

# Optimized homomorphic evaluation of Boolean functions

Nicolas Bon

Joint work with David Pointcheval and Matthieu Rivain

# Outline

- 1 . Introduction to Fully Homomorphic Encryption
2. Introduction to the TFHE cryptosystem
3. Our contributions : a framework for fast evaluation of Boolean functions
4. Bonus : the Bootstrapping and its transformation

# Outline

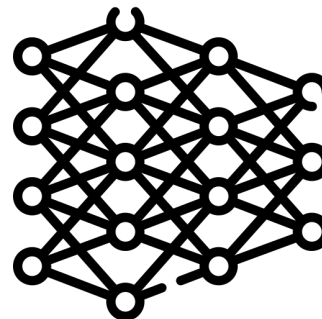
- 1 . Introduction to Fully Homomorphic Encryption
2. Introduction to the TFHE cryptosystem
3. Our contributions : a framework for fast evaluation of Boolean functions
4. Bonus : the Bootstrapping and its transformation

# What is FHE ?

Client



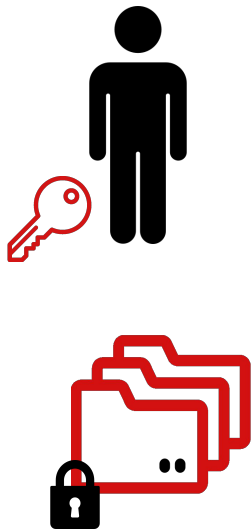
Server



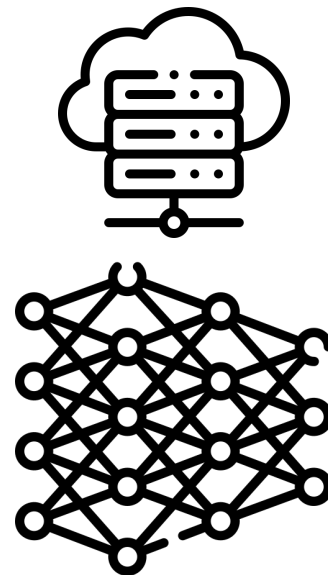
The client wants to use the neural network on its data.

# What is FHE ?

Client



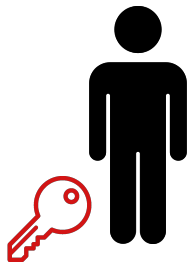
Server



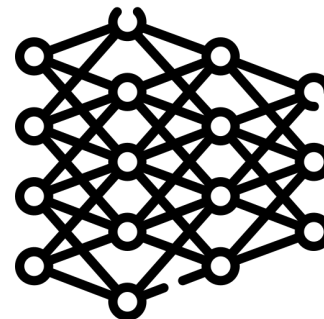
The client encrypts its data to protect its confidentiality during transfer.

# What is FHE ?

Client



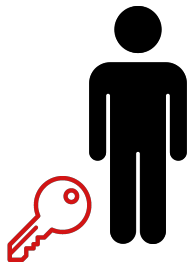
Server



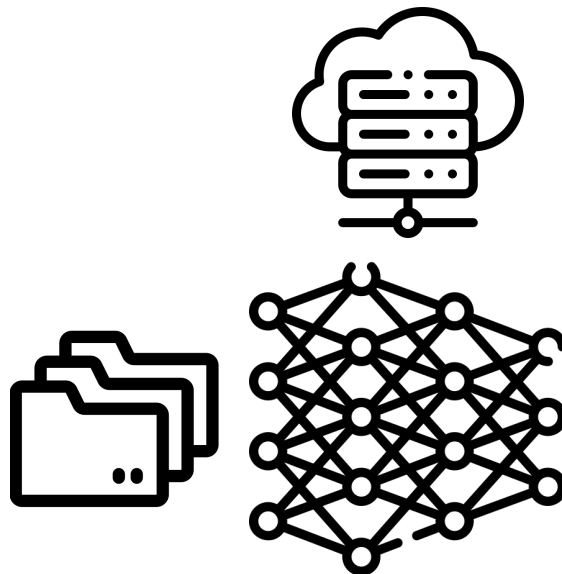
The server gets the encrypted data.

# What is FHE ?

Client



Server



The server needs to **decrypt** to be able to use the neural network, which breaks confidentiality !

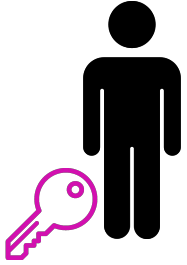
# FHE is a solution to this problem

- The encryption is made with a **homomorphic** scheme
- In this scenario, the server runs a **homomorphized** version of the neural network
- All computations can be performed **without any decryption or information leak.**



# FHE is a solution to this problem

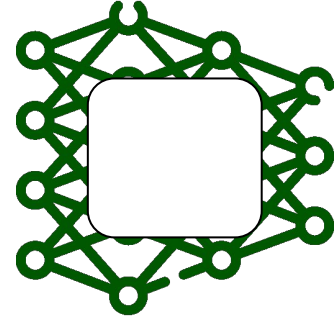
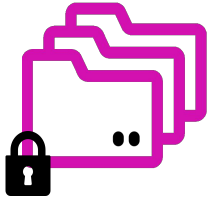
Client



Server

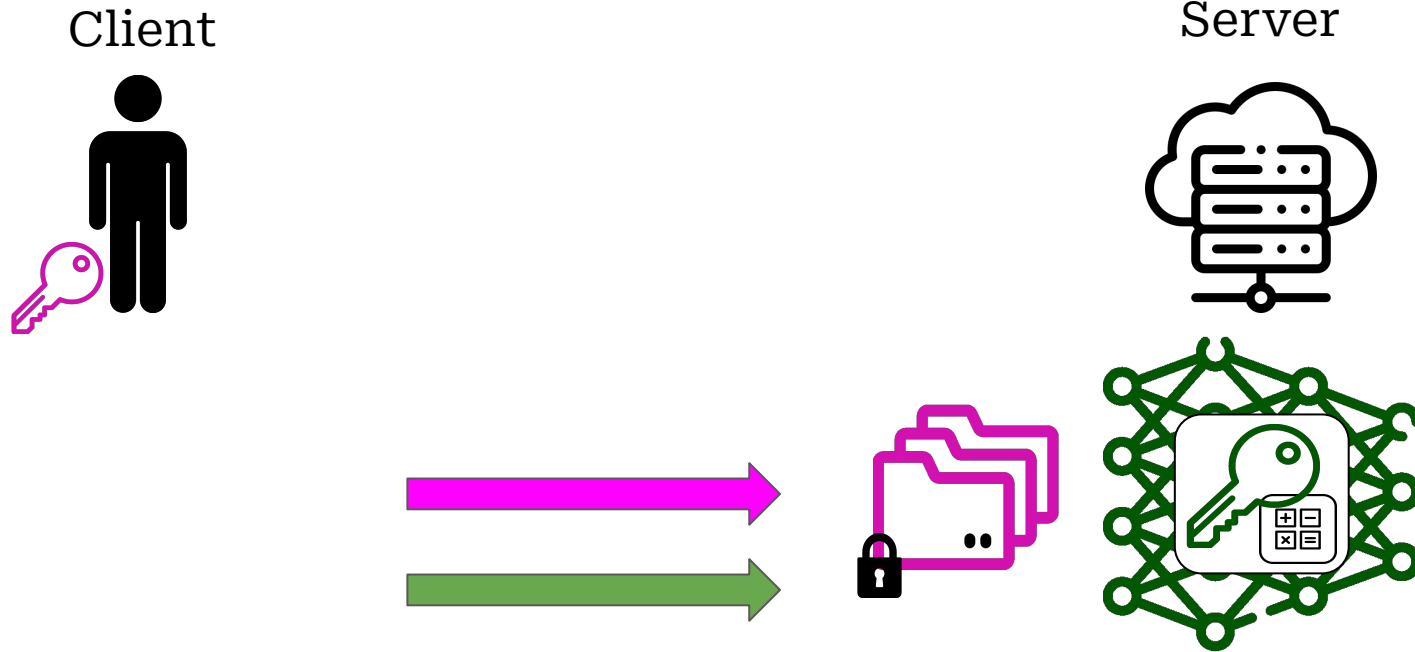


Evaluation key



The client encrypts its data and crafts an evaluation key that will be used in the homomorphized neural network.

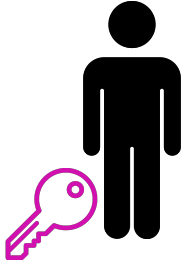
# FHE is a solution to this problem



The server gets the encrypted data and the evaluation key.

# FHE is a solution to this problem

Client



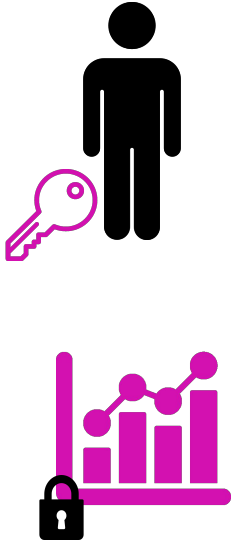
Server



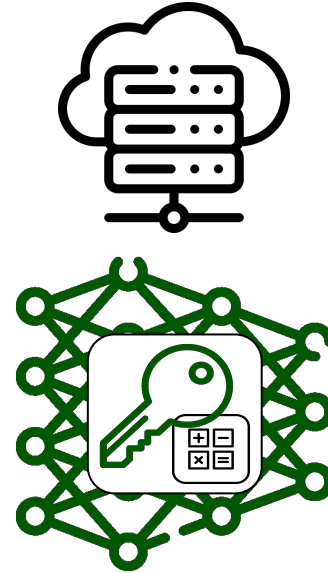
Thanks to the evaluation key, the server evaluates the neural network on the data **without decryption** and gets a result in an encrypted form

# FHE is a solution to this problem

Client



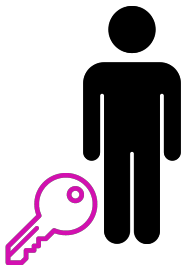
Server



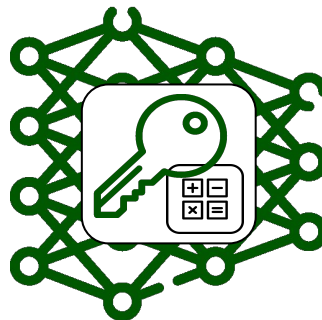
The server sends back the encrypted result to the client.

# FHE is a solution to this problem

Client



Server

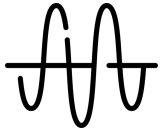


The client can then decrypt the result !

# Main challenges of FHE



Performances: Overhead in time and in size



Noise control: risk of losing correctness



Limited set of supported homomorphic operations

# Outline

- 1 . Introduction to Fully Homomorphic Encryption
2. Introduction to the TFHE cryptosystem
3. Our contributions : a framework for fast evaluation of Boolean functions
4. Bonus : the Bootstrapping and its transformation

# TFHE : description of the scheme

- Very fast by FHE standards
- Potentially infinite series of operations
- Evaluation of encrypted look-up tables : possibility to evaluate any univariate function
- But precisions on plaintexts limited to a few bits

## TFHE: Fast Fully Homomorphic Encryption over the Torus\*

Ilaria Chillotti<sup>1</sup>, Nicolas Gama<sup>3,2</sup>, Mariya Georgieva<sup>4,3</sup>, and Malika Izabachène<sup>5</sup>

<sup>1</sup> imec-COSIC, KU Leuven,  
Kasteelpark Arenberg 10, Bus 2452, B-3001 Leuven-Heverlee, Belgium  
ilaria.chillotti@kuleuven.be

<sup>2</sup> Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université  
Paris-Saclay, 78035 Versailles, France

<sup>3</sup> Inpher, Lausanne, Switzerland  
nicolas@inpher.io, mariya@inpher.io

<sup>4</sup> EPFL, Route Cantonale, CH-1015 Lausanne, Switzerland

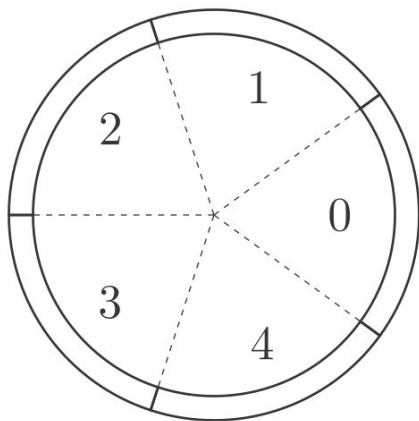
<sup>5</sup> CEA, LIST, Point Courrier 172, 91191 Gif-sur-Yvette Cedex, France  
malika.izabachene@cea.fr

**Abstract.** This work describes a fast fully homomorphic encryption scheme over the torus (TFHE), that revisits, generalizes and improves the fully homomorphic encryption (FHE) based on GSW and its ring variants. The simplest FHE schemes consist in bootstrapped binary gates. In this gate bootstrapping mode, we show that the scheme FHEW of [29] can be expressed only in terms of external product between a GSW and a LWE ciphertext. As a consequence of this result and of other optimizations, we decrease the running time of their bootstrapping from 690ms to 13ms single core, using 16MB bootstrapping key instead of 1GB, and preserving the security parameter. In leveled homomorphic mode, we propose two methods to manipulate packed data, in order to decrease the ciphertext expansion and to optimize the evaluation of look-up tables and arbitrary functions in RingGSW based homomorphic schemes. We also extend the automata logic, introduced in [31], to the efficient leveled evaluation of weighted automata, and present a new homomorphic counter called TBSR, that supports all the elementary operations that occur in a multiplication. These improvements speed-up the evaluation of most arithmetic functions in a packed leveled mode, with a noise overhead that remains additive. We finally present a new circuit bootstrapping that converts LWE ciphertexts into low-noise RingGSW ciphertexts



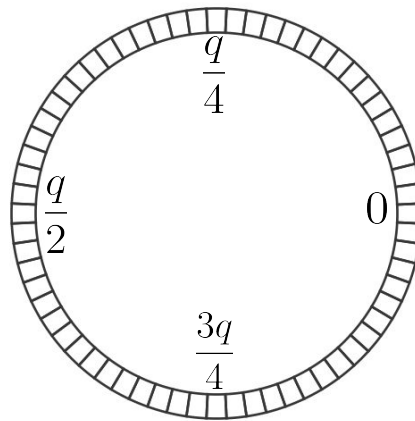
# TFHE : description of the scheme

Clear space:  $\mathbb{T}_p$



$p$  has a size of few bits.

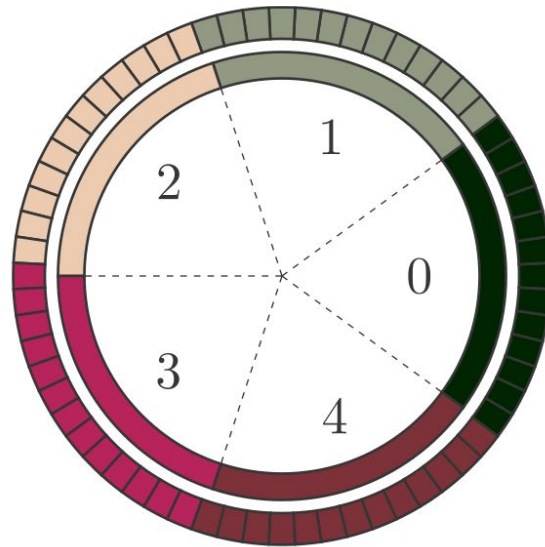
Encrypted space:  $\mathbb{T}_q$



$q = 2^{32}$  or  $2^{64}$

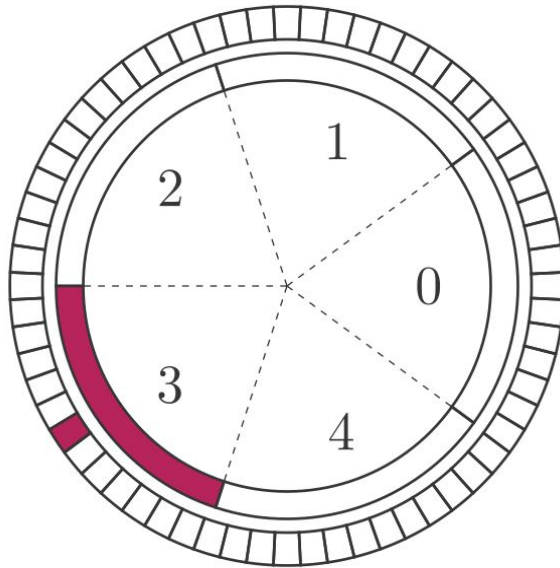
# TFHE : description of the scheme

Natural embedding of  $\mathbb{T}_p$  in  $\mathbb{T}_q$



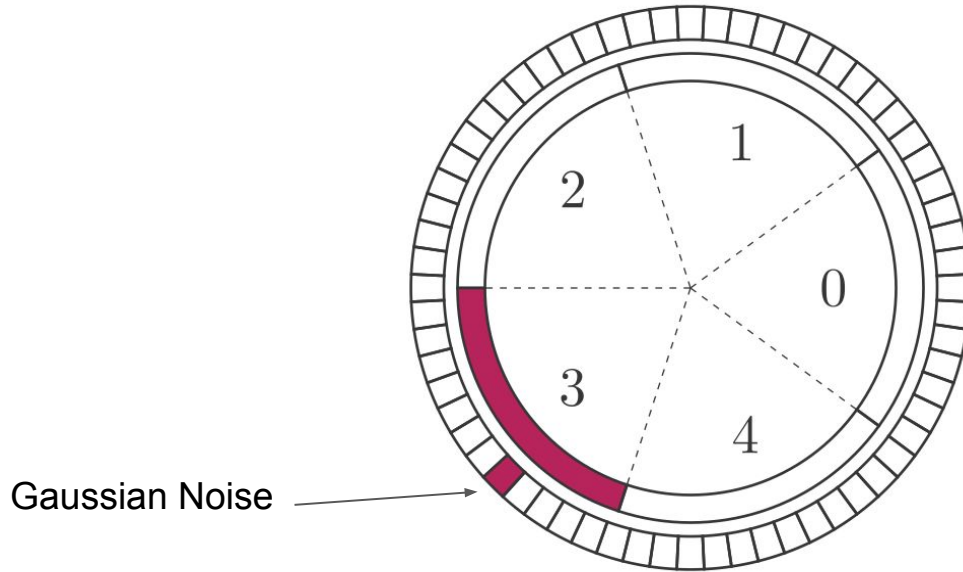
# TFHE : description of the scheme

Encoding of a message  $m \in \mathbb{T}_p$



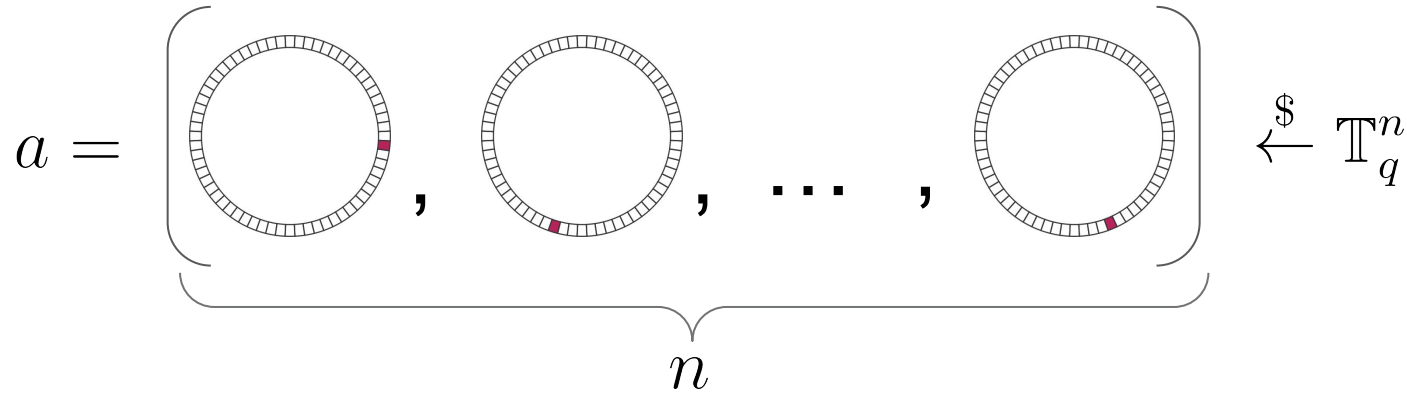
# TFHE : description of the scheme

Encoding of a message  $m \in \mathbb{T}_p$



# TFHE : description of the scheme

Sampling of a mask :



# TFHE : description of the scheme

Sampling of a mask :

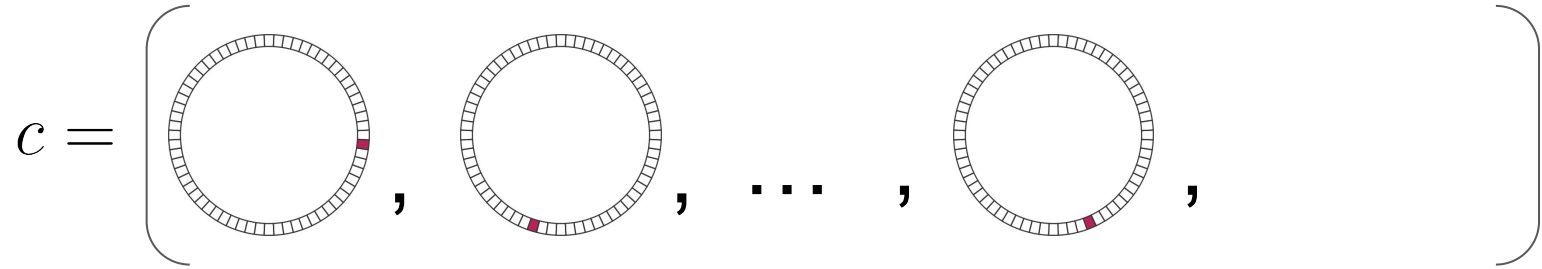
$$a = \left( \underbrace{\left( \text{circle}_1, \text{circle}_2, \dots, \text{circle}_n \right)}_n \right) \stackrel{\$}{\leftarrow} \mathbb{T}_q^n$$

Secret key:

$$s_k = (1, 0, \dots, 1) \stackrel{\$}{\leftarrow} \mathbb{B}^n$$

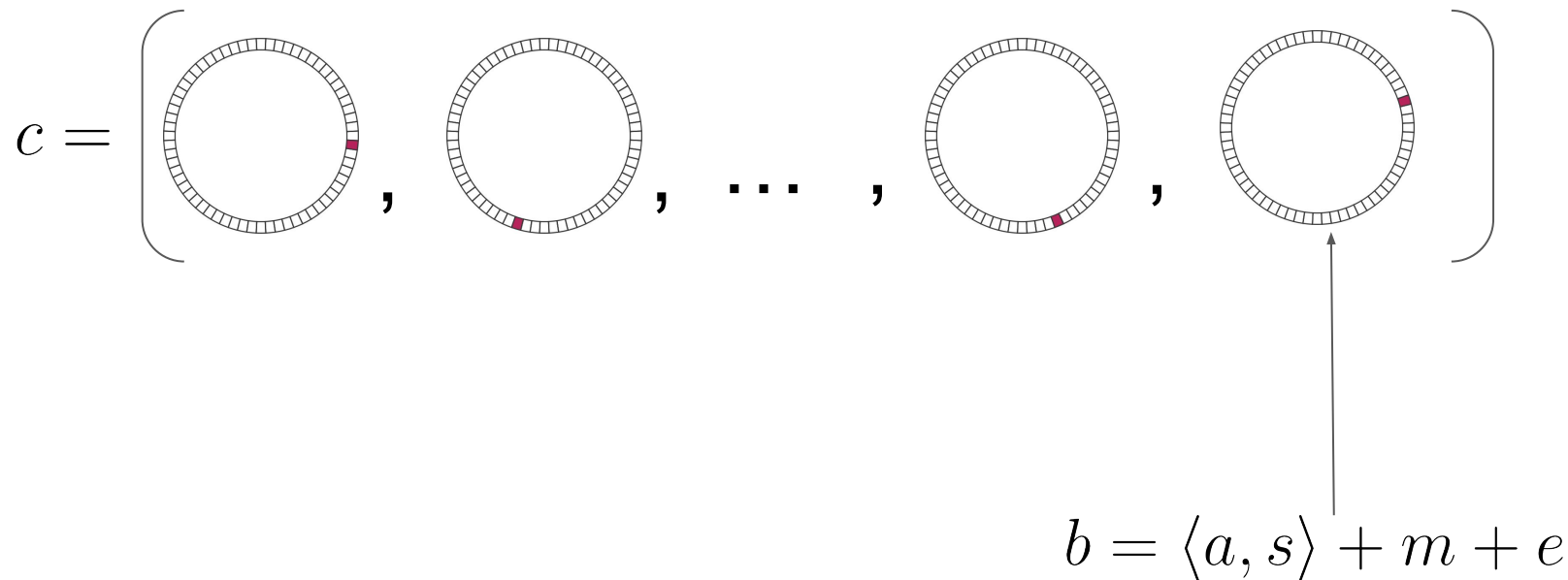
# TFHE : description of the scheme

Construction of ciphertexts :



# TFHE : description of the scheme

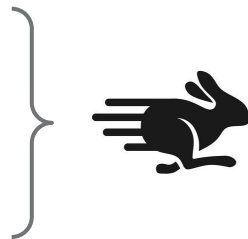
Construction of ciphertexts :





# TFHE: available operations

- Sum on  $\mathbb{T}_p$
- External product on  $\mathbb{T}_p$  by a clear constant



- Programmable Bootstrapping

Resets the noise level



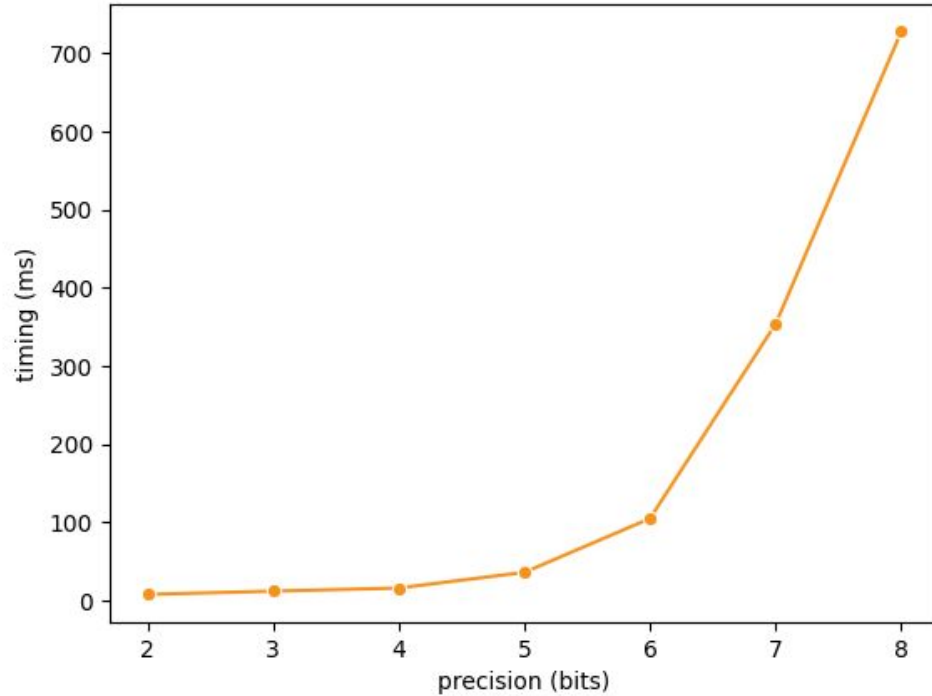
Evaluates any Look-up table on the ciphertext



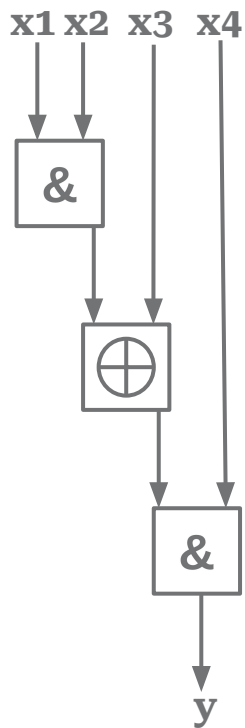
**BUT** slow and heavy operation



# TFHE only allows small precisions

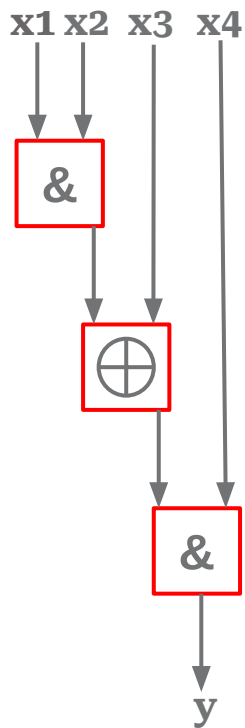


# Natural approach of Boolean function evaluation: gate bootstrapping



- See Boolean functions as Boolean circuits
- Each bit is a ciphertext
- Each gate is a 2-input Look-up table

# Natural approach of Boolean function evaluation: gate bootstrapping



- See Boolean functions as Boolean circuits
- Each bit is a ciphertext
- Each gate is a 2-input Look-up table

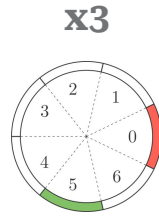
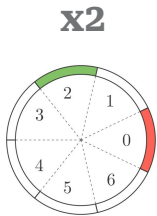
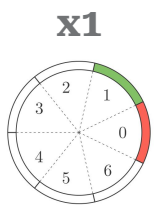
Problem: each gate costs 1 Programmable Bootstrapping

# Outline

- 1 . Introduction to Fully Homomorphic Encryption
2. Introduction to the TFHE cryptosystem
3. Our contributions : a framework for fast evaluation of Boolean functions
4. Bonus : the Bootstrapping and its transformation

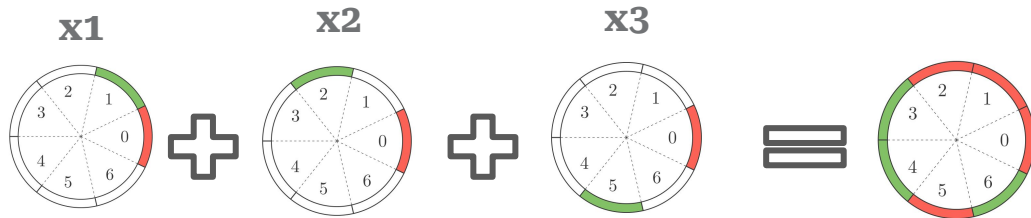
# Overview of our strategy

- Pick a  $p$  (better if prime) and embed each bit in  $\mathbb{T}_p$



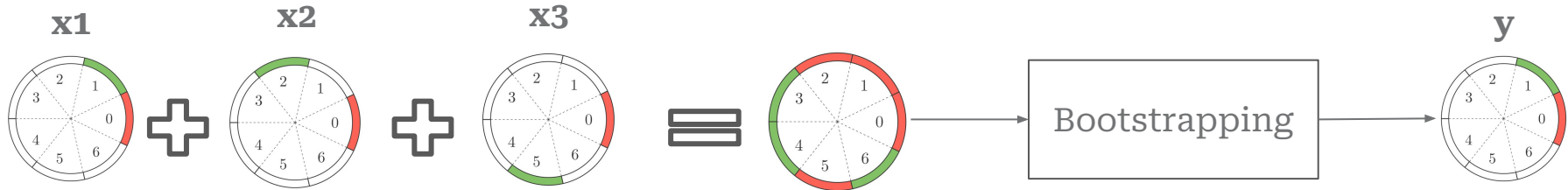
# Overview of our strategy

- Pick a  $p$  (better if prime) and embed each bit in  $\mathbb{T}_p$
- Compute the sum (fast !) and label the sectors according to the function we want to evaluate



# Overview of our strategy

- Pick a  $p$  (better if prime) and embed each bit in  $\mathbb{T}_p$
- Compute the sum (fast !) and label the sectors according to the function we want to evaluate
- Compute a Bootstrapping on the sum and get a fresh ciphertext





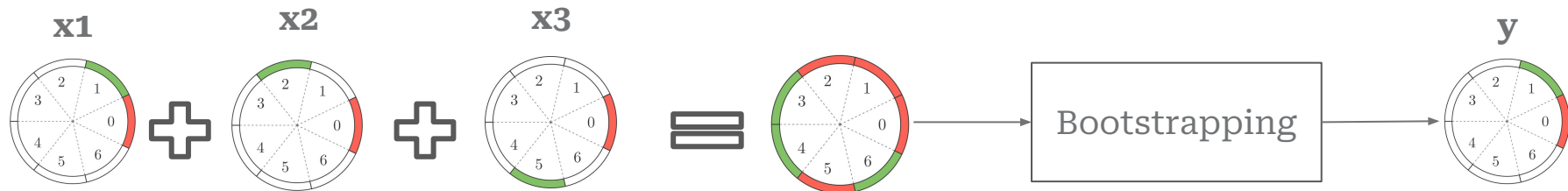
# Overview of our strategy

- Pick a  $p$  (better if prime) and embed each bit in  $\mathbb{T}_p$
- Compute the sum (fast !) and label the sectors according to the function we want to evaluate
- Compute a Bootstrapping on the sum and get a fresh ciphertext

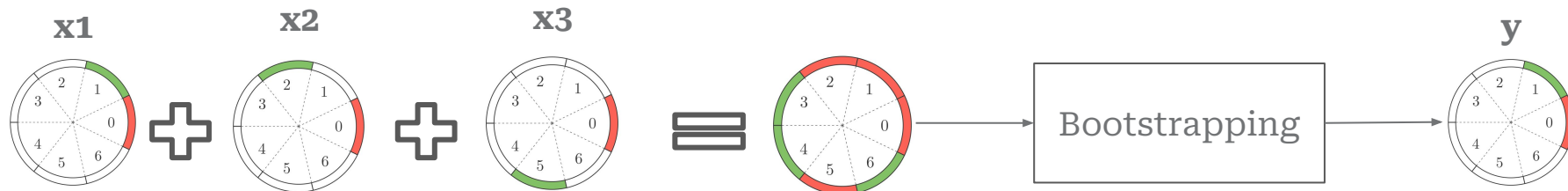
We do not use the notion of circuit anymore

We evaluate Boolean functions in one single bootstrapping no matter the number of inputs

We do not need to extend the LUT to fit more inputs



# Overview of our strategy



For a given function:

- How to select encodings such that the sum is valid (i.e. no overlap between true and false ciphertexts) ?
- Which  $p$  to use ? (the lower the better)

Our search algorithm finds the optimal solution to this problem

# Formalization of the notion of p-encodings

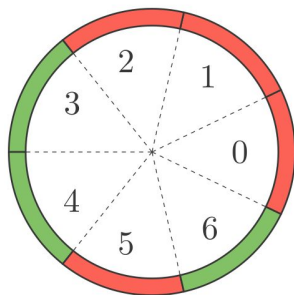
A p-encoding is a function  $\mathcal{E} : \mathbb{B} \mapsto 2^{\mathbb{Z}_p}$

$$\mathcal{E} = \begin{cases} 0 \mapsto \{\alpha_i\}_{0 \leq i \leq l_0} \\ 1 \mapsto \{\beta_i\}_{0 \leq i \leq l_1} \end{cases}$$

# Formalization of the notion of p-encodings

A p-encoding is a function  $\mathcal{E} : \mathbb{B} \mapsto 2^{\mathbb{Z}_p}$

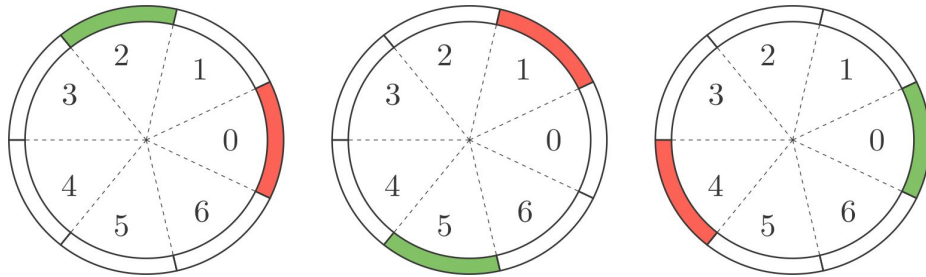
$$\mathcal{E} = \begin{cases} 0 \mapsto \{\alpha_i\}_{0 \leq i \leq l_0} \\ 1 \mapsto \{\beta_i\}_{0 \leq i \leq l_1} \end{cases}$$



# Boolean function on p-encodings

Let  $f$  be a Boolean function:  $f : \mathbb{B}^3 \mapsto \mathbb{B}$

Let  $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$  be three p-encodings.

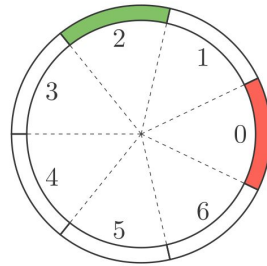


# Boolean function on p-encodings

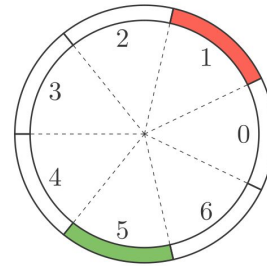
Truth table of f:

b1	b2	b3	f(b1, b2, b3)
0	0	0	
1	0	0	
0	1	0	

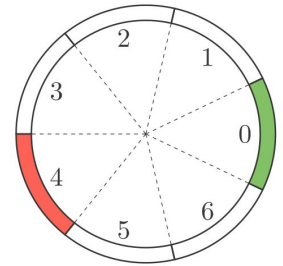
$\mathcal{E}_1$



$\mathcal{E}_2$



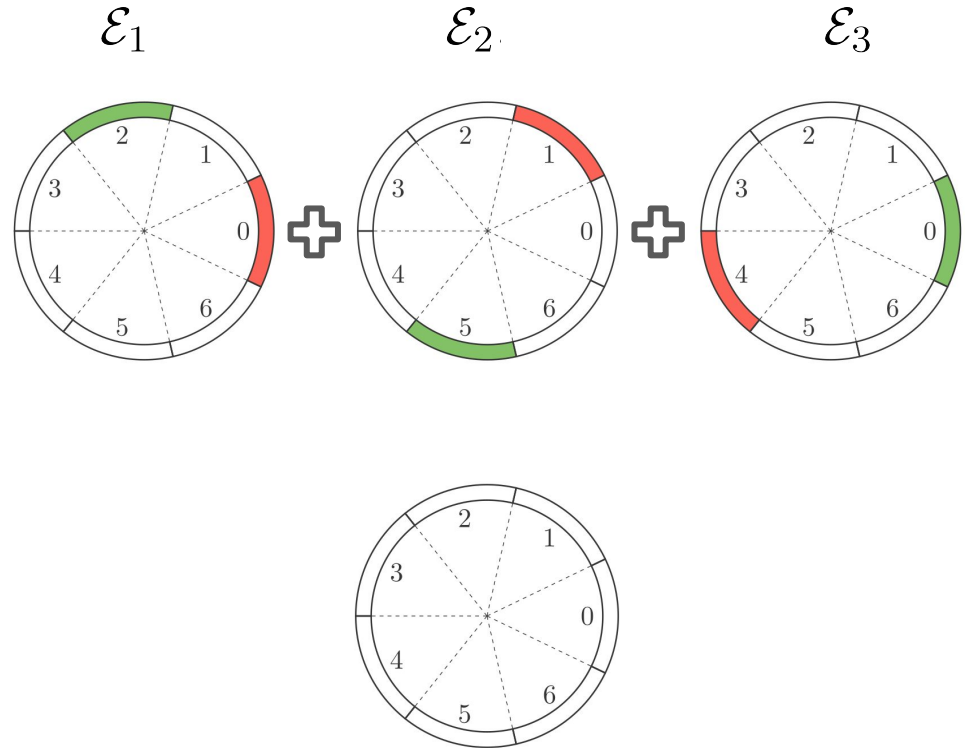
$\mathcal{E}_3$



# Boolean function on p-encodings

Truth table of f:

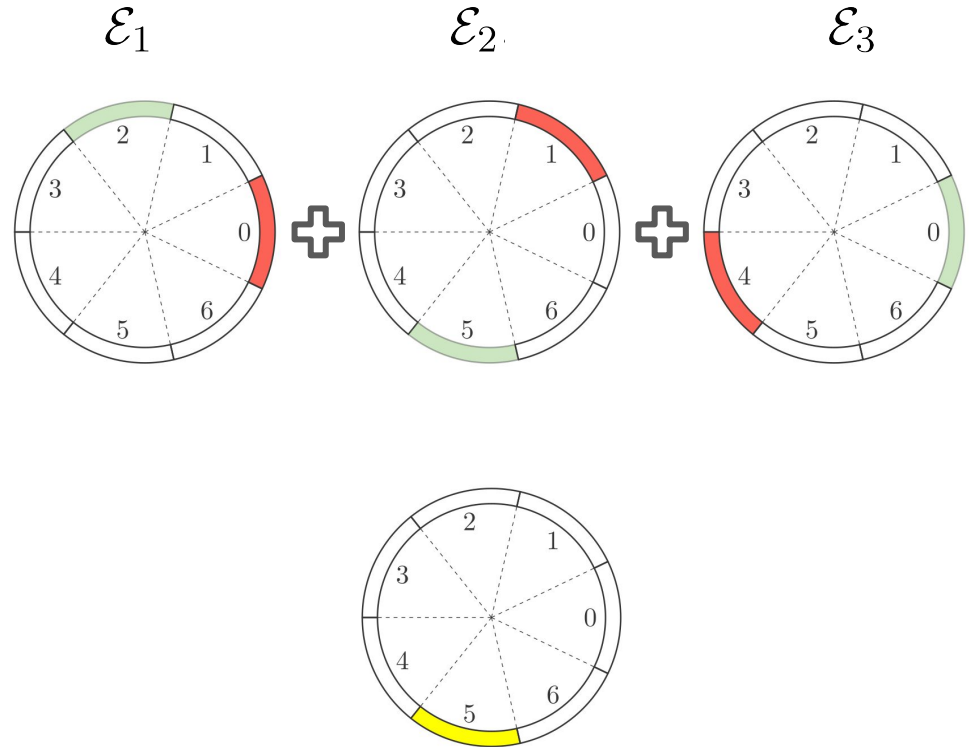
b1	b2	b3	f(b1, b2, b3)
0	0	0	
1	0	0	
0	1	0	



# Boolean function on p-encodings

Truth table of f:

b1	b2	b3	f(b1, b2, b3)
0	0	0	
1	0	0	
0	1	0	

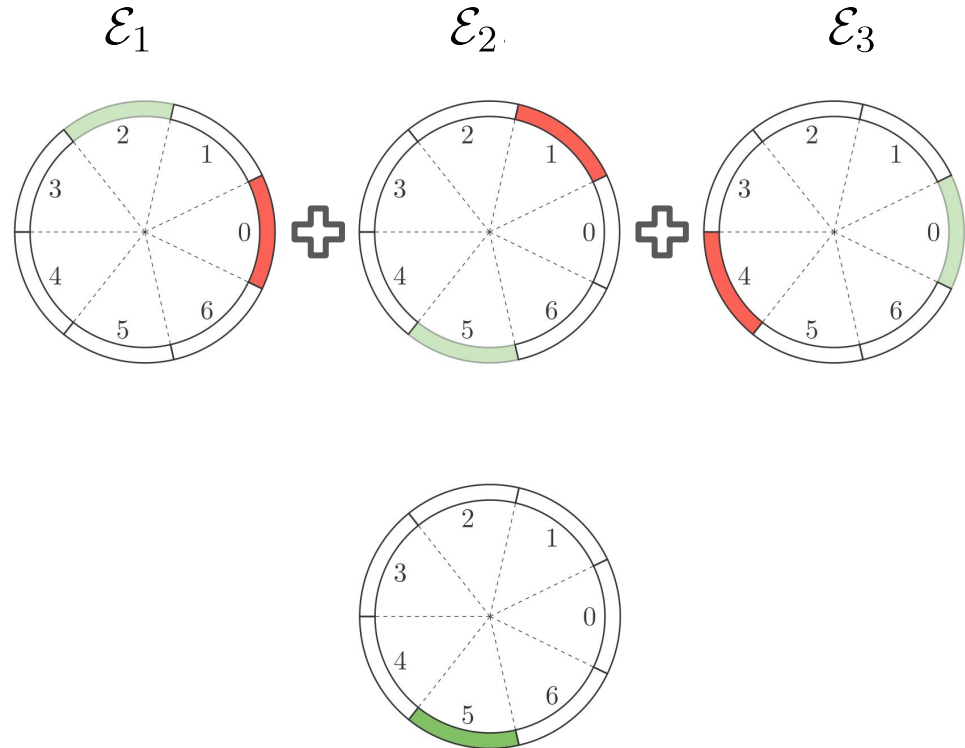




# Boolean function on p-encodings

Truth table of f:

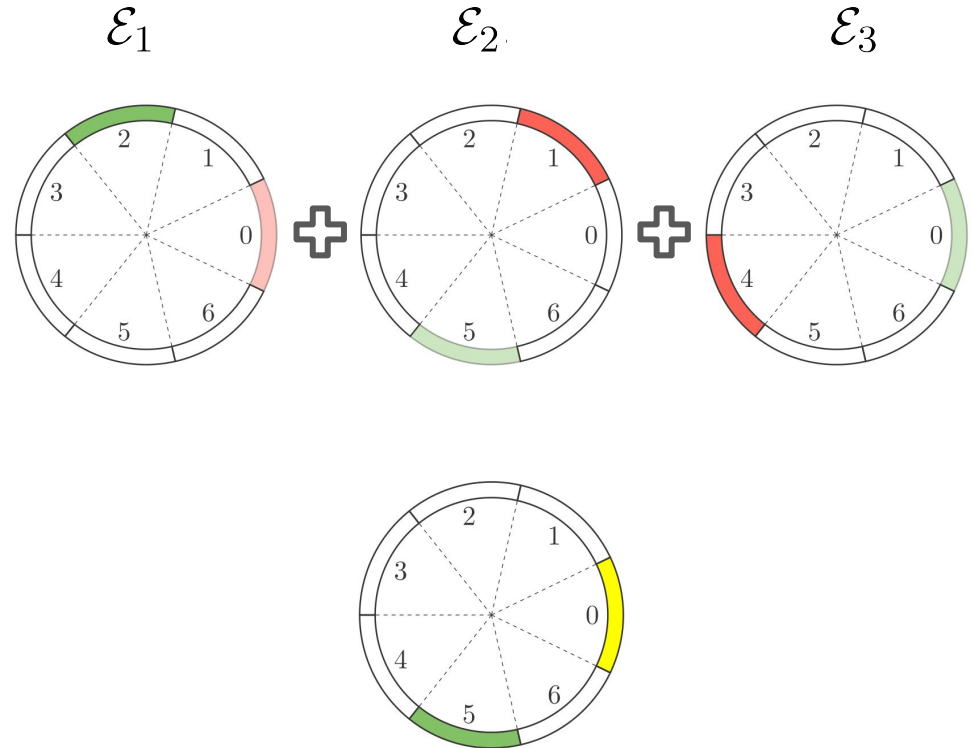
b1	b2	b3	f(b1, b2, b3)
0	0	0	1
1	0	0	
0	1	0	



# Boolean function on p-encodings

Truth table of f:

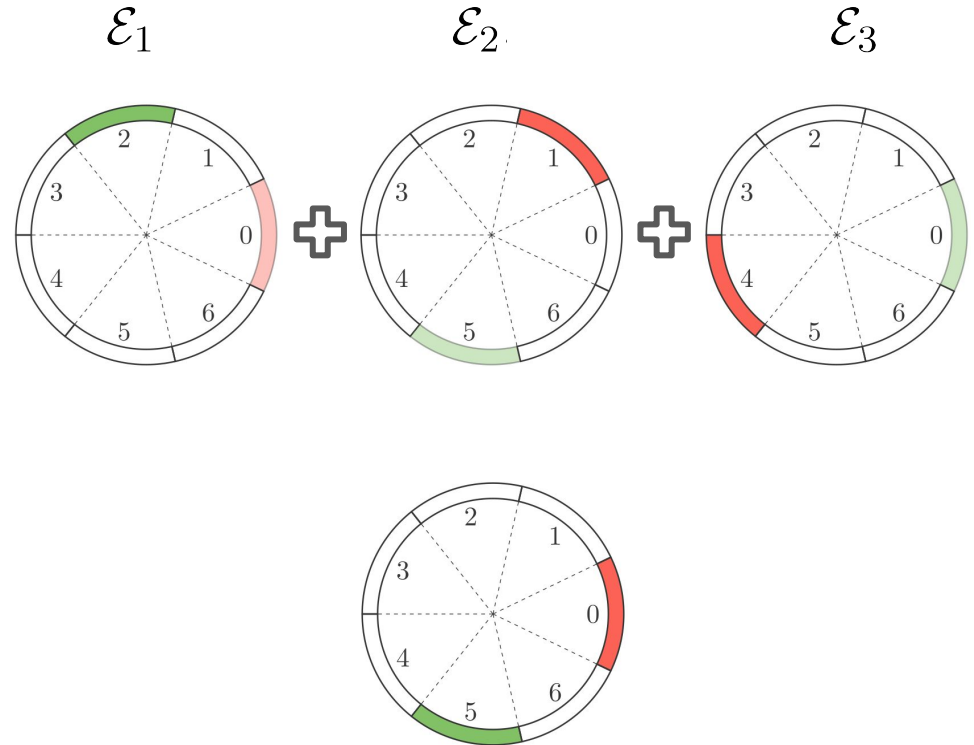
b1	b2	b3	f(b1, b2, b3)
0	0	0	1
1	0	0	
0	1	0	



# Boolean function on p-encodings

Truth table of f:

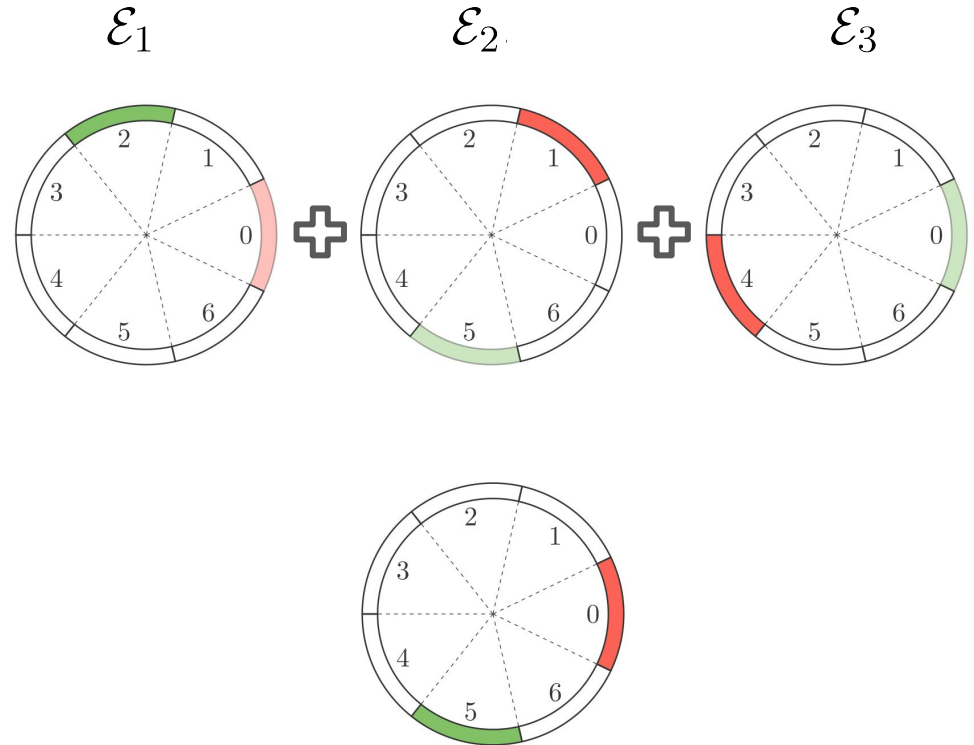
b1	b2	b3	f(b1, b2, b3)
0	0	0	1
1	0	0	0
0	1	0	



# Boolean function on p-encodings

Truth table of  $f$ :

b1	b2	b3	$f(b1, b2, b3)$
0	0	0	1
1	0	0	0
0	1	0	



The encodings are valid if the red and the green parts do not overlap after all the lines have been treated.

# Boolean function on p-encodings

Problem : for a given function, how to find a valid set of p-encodings (and the best p) ?

=> Exhaustive search. But some restrictions have to be made in the search space.

=> We restrict the search to p-encodings with form:

$$\mathcal{E}_i = \begin{cases} 0 \mapsto \{0\} \\ 1 \mapsto \{d_i\} \end{cases} \quad \text{with: } \mathcal{E}_1 = \begin{cases} 0 \mapsto \{0\} \\ 1 \mapsto \{1\} \end{cases}$$

with no loss of generality

## An other point of view on the problem

The encodings have the form  $\mathcal{E}_i = \begin{cases} 0 \mapsto \{0\} \\ 1 \mapsto \{d_i\} \end{cases}$

The truth table of the function can be rewritten as the linear system:

$$\begin{cases} 0 \cdot d_1 + 0 \cdot d_2 + \dots + 0 \cdot d_\ell = r_0 \\ 0 \cdot d_1 + 0 \cdot d_2 + \dots + 1 \cdot d_\ell = r_1 \\ \dots \\ 1 \cdot d_1 + 1 \cdot d_2 + \dots + 1 \cdot \dots \cdot d_\ell = r_{2^\ell} \end{cases}$$



## An other point of view on the problem

The encodings have the form  $\mathcal{E}_i = \begin{cases} 0 \mapsto \{0\} \\ 1 \mapsto \{d_i\} \end{cases}$

We can apply the coloration associated to the Boolean function f:

$$\begin{cases} 0 \cdot d_1 + 0 \cdot d_2 + \dots + 0 \cdot d_\ell = r_0 \\ 0 \cdot d_1 + 0 \cdot d_2 + \dots + 1 \cdot d_\ell = r_1 \\ \dots \\ 1 \cdot d_1 + 1 \cdot d_2 + \dots + 1 \cdot \dots \cdot d_\ell = r_{2^\ell} \end{cases}$$

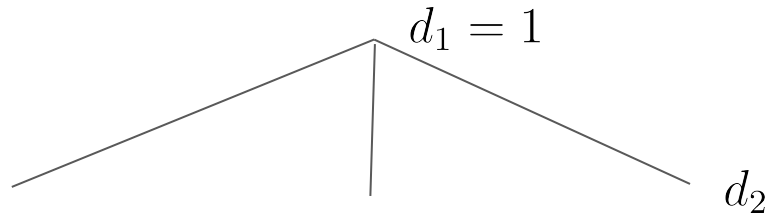
## An other point of view on the problem

By writing all the inequations   $\neq$   we get:

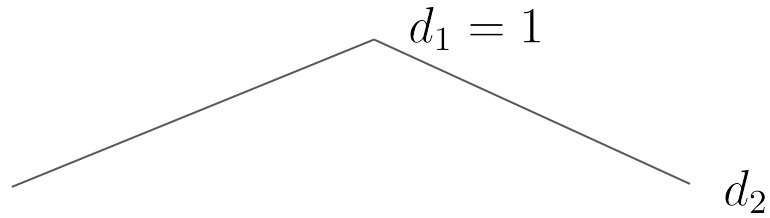
$$\begin{cases} c_1^{(1)} \cdot d_1 + \cdots + c_l^{(1)} \cdot d_l \neq 0 \pmod{p} \\ c_1^{(2)} \cdot d_1 + \cdots + c_l^{(2)} \cdot d_l \neq 0 \pmod{p} \\ \vdots \end{cases} \quad \text{with } c_i^{(j)} \in \{0, \pm 1\}$$



# The search algorithm

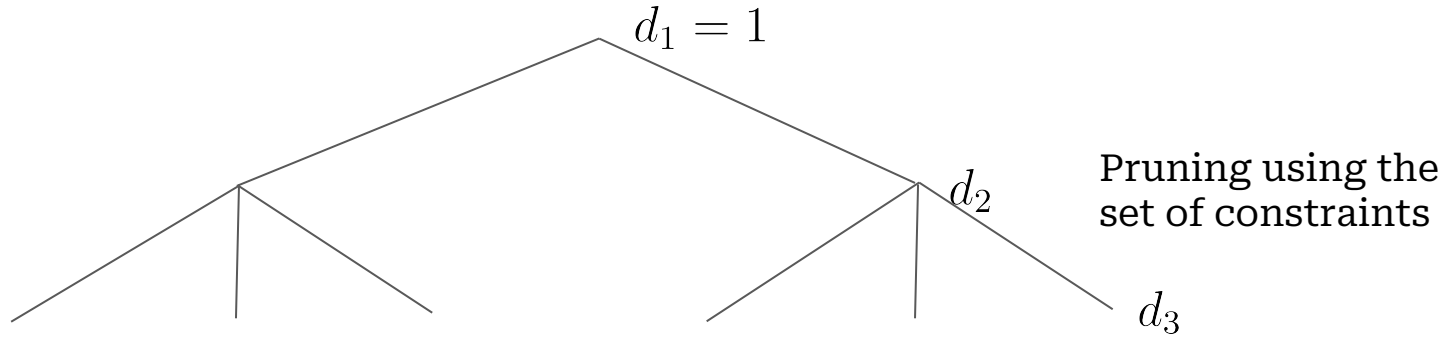


# The search algorithm

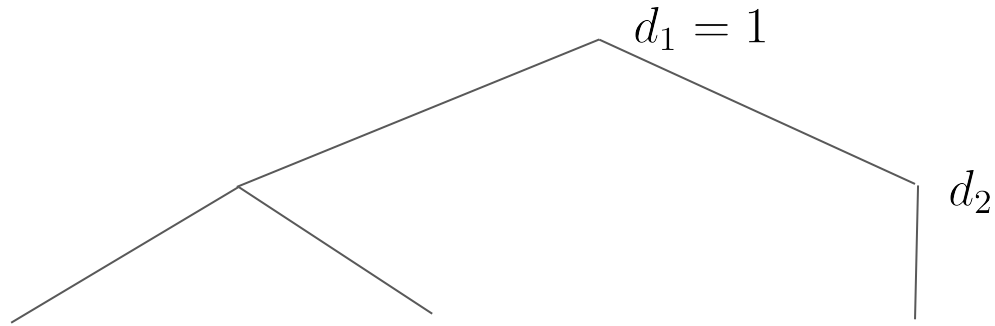


Pruning using the  
set of constraints

# The search algorithm



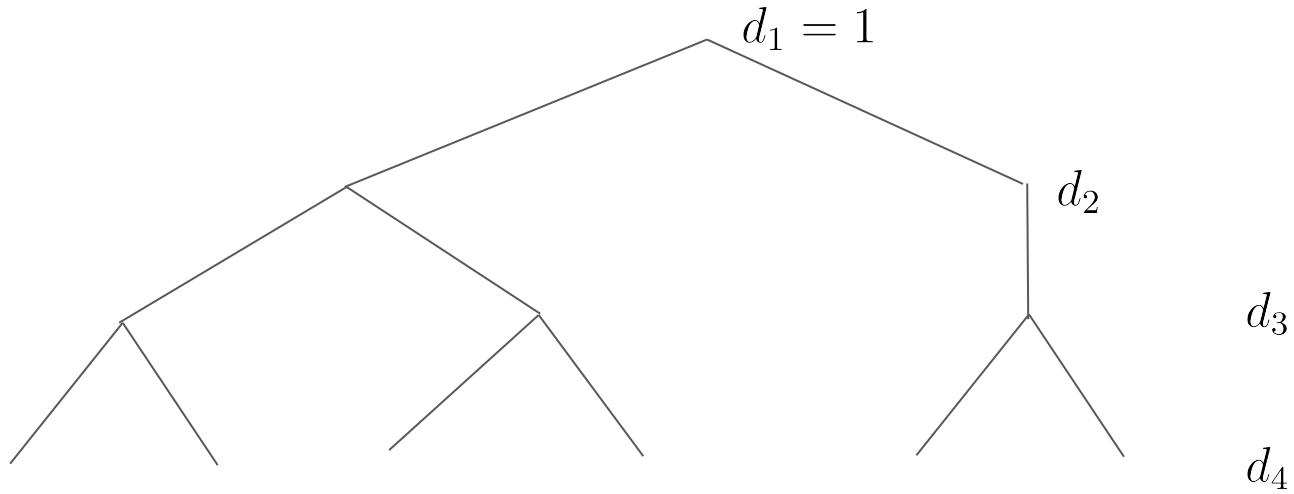
# The search algorithm



$d_3$

Pruning using the  
set of constraints

# The search algorithm



... until we find a  
path of length  $\ell$

# The search algorithm

- The search algorithm finds an **optimal** solution for a given  $p$ .
- To identify relevant values for  $p$  we developed an **heuristic** method that finds an upper bound on the optimal  $p$ .

## High-level idea for the heuristic

- Sampling values for  $d_i$ 's in  $\mathbb{Z}$ !
- Check divisibility of the result by  $p$ .
- The smallest  $p$  that does not divide any row gives an upper bound on the optimal solution

$$\left\{ \begin{array}{l} 0 \cdot d_1 + 0 \cdot d_2 + \cdots + 0 \cdot d_\ell = r_0 \\ 0 \cdot d_1 + 0 \cdot d_2 + \cdots + 1 \cdot d_\ell = r_1 \\ \cdots \\ 1 \cdot d_1 + 1 \cdot d_2 + \cdots + 1 \cdot \cdots d_\ell = r_{2^\ell} \end{array} \right.$$

# Application to cryptographic primitives

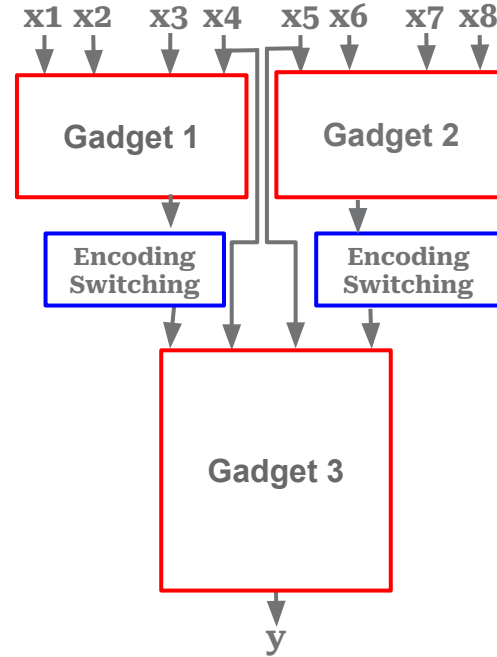
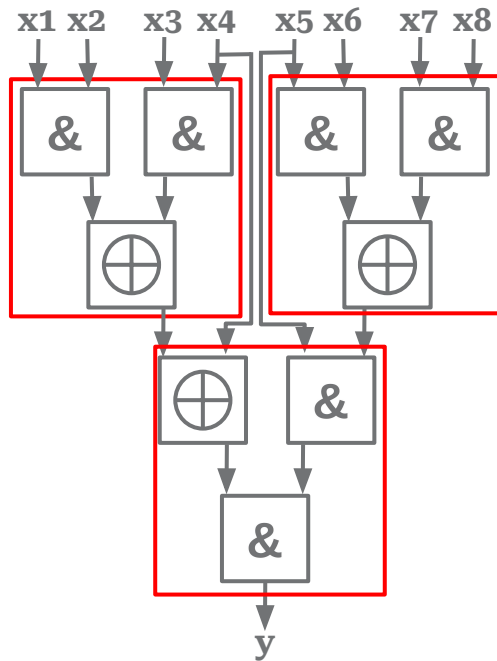
- For use-cases such as transciphering, OPRF, ...
- Efficient solutions for acceptable modulus for some lightweight block ciphers (namely SIMON, Trivium and ASCON) and hash function SHA-3.
- Our implementation beats the state of the art for these primitives.
- But no solution for AES: the circuit of the sbox is too complex !



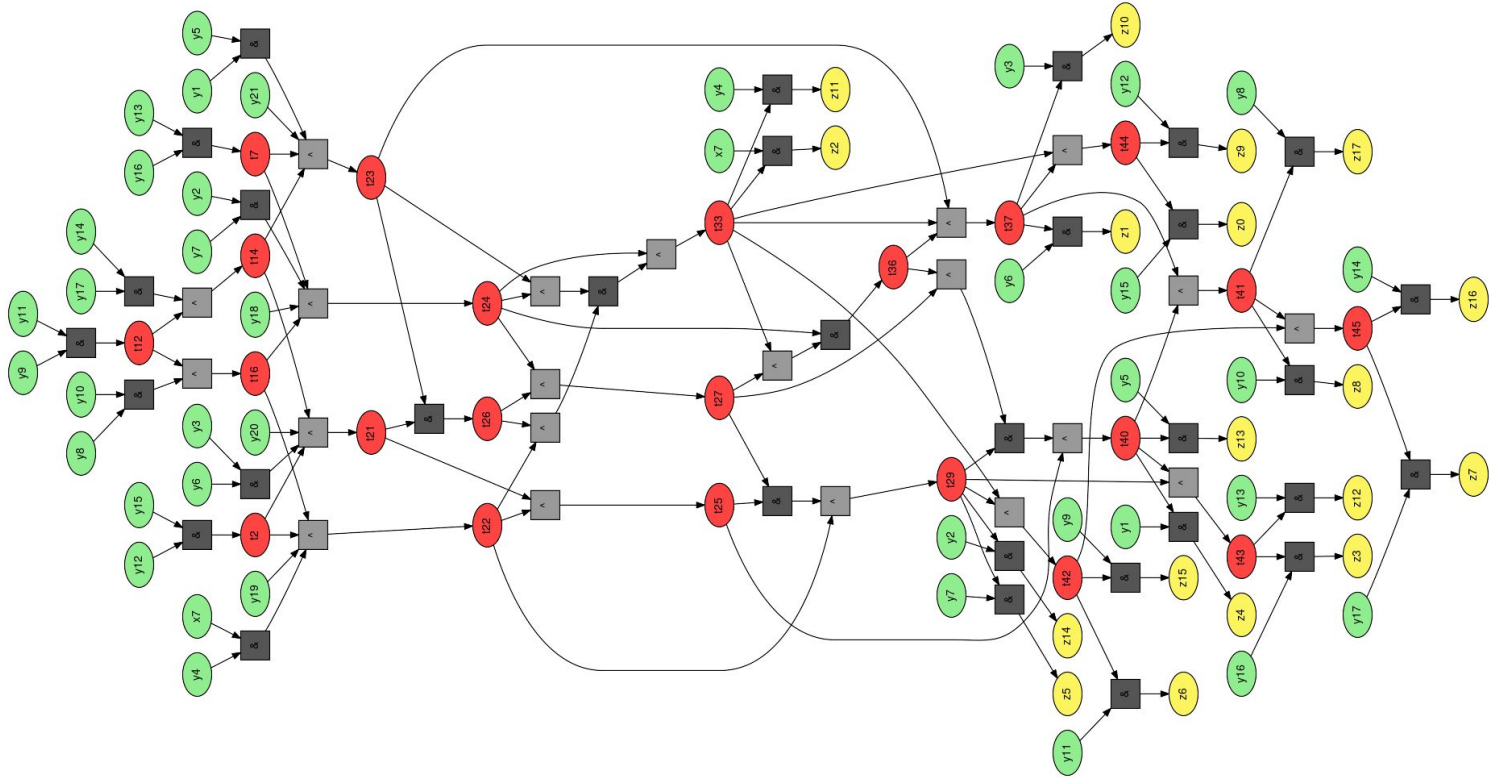
# Experimental results

Primitive	Section or Other work	Performances
One full run of SIMON	Gate Bootstrapping	174 s
	[BSS <sup>+</sup> 23] †	128 s
	Our work (Section 7.1)	10 s
One warm-up phase of Trivium (*)	Gate Bootstrapping	1498 s
	[BOS23] (estimation on our machine)	53 s
	Our work (Section 7.2)	32.8 s
One Full Keccak permutation (*)	Gate Bootstrapping	30.7 min
	Our work (Section 7.3)	8.8 min
One Ascon hashing (*)	Gate Bootstrapping	200s
	Our work (Section 7.4)	92 s

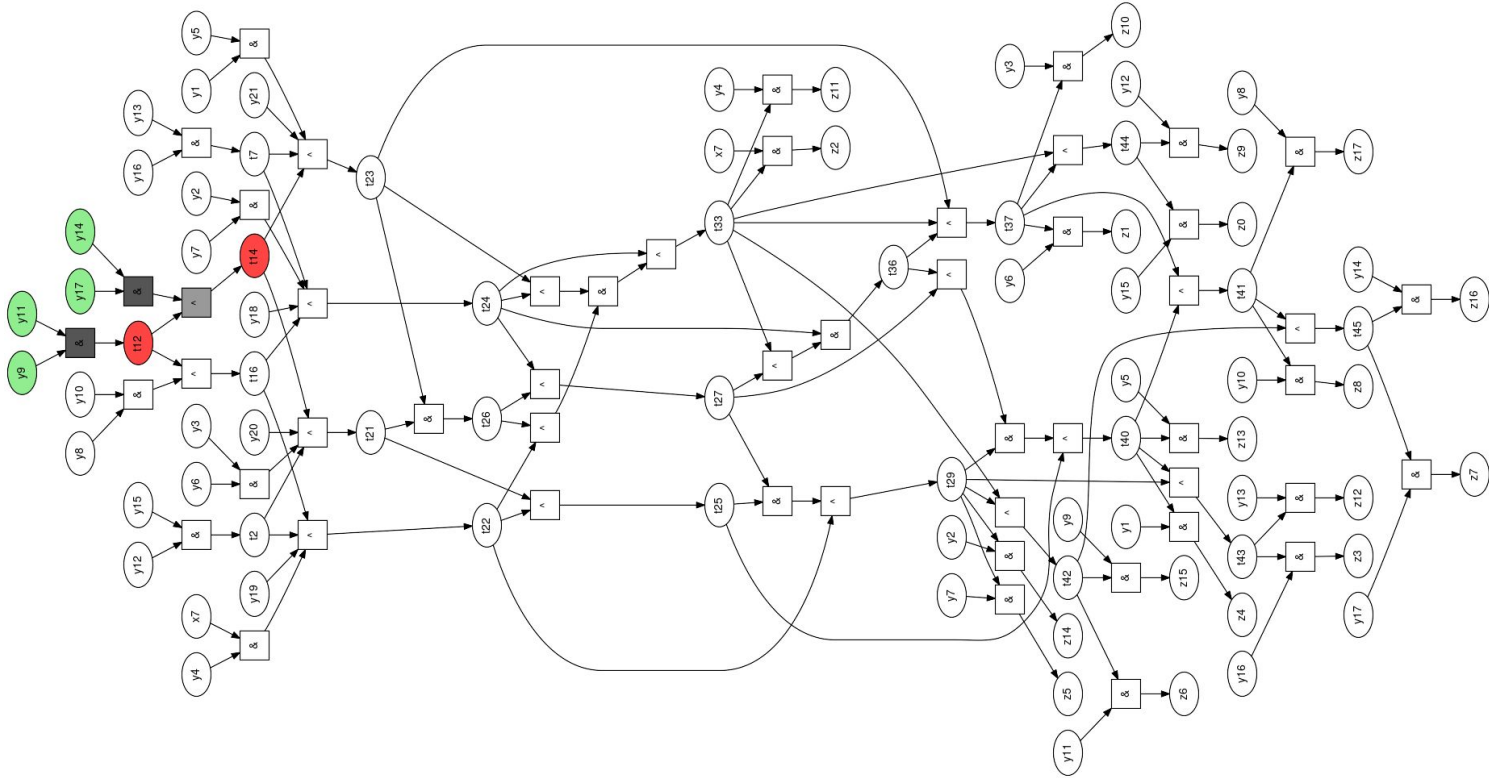
# Extension to bigger circuits



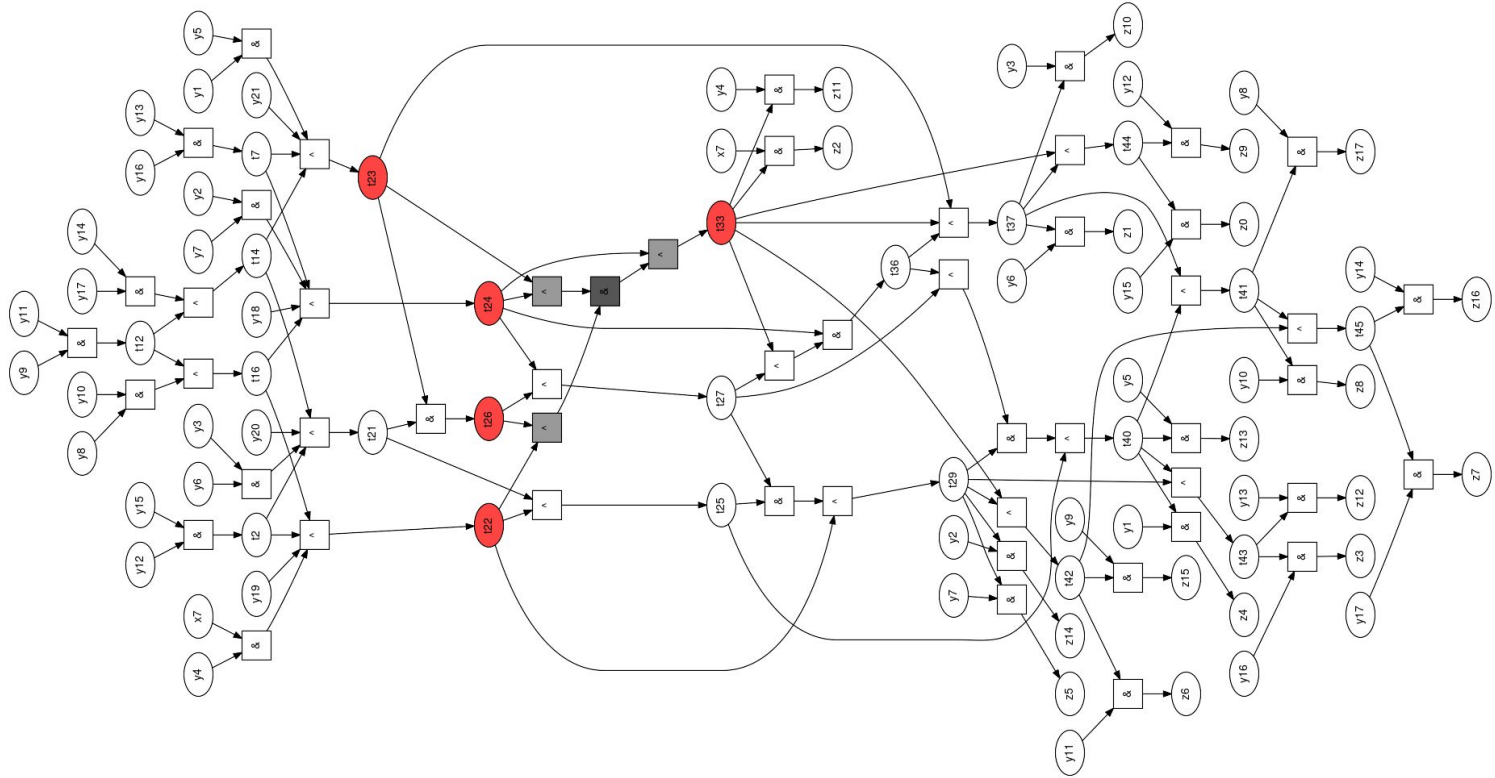
# Extension to bigger circuits (e.g. AES)



# Extension to bigger circuits (e.g. AES)



# Extension to bigger circuits (e.g. AES)



# AES: performances

One full evaluation of AES-128 ( $\epsilon = 2^{-23}$ ) on one thread	[GHS12] †	18 min
	[CLT14] †	5 min
	[TCBS23]	270 s
	Our work (Section 7.5)	103 s
One full evaluation of AES-128 ( $\epsilon = 2^{-40}$ ) on one thread	Gate Bootstrapping	234 s
	Our work (Real implementation)	135 s
	Our work (Theoretical timing with two keys)	105 s

One evaluation of the AES s-box	36 $\text{PBS}_{(11,4)}$
A full run of AES-128	5760 $\text{PBS}_{(11,4)}$ + 1280 $\text{PBS}_{(2,1)}$

# Conclusion

- New Framework to evaluate Boolean functions in TFHE
- One bootstrapping per function, with any number of input. Fixed size.
- Optimal algorithm to find a solution for a given function
- Heuristic to split bigger circuits into evaluable functions
- Adaptation of the bootstrapping to remove the padding bit

# Outline

- 1 . Introduction to Fully Homomorphic Encryption
2. Introduction to the TFHE cryptosystem
3. Our contributions : a framework for fast evaluation of Boolean functions
4. Bonus : the Bootstrapping and its transformation



## How does the bootstrapping works ?

We use polynomials in the ring  $\mathbb{Z}[X]/(X^N + 1)$  :

Let  $v(X) = v_0 + v_1 \cdot X + \dots + v_{N-1}X^{N-1}$

In this ring, something interesting happen:

$$X^{-a} \cdot v(X) = v_a + v_{a+1} \cdot X + \dots \text{ if: } a \in [0, N[$$

## How does the bootstrapping works ?

We use polynomials in the ring  $\mathbb{Z}[X]/(X^N + 1)$  :

Let  $v(X) = v_0 + v_1 \cdot X + \dots + v_{N-1}X^{N-1}$

In this ring, something interesting happen:

$$X^{-a} \cdot v(X) = v_a + v_{a+1} \cdot X + \dots \text{ if: } a \in [0, N[$$

and:

$$X^{-a} \cdot v(X) = -v_a - v_{a+1} \cdot X - \dots \text{ if: } a \in [N, 2N[$$

# How does the bootstrapping works ?

Let  $\mu$  be a ciphertext. It lives in  $\mathbb{Z}_q$ .

First thing to do is to change its modulus to send it in  $\mathbb{Z}_{2N}$

$$\tilde{\mu} = \left\lfloor 2N \frac{m + e}{q} \right\rfloor$$

# How does the bootstrapping works ?

Now if we compute:

$$X^{-\tilde{\mu}} \cdot v(X) = v_{\tilde{\mu}} + \dots \text{ if: } \tilde{\mu} < N$$

We can retrieve  $v_{\tilde{\mu}}$  by extracting the first coefficient.

But if  $\tilde{\mu} \geq N$  : It gets **negated**.

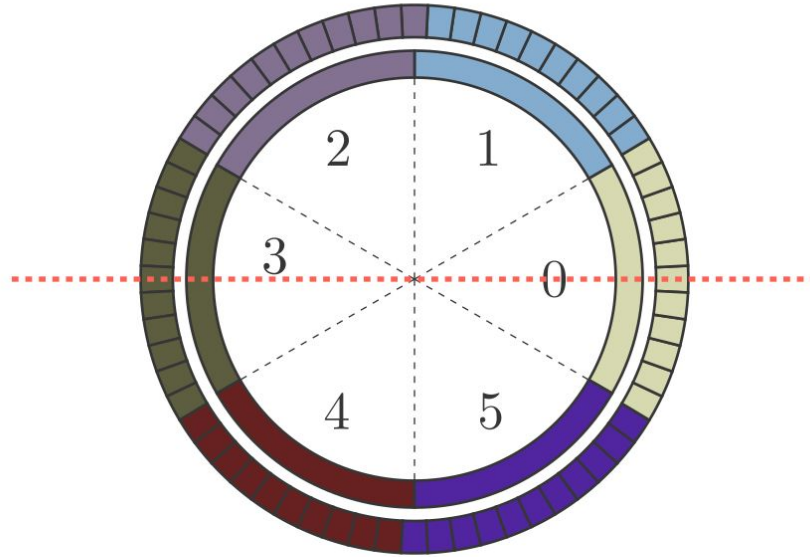
This is known as the **negacyclicity problem**

# How does the bootstrapping works ?

If we choose  $v_{\tilde{\mu}} = f(\tilde{\mu})$  we get an evaluation of Look-up Table !

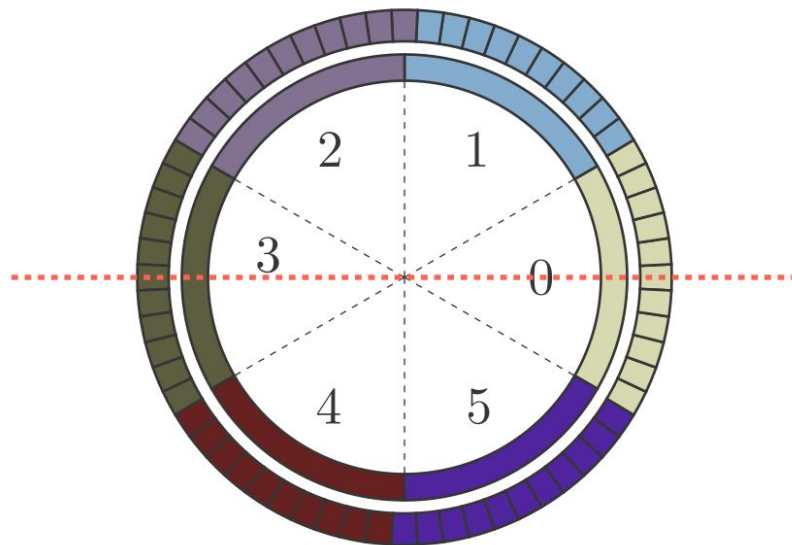
And we have control of the noise in the evaluation key and the coefficients of the polynomials, so we can reset the noise at a nominal level at the same time !

# How does the bootstrapping works



$$v(X) = \begin{array}{|c|c|c|c|} \hline f(0) & f(1) & f(2) & f(3) \\ \hline \end{array}$$

How does the bootstrapping works ?



$$v(X) = \begin{array}{|c|c|c|c|} \hline f(0) & f(1) & f(2) & f(3) \\ \hline \end{array}$$

$$X^N \cdot v(X) = \begin{array}{|c|c|c|c|} \hline -f(3) & -f(4) & -f(5) & -f(6) \\ \hline \end{array}$$

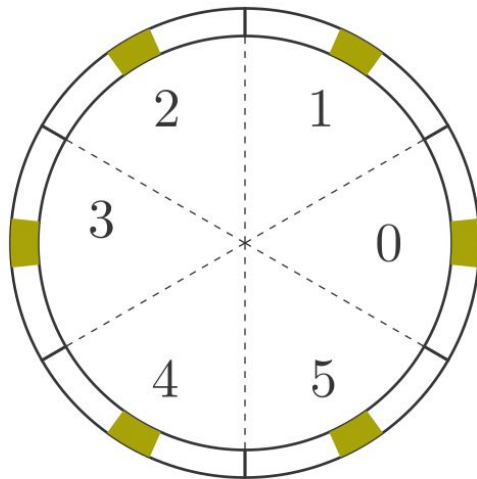
# Adaptation of the bootstrapping

- Common solution: fix the MSB to zero to stay in the upper part of the torus
- Problem: values may overflow in the MSB during homomorphic linear computations.
- Our solution uses *odd* values for  $p$  so the problem vanishes.



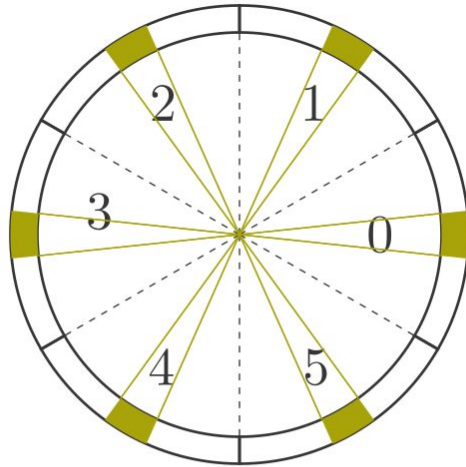
# Adaptation of the bootstrapping

Density of probability is not uniform across the torus:

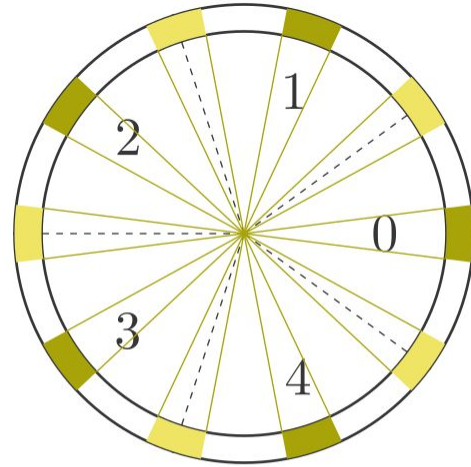


# Adaptation of the bootstrapping

With odd values, the “dense spots” do not face each others:



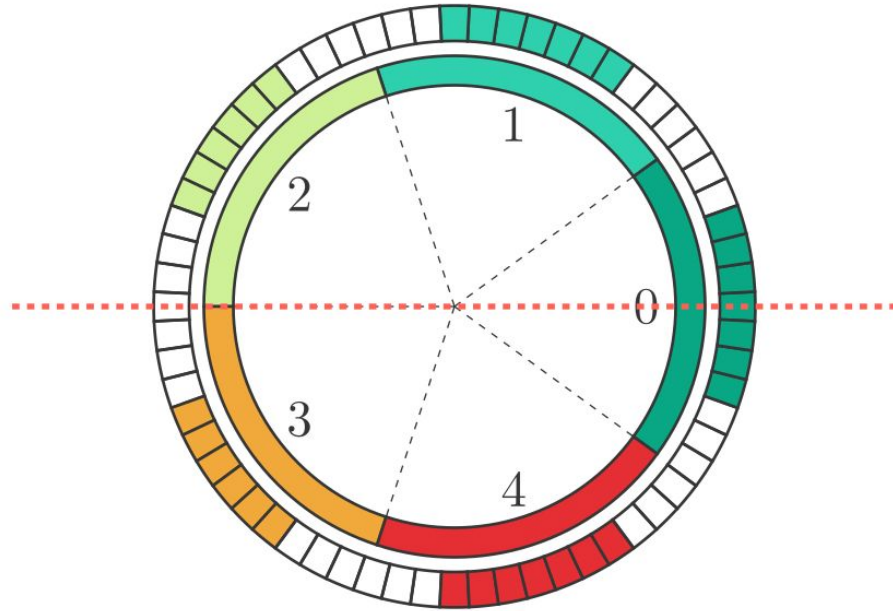
$$p = 6$$



$$p = 5$$

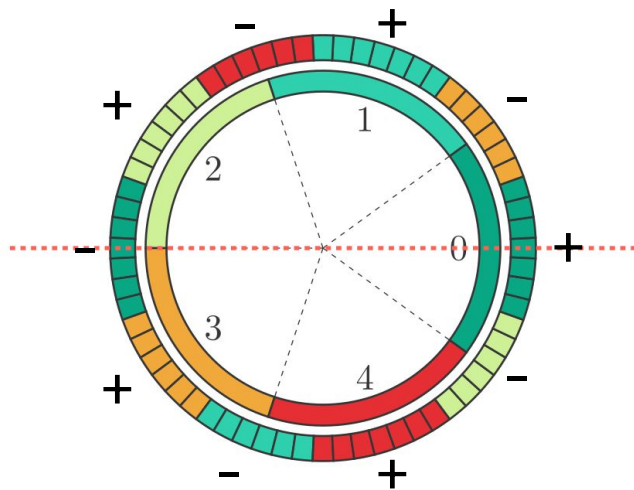
# Adaptation of the bootstrapping

Solution:





# Adaptation of the bootstrapping



$$v(X) = \begin{array}{|c|c|c|c|c|c|} \hline f(0) & -f(3) & f(1) & -f(4) & f(2) & -f(0) \\ \hline \end{array}$$

$$X^N \cdot v(X) = \begin{array}{|c|c|c|c|c|c|} \hline -f(0) & f(3) & -f(1) & f(4) & -f(2) & f(0) \\ \hline \end{array}$$

$$0 \quad \frac{N}{2p} \quad \frac{N}{p} \quad \frac{2N}{p} \quad \frac{3N}{p} \quad \frac{4N}{p}$$

Thank you !

<https://eprint.iacr.org/2023/1589>  
For more details