}_{\text{from Phase 2}}$$

# First phase: Construction of the decomposition sets

**For each degree:** We define a specific list of possible decompositions, ordered by increasing multiplicative depth and increasing cost.

# First phase: Construction of the decomposition sets

**For each degree:** We define a specific list of possible decompositions, ordered by increasing multiplicative depth and increasing cost.

**Search strategy:**

- We explore decomposition starting from the lowest cost.
- If valid solutions are found for a given cost, we stop exploring the next, more expensive ones.
- Divisions are tested successively with the divisor $d$ belonging to sets of increasing cost:

$$d \in \mathcal{L}_n \ \rightarrow \ d \in \mathcal{Q}^L_{1,n} \ \rightarrow \ d \in \mathcal{Q}^L_{2,n}$$

# First phase: Construction of the decomposition sets

**For each degree:** We define a specific list of possible decompositions, ordered by increasing multiplicative depth and increasing cost.

**Search strategy:**

- We explore decomposition starting from the lowest cost.
- If valid solutions are found for a given cost, we stop exploring the next, more expensive ones.
- Divisions are tested successively with the divisor $d$ belonging to sets of increasing cost:

$$d \in \mathcal{L}_n \;\rightarrow\; d \in \mathcal{Q}^{L}_{1,n} \;\rightarrow\; d \in \mathcal{Q}^{L}_{2,n}$$

These strategies ensure that only **locally minimal** decompositions are kept.

# First phase: Instantiation of the decomposition sets using divisions between multivariate polynomials

**Objective:** Given $p_1, p_2 \in \mathbb{F}_2[x_1, ..., x_n]$ with $deg(p_2) < deg(p_1)$, division means finding a couple $q$, $r \in \mathbb{F}_2[x_1, ..., x_n]$ such that:

$$p_1 = p_2 \times q \oplus r, \text{ with } r < p_2.$$

# First phase: Instantiation of the decomposition sets using divisions between multivariate polynomials

**Objective:** Given $p_1, p_2 \in \mathbb{F}_2[x_1, ..., x_n]$ with $deg(p_2) < deg(p_1)$, division means finding a couple $q$, $r \in \mathbb{F}_2[x_1, ..., x_n]$ such that:

$$p_1 = p_2 \times q \oplus r, \text{ with } r < p_2.$$

To define this division, we need to:

- define how to divide two monomials,
- set a monomial order to process the monomials.

$\rightarrow$ we will successively divide the monomials of the divisor by the monomials of the dividend.

The difficult part here is to define a fitting operator "$<$" over $\mathbb{F}_2[x_1, ..., x_n]$.

# First phase: Monomial Orders

**Lexicographic order:** Example for $x_0 > x_1 > x_2 > x_3 > 1$:

$$x_0x_1x_2x_3 > x_0x_1x_2 > x_0x_1x_3 > x_0x_2x_3 > x_1x_2x_3 > x_0x_1 > \cdots > x_3 > 1$$

# First phase: Monomial Orders

**Lexicographic order:** Example for $x_0 > x_1 > x_2 > x_3 > 1$:

$$x_0x_1x_2x_3 > x_0x_1x_2 > x_0x_1x_3 > x_0x_2x_3 > x_1x_2x_3 > x_0x_1 > \cdots > x_3 > 1$$

Consider $p = x_0x_1x_2 \oplus x_0x_2 \oplus x_0x_3 \oplus x_1x_2x_3$ and $d = x_0 \oplus x_1x_2$.

$$\begin{array}{rr|l}
 & x_0x_1x_2 \oplus x_1x_2x_3 \oplus x_0x_2 \oplus x_0x_3 & x_1x_2 \oplus x_0 \\ \hline
\oplus & \underline{x_0x_1x_2 \oplus x_0} & x_0 \oplus x_3 \\
 & x_1x_2x_3 \oplus x_0x_2 \oplus x_0x_3 \oplus x_0 & \\
\oplus & \underline{x_1x_2x_3 \oplus x_0x_3} & \\
 & x_0x_2 \oplus x_0 &
\end{array}$$

# First phase: Monomial Orders

**Lexicographic order:** Example for $x_0 > x_1 > x_2 > x_3 > 1$:

$$x_0x_1x_2x_3 > x_0x_1x_2 > x_0x_1x_3 > x_0x_2x_3 > x_1x_2x_3 > x_0x_1 > \cdots > x_3 > 1$$

Consider $p = x_0x_1x_2 \oplus x_0x_2 \oplus x_0x_3 \oplus x_1x_2x_3$ and $d = x_0 \oplus x_1x_2$.

$$\begin{array}{r|l}
x_0x_1x_2 \oplus x_1x_2x_3 \oplus x_0x_2 \oplus x_0x_3 & x_1x_2 \oplus x_0 \\
\hline
\oplus \quad \underline{x_0x_1x_2 \oplus x_0} & x_0 \oplus x_3 \\
x_1x_2x_3 \oplus x_0x_2 \oplus x_0x_3 \oplus x_0 & \\
\oplus \quad \underline{x_1x_2x_3 \oplus x_0x_3} & \\
x_0x_2 \oplus x_0 &
\end{array}$$

**Motivation for adaptation:**

- In classical lexicographic order, monomials are sorted mainly by degree.
- For effective division, we prefer monomials containing a specific variable (e.g. $x_0$) to appear first.

# First phase: Monomial Orders

**Adapted order:** We modify the variable hierarchy so that one variable leads the order: $x_i > 1 >$ remaining variables (in index order).

# First phase: Monomial Orders

**Adapted order:** We modify the variable hierarchy so that one variable leads the order: $x_i > 1 >$ remaining variables (in index order).
Example for $x_0 > 1 > x_1 > x_2 > x_3$:

$$x_0x_1x_2x_3 > x_0x_1x_2 > x_0x_1x_3 > x_0x_2x_3 > x_0x_1 > x_0x_2 > x_0x_3 > x_0 > \cdots$$

$\Rightarrow$ We call this the **monomial order induced by** $x_i$.

# First phase: Monomial Orders

**Adapted order:** We modify the variable hierarchy so that one variable leads the order: $x_i > 1 >$ remaining variables (in index order).
Example for $x_0 > 1 > x_1 > x_2 > x_3$:

$$x_0x_1x_2x_3 > x_0x_1x_2 > x_0x_1x_3 > x_0x_2x_3 > x_0x_1 > x_0x_2 > x_0x_3 > x_0 > \cdots$$

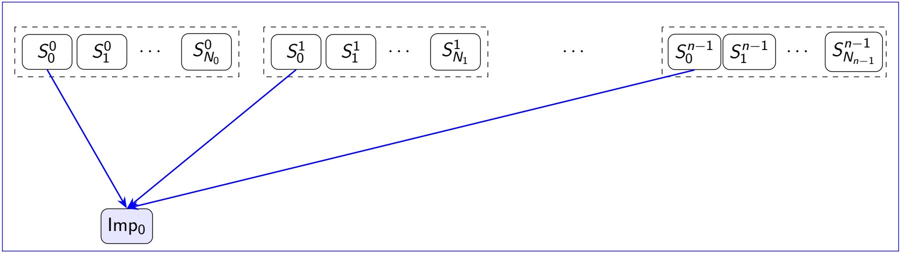
$\Rightarrow$ We call this the **monomial order induced by** $x_i$.

consider now the order $x_0 > 1 > x_1 > x_2 > x_3$

$$\begin{array}{rr|l}
 & x_0x_1x_2 \oplus x_0x_2 \oplus x_0x_3 \oplus x_1x_2x_3 & x_0 \oplus x_1x_2 \\
\hline\hline
\oplus & x_0x_1x_2 \oplus x_1x_2 & x_1x_2 \oplus x_2 \oplus x_3 \\
\oplus & x_0x_2 \oplus x_1x_2 & \\
\oplus & \underline{x_0x_3 \oplus x_1x_2x_3} & \\
 & 0 &
\end{array}$$

# Second phase: Aggregation of the decomposition sets

**Objective: Find a combination of decomposition sets to obtain a good final inplementation.**

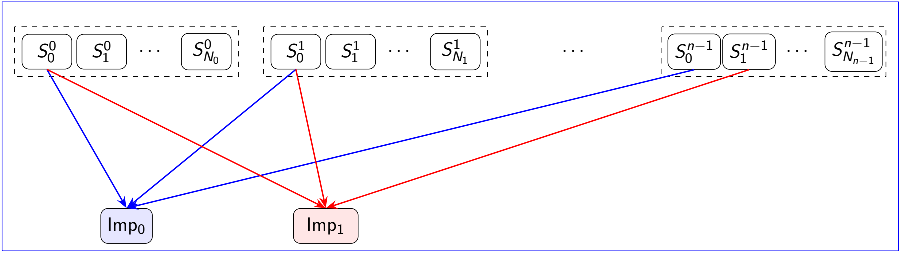



$\boxed{S_j^i}$ : j-th decomposition set for output $y_i$,
set of polynomials in $\mathcal{L}_n$ and $\mathcal{Q}_{1,n}$ enabling the implementation of output bit $y_i$

$\boxed{\mathrm{imp}_k}$ : k-th set of polynomials in $\mathcal{Q}_{1,n}$ enabling the implementation of the whole S-box

# Second phase: Aggregation of the decomposition sets

**Objective: Find a combination of decomposition sets to obtain a good final inplementation.**

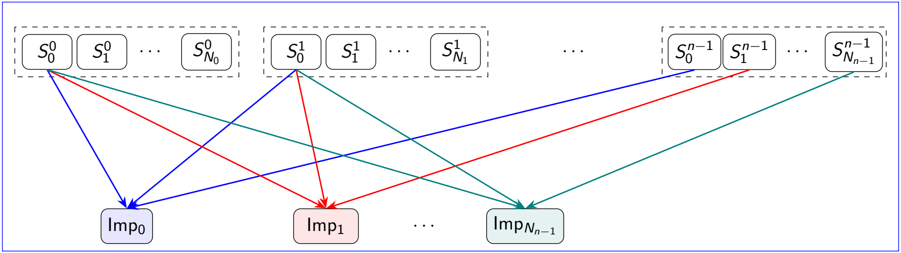



$\boxed{S_j^i}$ : j-th decomposition set for output $y_i$,
set of polynomials in $\mathcal{L}_n$ and $\mathcal{Q}_{1,n}$ enabling the implementation of output bit $y_i$

$\boxed{\mathrm{imp}_k}$ : k-th set of polynomials in $\mathcal{Q}_{1,n}$ enabling the implementation of the whole S-box

# Second phase: Aggregation of the decomposition sets

**Objective: Find a combination of decomposition sets to obtain a good final inplementation.**

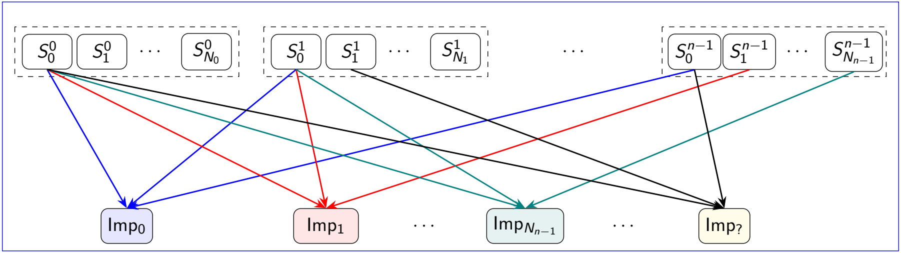



$\boxed{S_j^i}$ : j-th decomposition set for output $y_i$,
set of polynomials in $\mathcal{L}_n$ and $\mathcal{Q}_{1,n}$ enabling the implementation of output bit $y_i$

$\boxed{\mathrm{imp}_k}$ : k-th set of polynomials in $\mathcal{Q}_{1,n}$ enabling the implementation of the whole S-box

# Second phase: Aggregation of the decomposition sets

**Objective: Find a combination of decomposition sets to obtain a good final inplementation.**




$\boxed{S_j^i}$ : j-th decomposition set for output $y_i$,
set of polynomials in $\mathcal{L}_n$ and $\mathcal{Q}_{1,n}$ enabling the implementation of output bit $y_i$

$\boxed{\mathrm{imp}_k}$ : k-th set of polynomials in $\mathcal{Q}_{1,n}$ enabling the implementation of the whole S-box

$\rightarrow$ Find a criterion to assess whether a given family of polynomials can be efficiently implemented together or not.

# Second phase: Rank computation

$S^0 = \{x_0x_1 \oplus x_0x_3, x_2x_3\} \quad S^1 = \{x_0x_2 \oplus x_2x_3, x_0x_3\} \quad S^2 = \{x_0x_1, x_2x_3\}$

# Second phase: Rank computation

$S^0 = \{x_0x_1 \oplus x_0x_3, x_2x_3\}$ $S^1 = \{x_0x_2 \oplus x_2x_3, x_0x_3\}$ $S^2 = \{x_0x_1, x_2x_3\}$

$$\mathrm{Imp} = \bigcup_{i=0}^{2} S^i = \{x_0x_1 \oplus x_0x_3,\ x_2x_3,\ x_0x_2 \oplus x_2x_3,\ x_0x_3,\ x_0x_1,\ x_2x_3\}$$

# Second phase: Rank computation

$S^0 = \{x_0x_1 \oplus x_0x_3, x_2x_3\} \quad S^1 = \{x_0x_2 \oplus x_2x_3, x_0x_3\} \quad S^2 = \{x_0x_1, x_2x_3\}$

$$\mathrm{Imp} = \bigcup_{i=0}^{2} S^i = \{x_0x_1 \oplus x_0x_3,\ x_2x_3,\ x_0x_2 \oplus x_2x_3,\ x_0x_3,\ x_0x_1,\ x_2x_3\}$$

$$M_{imp} = \begin{array}{c} \begin{array}{cccccc} x_0x_1 & x_0x_2 & x_0x_3 & x_1x_2 & x_1x_3 & x_2x_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

# Second phase: Rank computation

$S^0 = \{x_0x_1 \oplus x_0x_3, x_2x_3\}$ $\quad S^1 = \{x_0x_2 \oplus x_2x_3, x_0x_3\}$ $\quad S^2 = \{x_0x_1, x_2x_3\}$

$$\text{Imp} = \bigcup_{i=0}^{2} S^i = \{x_0x_1 \oplus x_0x_3,\ x_2x_3,\ x_0x_2 \oplus x_2x_3,\ x_0x_3,\ x_0x_1,\ x_2x_3\}$$

$$M_{imp} = \begin{array}{c} \begin{array}{cccccc} x_0x_1 & x_0x_2 & x_0x_3 & x_1x_2 & x_1x_3 & x_2x_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

**Rank of** $M$ = Required number of AND gates to implement all the quadratic polynomials in the set Imp.

# Summary of the algorithm

**Total cost of an implementation:**

$$\underbrace{\text{Cost of the decomposition}}_{\#\ \text{AND in formal expression}} + \underbrace{\text{Cost of the quadratic polynomials used}}_{\text{from rank computation}}$$

# Summary of the algorithm

**Total cost of an implementation:**

$$\underbrace{\text{Cost of the decomposition}}_{\#\text{ AND in formal expression}} + \underbrace{\text{Cost of the quadratic polynomials used}}_{\text{from rank computation}}$$

**Current capabilities:**

- Up to **degree 5** on **6-bit** functions.
- Up to **degree 3** on **7-bit** functions.

# Summary of the algorithm

**Total cost of an implementation:**

$$\underbrace{\text{Cost of the decomposition}}_{\#\text{ AND in formal expression}} + \underbrace{\text{Cost of the quadratic polynomials used}}_{\text{from rank computation}}$$

**Current capabilities:**

- Up to **degree 5** on **6-bit** functions.
- Up to **degree 3** on **7-bit** functions.

**Remarks:**

- The algorithm performs well for decompositions from **depth 2** → **depth 1** (e.g., degree 4 or 3 to degree 2).
- Increasing the depth makes decompositions more complex and requires more divisions.
- We achieved **depth 3** with degree 5 on 6-bit functions using specific precomputations for the **depth 3** → **depth 2** step.

# Precomputation Strategy for Degree-5 Polynomials

Phase 1 is time-consuming but uses little memory.
**Idea:** Use precomputation to optimise the time-memory trade-off

# Precomputation Strategy for Degree-5 Polynomials

Phase 1 is time-consuming but uses little memory.
**Idea:** Use precomputation to optimise the time-memory trade-off

- Use truncated division tables: $y = q \times c_q \oplus c_r$, with degree constraints $\deg(c_q), \deg(c_r) \leq 3$.
- Only higher-degree terms (degrees 4-5) matter to perform the division lower-degree terms can be absorbed into remainder.
- This reduces the number of polynomials to $2^{\binom{6}{5}+\binom{6}{4}} = 2^{6+15} = 2^{21}$, making precomputation feasible.

# Precomputation Strategy for Degree-5 Polynomials

- Decompose output $y$ into high- and low-degree parts:

$$y = y_{\geq 4} \oplus y_{\leq 3}$$

- Using the precomputed table:

$$y_{\geq 4} = (c_1 \times q_1) \oplus c_2 \quad \Rightarrow \quad y = (c_1 \times q_1) \oplus c_3,\ c_3 = y_{\leq 3} \oplus c_2$$

$\rightarrow$ we can use the previous technique to deal with $c_1$ and $c_3$.

# Precomputation Strategy for Degree-5 Polynomials

- Decompose output $y$ into high- and low-degree parts:

$$y = y_{\geq 4} \oplus y_{\leq 3}$$

- Using the precomputed table:

$$y_{\geq 4} = (c_1 \times q_1) \oplus c_2 \quad \Rightarrow \quad y = (c_1 \times q_1) \oplus c_3,\ c_3 = y_{\leq 3} \oplus c_2$$

$\rightarrow$ we can use the previous technique to deal with $c_1$ and $c_3$.

- Precomputation storage $\approx 51GB$

# Precomputation Strategy for Degree-5 Polynomials

- Decompose output $y$ into high- and low-degree parts:

$$y = y_{\geq 4} \oplus y_{\leq 3}$$

- Using the precomputed table:

$$y_{\geq 4} = (c_1 \times q_1) \oplus c_2 \quad \Rightarrow \quad y = (c_1 \times q_1) \oplus c_3,\ c_3 = y_{\leq 3} \oplus c_2$$

$\rightarrow$ we can use the previous technique to deal with $c_1$ and $c_3$.

- Precomputation storage $\approx 51GB$

**What about degree-5 polynomials over 7 bits ?**

The number of polynomials containing only monomials of degree 4 and 5 would be $2^{\binom{7}{5}+\binom{7}{4}} = 2^{21+35} = 2^{56}$, making this approach unfeasible for higher sizes.

# Results for 4-bit functions

| S-box | Security properties | | | | Implementation properties | |
|---|---|---|---|---|---|---|
| | deg | $\delta$ | $\mathcal{L}$ | Bij | Our **D**/**AND**/XOR | Previous **D**/**AND**/XOR |
| Present [BKL+07] | 3 | 4 | 8 | yes | **2**/**4**/18 | **2**/**4**/14 [FWZ+24] |
| Rectangle [ZBL+15] | 3 | 4 | 8 | yes | **2**/**4**/18 | **2**/**4**/11 [FWZ+24] |
| Gift [BPP+17] | 3 | 6 | 8 | yes | **2**/**5**/18 | **2**/**5**/16 [FWZ+24] |
| Prince [BEK+20] | 3 | 4 | 8 | yes | **2**/**6**/23 | **2**/**6**/24 [BDD+20] |
| Prost [KLL+14] | 3 | 4 | 8 | yes | **2**/**4**/8 | **2**/**4**/15 [FWZ+24] |
| Piccolo [SIH+11] | 3 | 4 | 8 | yes | **2**/**4**/17 | **2**/**4**/4 [FWZ+24] |
| Elephant [DM19] | 3 | 4 | 8 | yes | **2**/**4**/17 | **2**/**4**/14 [FWZ+24] |
| sc2000-4 [SYY+02] | 3 | 4 | 8 | yes | **2**/**6**/19 | 4/5/23 [FWZ+24] |
| Twine [SMMK11] | 3 | 4 | 8 | yes | **2**/**6**/22 | **2**/**6**/21 [FWZ+24] |
| Minalpher [STA+14] | 3 | 4 | 8 | yes | **2**/**6**/25 | **2**/**6**/21 [FWZ+24] |
| $X^{-1}$ in $GF(2^4)$ | 3 | 4 | 8 | yes | **2**/**6**/12 | 3/5/11 [BP10]<br>2/7/10 [BP12] |

# Results for 5-bit functions

| S-box | Security properties | | | | Implementation properties | |
|---|---|---|---|---|---|---|
| | deg | $\delta$ | $\mathcal{L}$ | Bij | Our **D**/**AND**/XOR | Previous **D**/**AND**/XOR |
| $(X^3)^{-1}$ | 3 | 2 | 8 | yes | **2**/**10**/37 | 2/12/64 [BDD+20] |
| $(X^5)^{-1}$ | 3 | 2 | 8 | yes | **2**/**10**/37 | **2**/**10**/65 [BDD+20] |
| Fides Inv [BBK+13] | 3 | 2 | 8 | yes | **2**/**10**/40 | **2**/**10**/62 [BDD+20] |
| Keccak Inv [BDPA09] | 3 | 8 | 16 | yes | **2**/**9**/27 | **2**/**9**/52 [BDD+20] |
| sc2000-5 [SYY+02] | 3 | 2 | 8 | yes | **2**/**10**/44 | 7/10/– [FWZ+24] |
| $X^{-1}$ | 4 | 2 | 12 | yes | **2**/**12**/51 | 2/25/79 (ANF) |
| DB_1 [DB19] | 4 | 4 | 12 | yes | **2**/**9**/36 | 2/23/56 (ANF) |
| DB_17 [DB19] | 4 | 2 | 12 | yes | **2**/**12**/50 | 2/25/66 (ANF) |

# Results for 6-bit functions

| S-box | Security properties | | | | Implementation properties | |
|---|---|---|---|---|---|---|
| | deg | $\delta$ | $\mathcal{L}$ | Bij | Our **D**/**AND**/XOR | Previous **D**/**AND**/XOR |
| Q_2256 Inv [DB19] | 3 | 4 | 16 | yes | **2/13**/55 | 2/31/64 [BDD+20] |
| Q_2257 Inv [DB19] | 3 | 4 | 16 | yes | **2/13**/46 | 2/29/94 (ANF) |
| Q_2258 Inv [DB19] | 3 | 4 | 16 | yes | **2/11**/42 | 2/22/58 [BDD+20] |
| Q_2259 Inv [DB19] | 3 | 4 | 16 | yes | **2/13**/51 | 2/34/115 (ANF) |
| Q_2260 Inv [DB19] | 3 | 4 | 16 | yes | **2/12**/53 | 2/33/111 (ANF) |
| Q_2261 Inv [DB19] | 3 | 4 | 16 | yes | **2/11**/47 | 2/29/87 (ANF) |
| Q_2262 Inv [DB19] | 3 | 4 | 16 | yes | **2/13**/57 | 2/32/102 (ANF) |
| Q_2263 Inv [DB19] | 3 | 4 | 16 | yes | **2/16**/71 | 2/26/– [BDD+20] |
| Dillon [BDMW10] | 4 | 2 | 16 | yes | **2/17**/75 | 3/12/– [PUB16] |
| $X^{15}$ | 4 | 8 | 16 | no | **2/18**/86 | 2/50/177 (ANF) |
| $X^{31}$ (Affine eq. of $X^{-1}$) | 5 | 4 | 16 | yes | 3/29/109 | **3/21**/63 $X^{-1}$ TF |
| sc2000-6 [SYY+02] | 5 | 4 | 16 | yes | **3/28**/122 | 7/19/– [FWZ+24] |
| Speedy [LMMR21] | 5 | 8 | 28 | yes | **3/20**/51 | 5/12/– [FWZ+24] |

# Perspectives and conclusion

**Our work:** S-box implementations via pattern recognition and polynomial factorisation

**Achievements**

- Depth 1 (degree 2 up to 9 bits )
- Depth 2 (degree 3 and 4 up to 7 bits)
- Depth 3 (degree 5 up to 6 bits)

$\rightarrow$ *New available S-boxes for designers.*

**Next steps**

- Optimise algorithm and implementation
- Extend to higher depths
- Apply beyond binary case

# Perspectives and conclusion

**Our work:** S-box implementations via pattern recognition and polynomial factorisation

**Achievements**

- Depth 1 (degree 2 up to 9 bits )
- Depth 2 (degree 3 and 4 up to 7 bits)
- Depth 3 (degree 5 up to 6 bits)

**Next steps**

- Optimise algorithm and implementation
- Extend to higher depths
- Apply beyond binary case

→ *New available S-boxes for designers.*

**Thank you for your attention !**

# Monomial Division over $\mathbb{F}_2[x_1, ..., x_n]$

**Representation:** A multivariate monomial in $n$ variables is encoded by an integer on $n$ bits.

Example: $6_{10} = 0110_2 \Rightarrow m = x_2 x_1$

# Monomial Division over $\mathbb{F}_2[x_1, ..., x_n]$

**Representation:** A multivariate monomial in $n$ variables is encoded by an integer on $n$ bits.

Example: $6_{10} = 0110_2 \Rightarrow m = x_2 x_1$

**Division rule:**

- Let $m_1, m_2$ be represented by integers $a_1, a_2$ with $deg(m_1) > deg(m_2)$.
- $m_1$ is **divisible** by $m_2$ if $a_1 \mid a_2 = a_1$ (bitwise OR).
- The result of the division is the monomial represented by $a = a_1 \oplus a_2$.

# Monomial Division over $\mathbb{F}_2[x_1, ..., x_n]$

**Representation:** A multivariate monomial in $n$ variables is encoded by an integer on $n$ bits.

Example: $6_{10} = 0110_2 \Rightarrow m = x_2x_1$

**Division rule:**

- Let $m_1, m_2$ be represented by integers $a_1, a_2$ with $deg(m_1) > deg(m_2)$.
- $m_1$ is **divisible** by $m_2$ if $a_1 \mid a_2 = a_1$ (bitwise OR).
- The result of the division is the monomial represented by $a = a_1 \oplus a_2$.

Example:

| | |
|---|---|
| $m_1 = x_1x_2x_3$ | $a_1 = 1110$ |
| $m_2 = x_1x_3$ | $a_2 = 1010$ |

- $a_1|a_2 = 1110 = a_1 \Rightarrow m_1$ divisible by $m_2$
- $a_1 \oplus a_2 = 0100 \Rightarrow x_1x_2x_3 = x_1x_3 \times x_2$

# Main limitations:

- **Multiplicative depth:**
  - The first phase explores a tree of decompositions whose size grows exponentially with the depth.
  - A larger depth also produces more decompositions, increasing the search space and the number of rank computations in the second phase.
- **S-box size:**
  - For $n \leq 6$, degree-3 or degree-4 polynomials often have cost-1 decompositions.
  - When $n \geq 7$, such low-cost decompositions become rare, forcing the use of higher-cost ones.
  - Each additional cost level implies testing larger divisor sets ($\mathcal{L}_n$, then $\mathcal{Q}^L_{1,n}$, then $\mathcal{Q}^L_{2,n}$), whose size grows rapidly with $n$.

# Main limitations:

- **Multiplicative depth:**
  - The first phase explores a tree of decompositions whose size grows exponentially with the depth.
  - A larger depth also produces more decompositions, increasing the search space and the number of rank computations in the second phase.

- **S-box size:**
  - For $n \leq 6$, degree-3 or degree-4 polynomials often have cost-1 decompositions.
  - When $n \geq 7$, such low-cost decompositions become rare, forcing the use of higher-cost ones.
  - Each additional cost level implies testing larger divisor sets ($\mathcal{L}_n$, then $\mathcal{Q}^L_{1,n}$, then $\mathcal{Q}^L_{2,n}$), whose size grows rapidly with $n$.

Observation: At $n = 6$, about half of the quadratic polynomials can serve as divisors; at $n = 7$, this ratio drops below 0.15, and continues decreasing, making higher-degree divisions impractical in practice.