CRYPTOEXPERTS

WE INNOVATE TO SECURE YOUR BUSINESS

ENS

ÉCOLE NORMALE SUPÉRIEURE

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

Outline

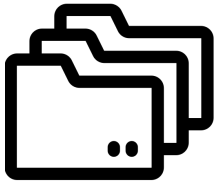
1 . Introduction to Fully Homomorphic Encryption

2. Introduction to the TFHE cryptosystem

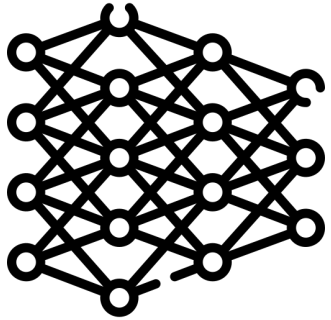
3. Our contributions : a framework for fast evaluation of Boolean functions

What is FHE ?

Client



Server



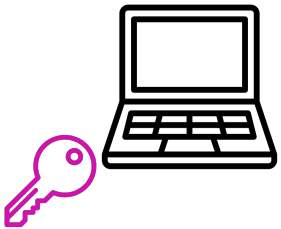
The client wants to use the neural network on its data.

FHE is a solution to this problem

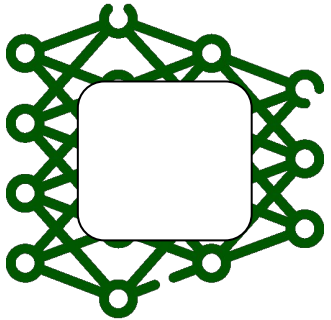
- The data is encrypted using a homomorphic scheme
- The server runs a homomorphized version of the neural network
- All computations can be performed without any decryption or information leak.

FHE is a solution to this problem

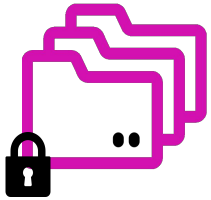
Client



Server

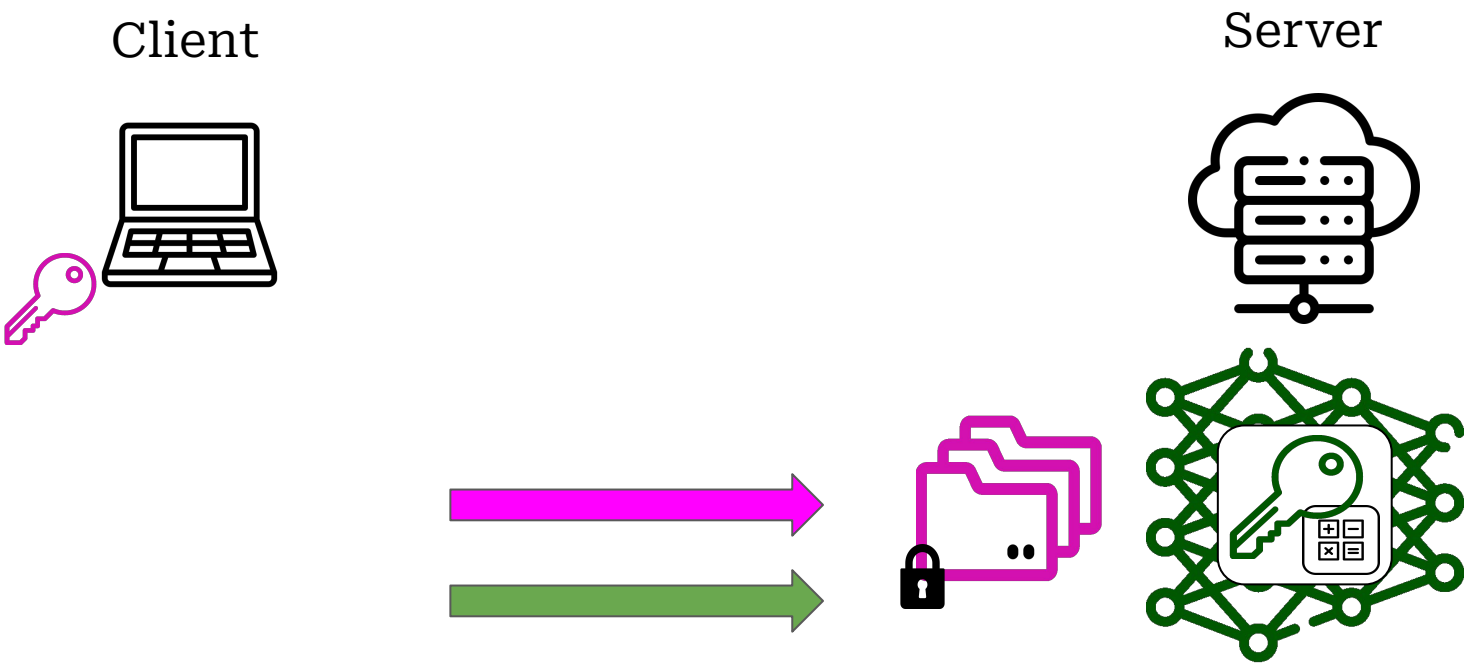


Evaluation key



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

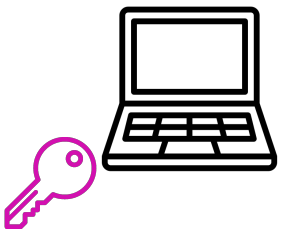
FHE is a solution to this problem



The server gets the encrypted data and the evaluation key.

FHE is a solution to this problem

Client



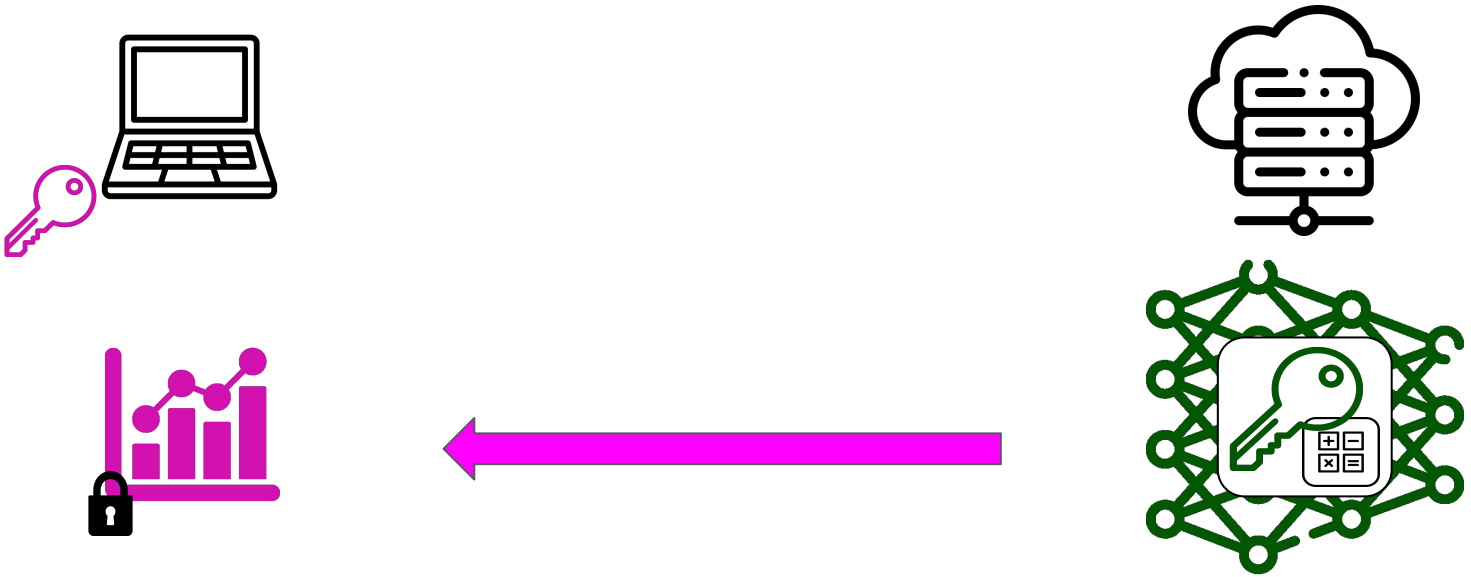
Server

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

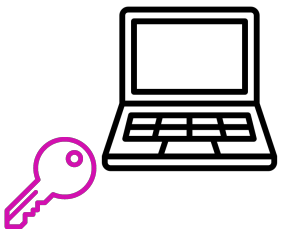
Server



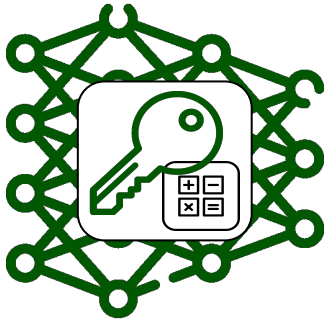
The server sends back the encrypted result to the client.

FHE is a solution to this problem

Client



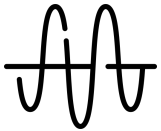
Server



The client can then decrypt the result !

Main challenges of FHE

Performances: Overhead in time and in memory



Noise control: risk of losing correctness

Limited set of supported homomorphic operations

Outline

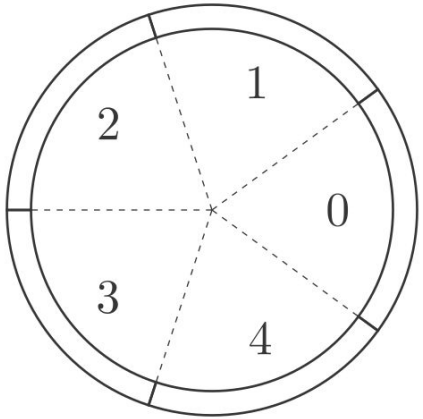
1 . Introduction to Fully Homomorphic Encryption

2. Introduction to the TFHE cryptosystem

3. Our contributions : a framework for fast evaluation of Boolean functions

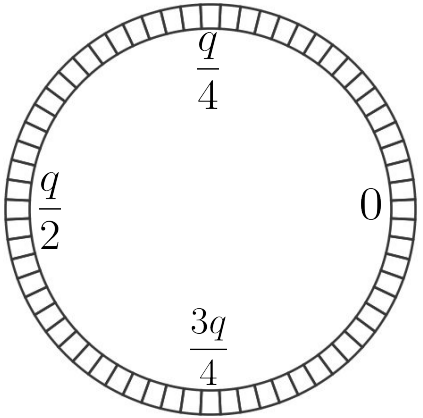
TFHE : description of the scheme

Clear space: $\mathbb{T}_p$



p has a size of few bits.

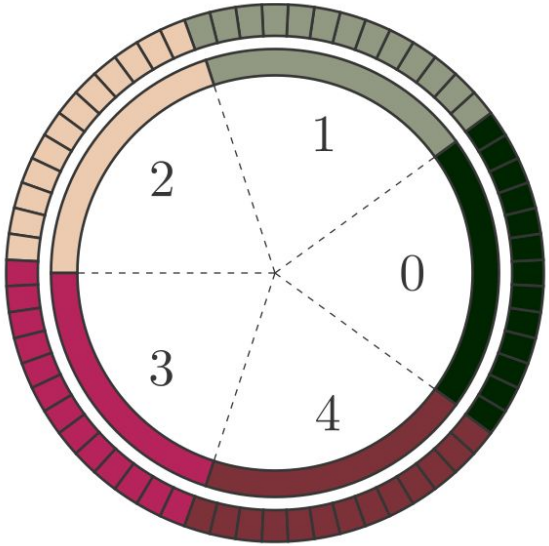
Encrypted space: $\mathbb{T}_q$



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

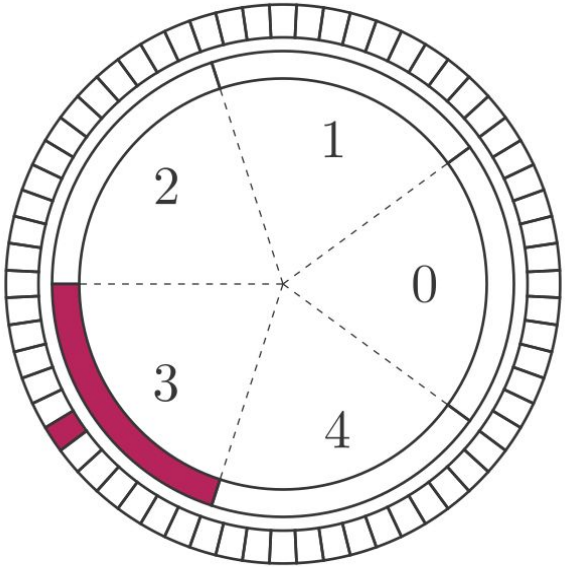
TFHE : description of the scheme

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



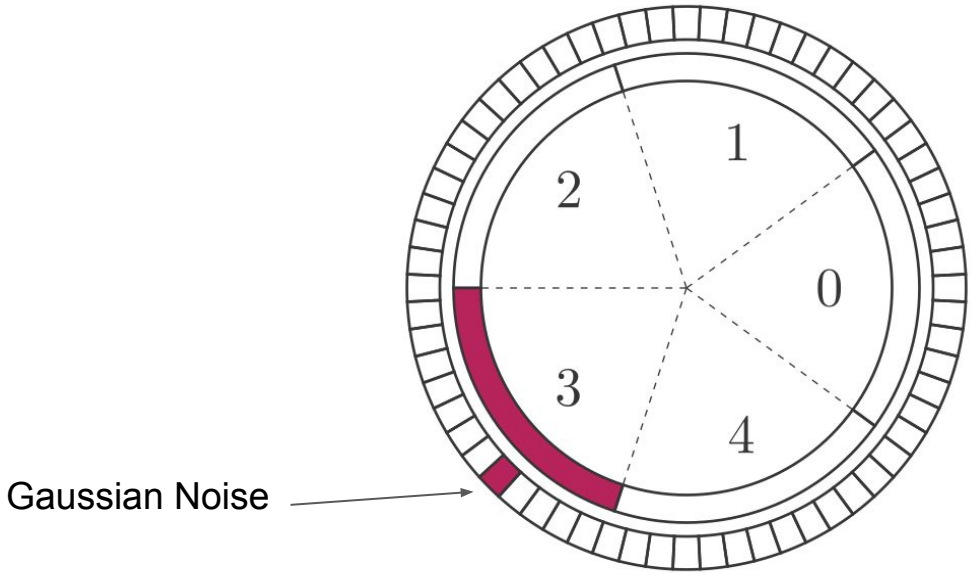
TFHE : description of the scheme

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 :

$$a = \underbrace{\left(\bigcirc , \bigcirc , \cdots , \bigcirc \right)}_{n} \xleftarrow{\$} \mathbb{T}_q^n$$

TFHE : description of the scheme

Sampling of a mask :

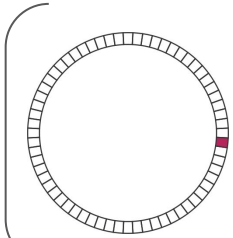
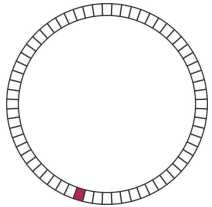
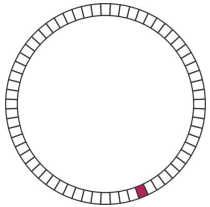
$$a = \underbrace{\left(\bigcirc , \bigcirc , \cdots , \bigcirc \right)}_{n} \xleftarrow{\$} \mathbb{T}_q^n$$

Secret key:

$$s_k = (1, 0, \ldots, 1) \xleftarrow{\$} \mathbb{B}^n$$

TFHE : description of the scheme

Construction of ciphertexts :

$$c = \left(\text{  }, \text{  }, \ldots, \text{  }, \right)$$

TFHE : description of the scheme

Construction of ciphertexts :

$$c = \left(\bigcirc , \bigcirc , \cdots , \bigcirc , \bigcirc \right)$$

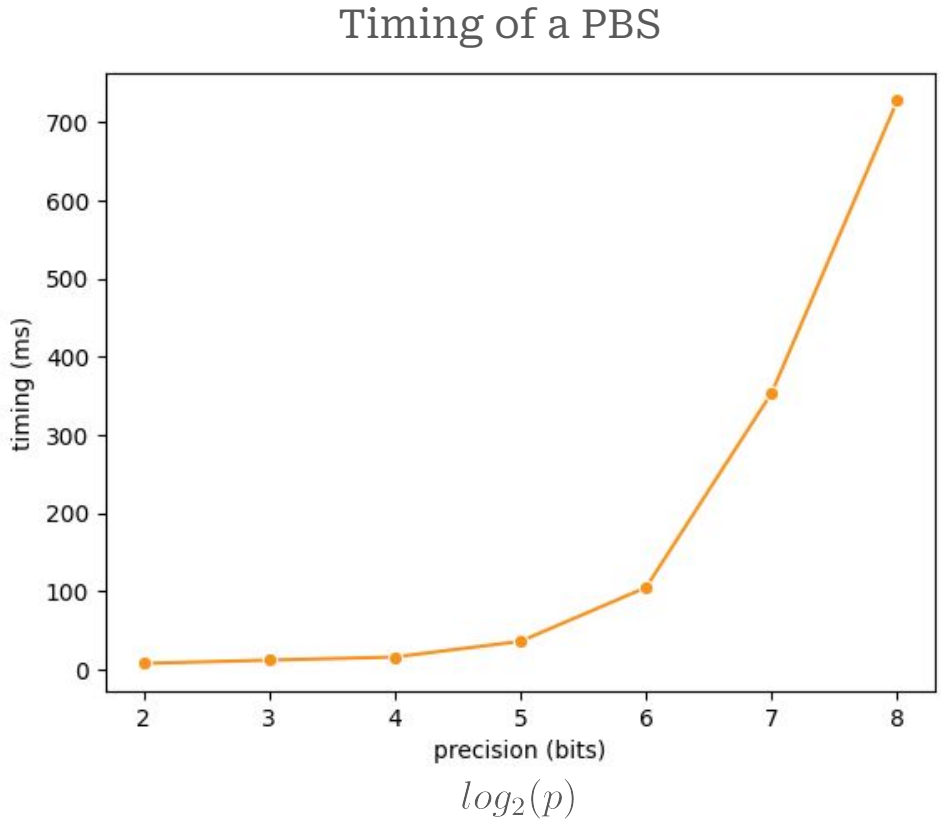
$$b = \langle a, s \rangle + m + e$$

TFHE: available operations

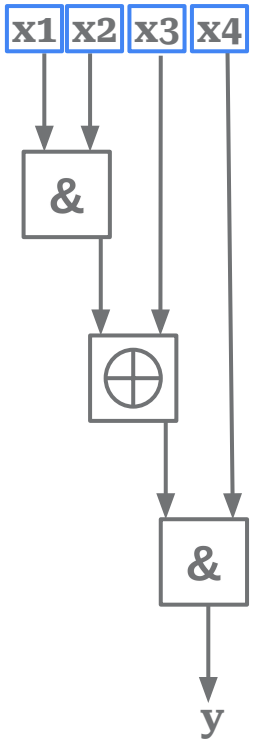
- Sum on $\mathbb{T}_p$
- External product on $\mathbb{T}_p$ by a clear constant

- Programmable Bootstrapping
  - Resets the noise level 
  - Evaluates any Look-up table on the ciphertext 
  - **BUT** very slow operation 

TFHE only allows small precisions

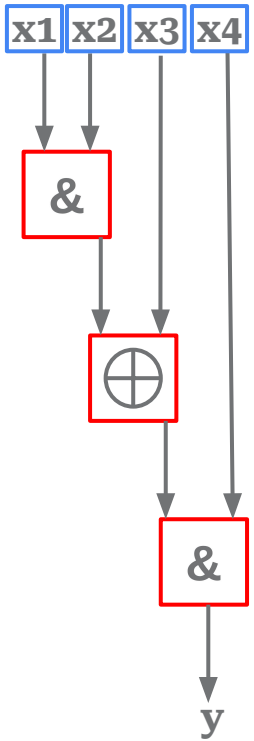


Natural approach of Boolean function evaluation: gate bootstrapping



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

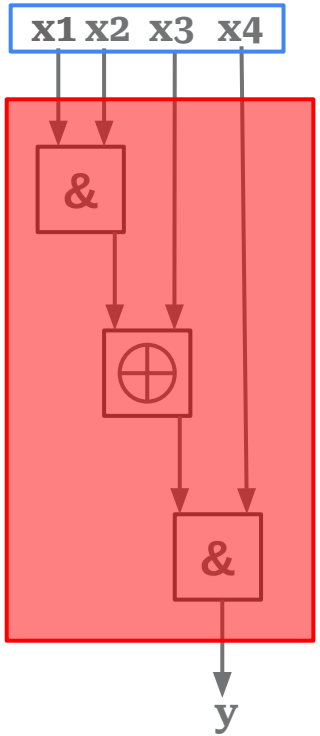
Natural approach of Boolean function evaluation: gate bootstrapping



- 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

Other approach of Boolean function evaluation: large LUT



- See Boolean Function as a LUT on ℓ bits
- Pack all bits in a single ciphertext of order p^ℓ
- Evaluate the function with a very large PBS.

Problem: the PBS becomes very slow and not practical.

Outline

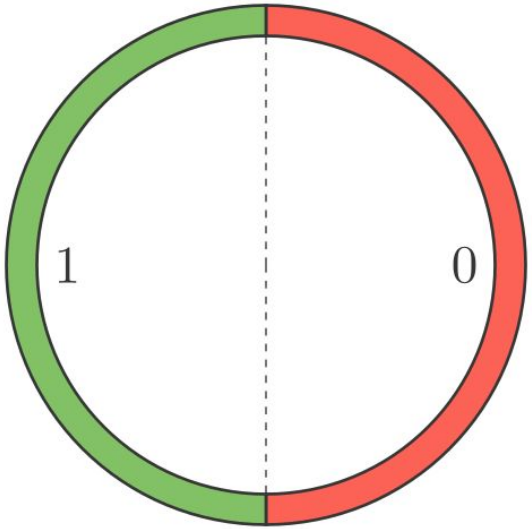
1 . Introduction to Fully Homomorphic Encryption

2. Introduction to the TFHE cryptosystem

3. Our contributions : a framework for fast evaluation of Boolean functions

How to represents bit in TFHE ?

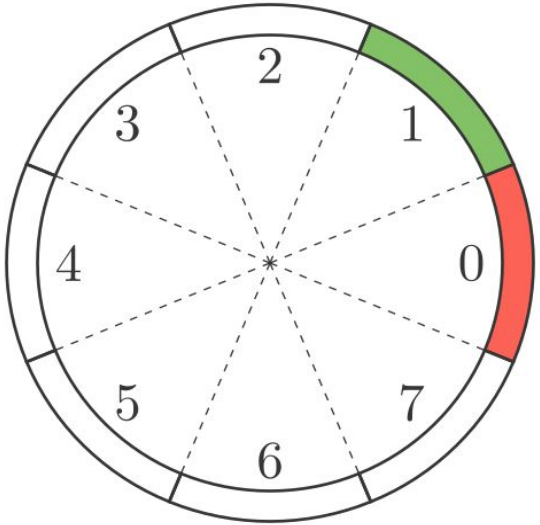
Obvious (and terrible) solution:



- No 2-input LUT
- Only XOR can be evaluated

How to represents bit in TFHE ?

Slightly better solution of the literature:



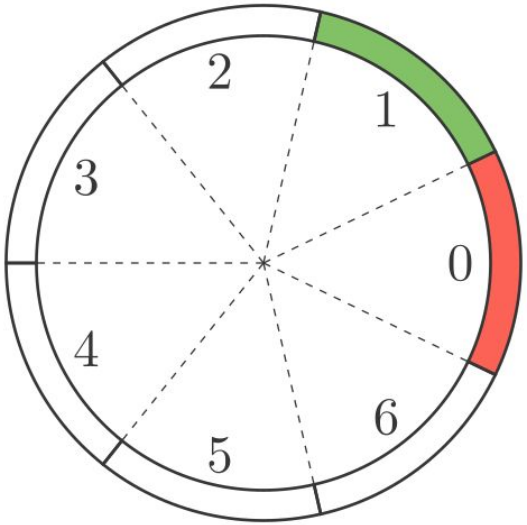
$$p = 2^k$$

Negacyclity problem causes one of the following:

-> Not every functions are evaluable
-> Sum is no longer homomorphic

How to represent bit in TFHE ?

Our approach:



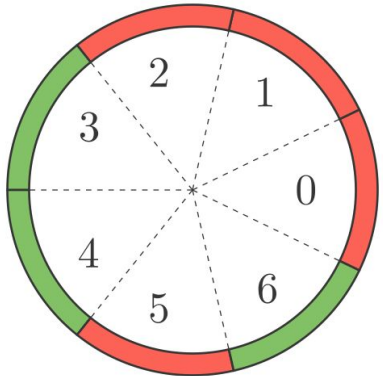
p prime

Negacyclicity problem vanishes!

Enter the p-encodings

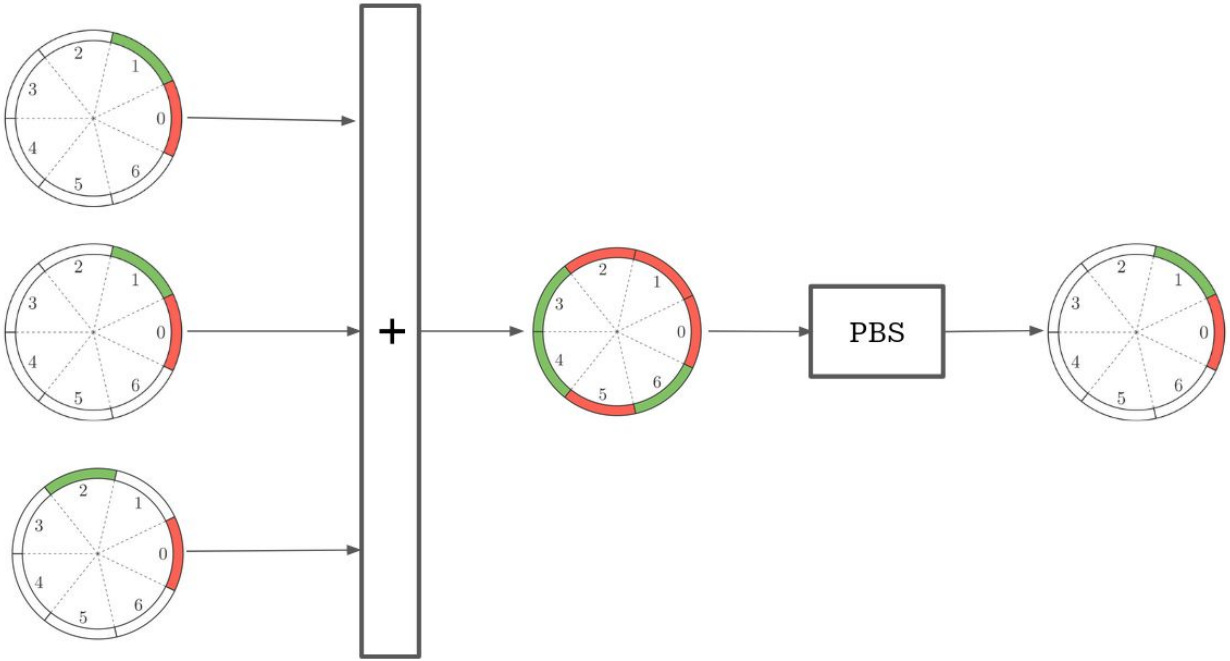
A p-encoding is a function $\mathcal{E} : \mathbb{B} \mapsto 2^{\mathbb{Z}_p}$

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



New function evaluation algorithn

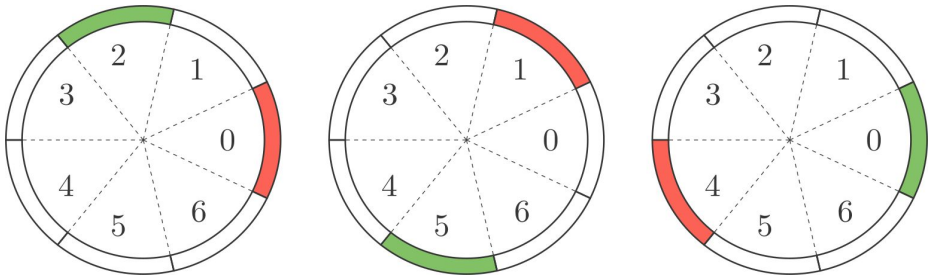
Function evaluation is made by a sum followed by a PBS



Toy Example

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.

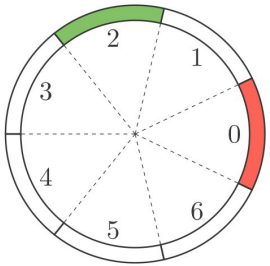


Toy Example

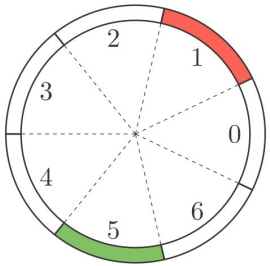
Truth table of f:

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

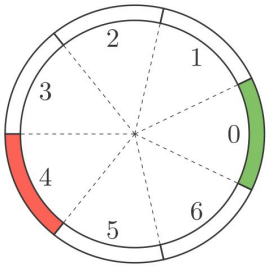
$\mathcal{E}_1$



$\mathcal{E}_2$



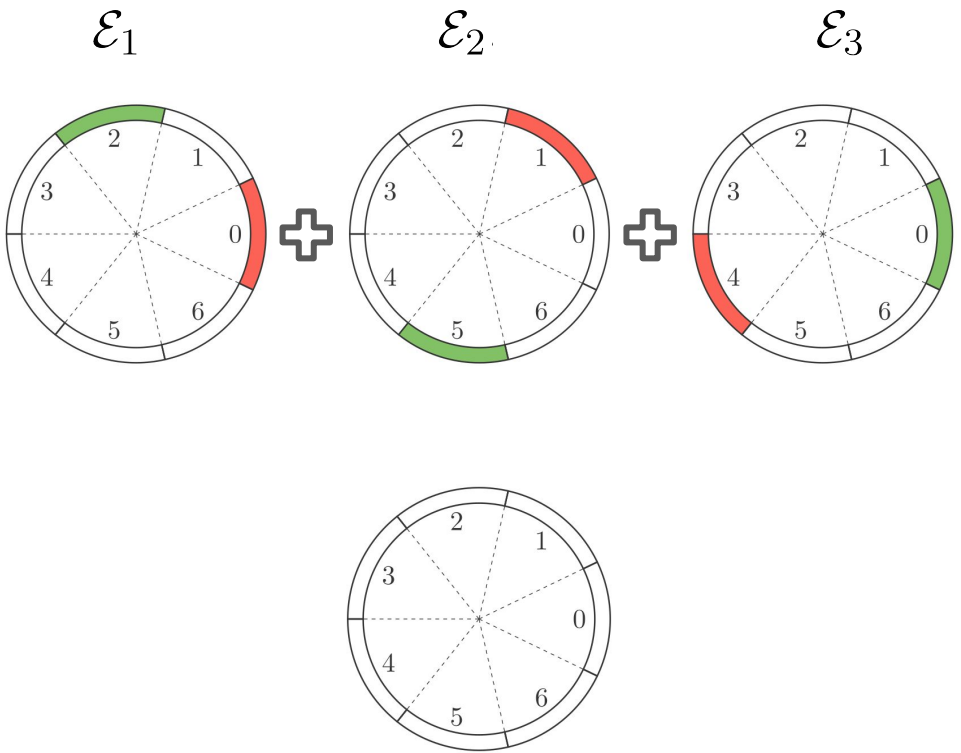
$\mathcal{E}_3$



Toy Example

Truth table of f:

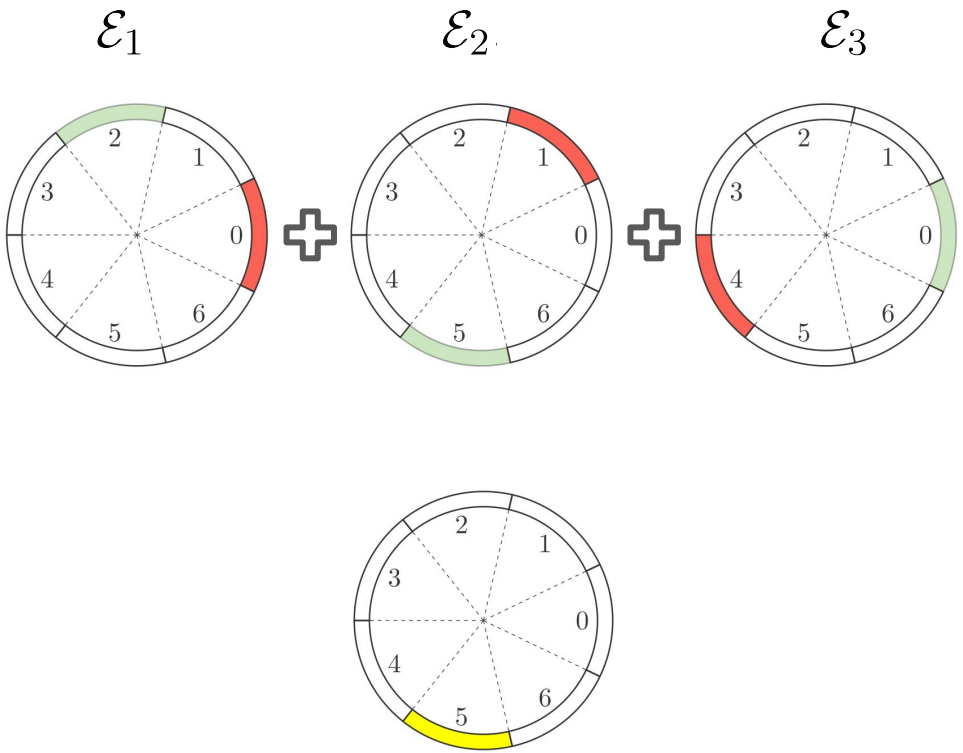
b1	b2	b3	f(b1, b2, b3)
0	0	0	
1	0	0	
0	1	0	



Toy Example

Truth table of f:

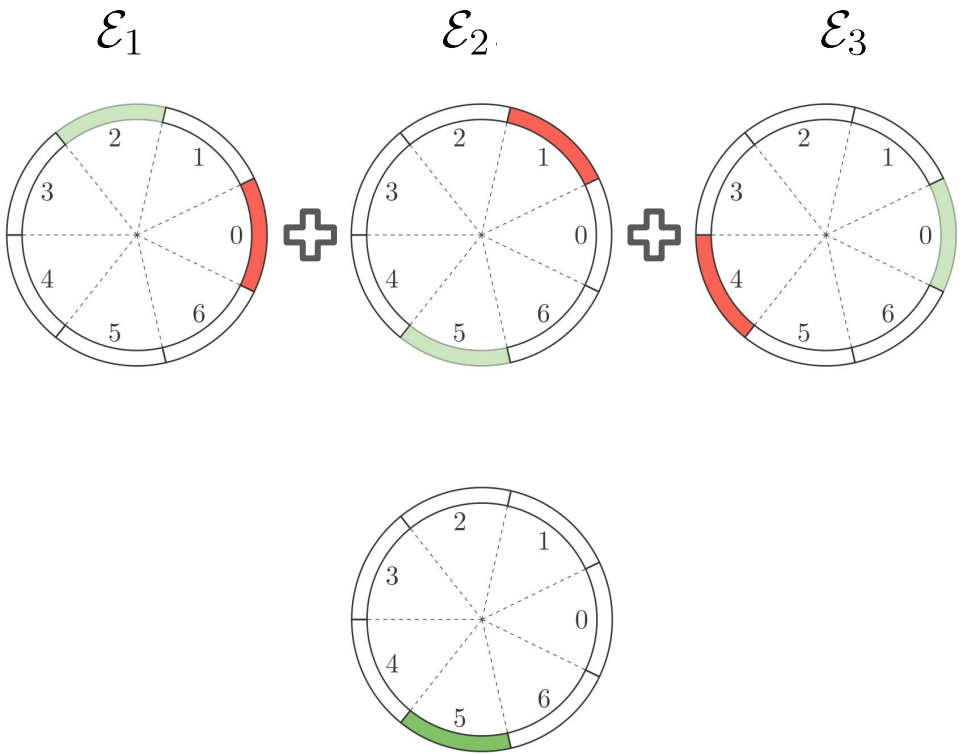
b1	b2	b3	f(b1, b2, b3)
0	0	0	
1	0	0	
0	1	0	



Toy Example

Truth table of f:

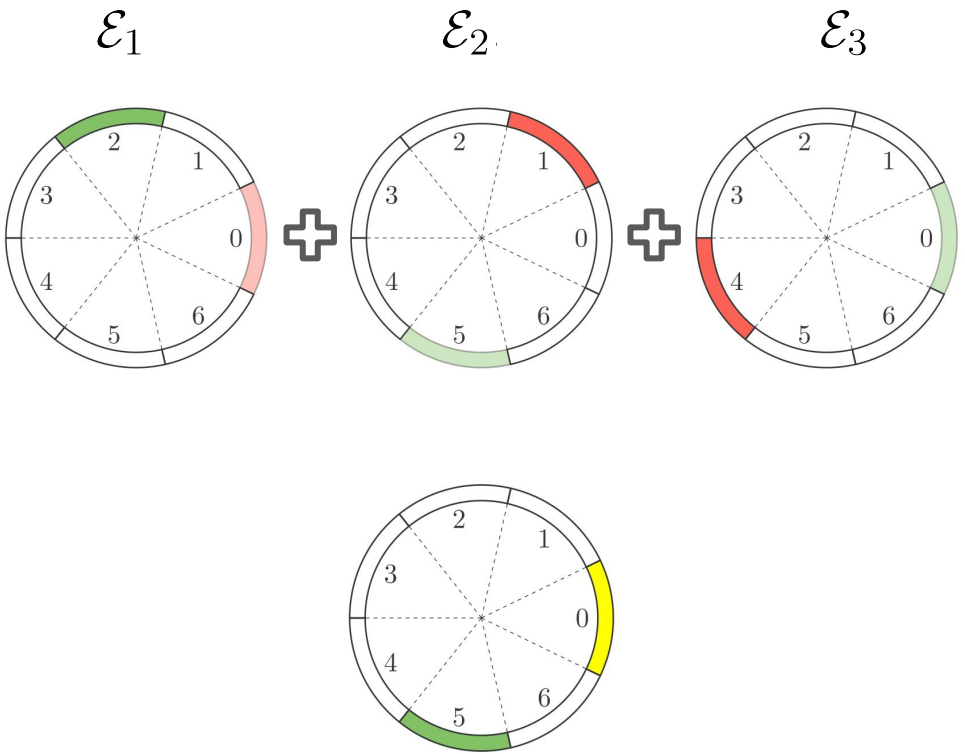
b1	b2	b3	f(b1, b2, b3)
0	0	0	1
1	0	0	
0	1	0	



Toy Example

Truth table of f:

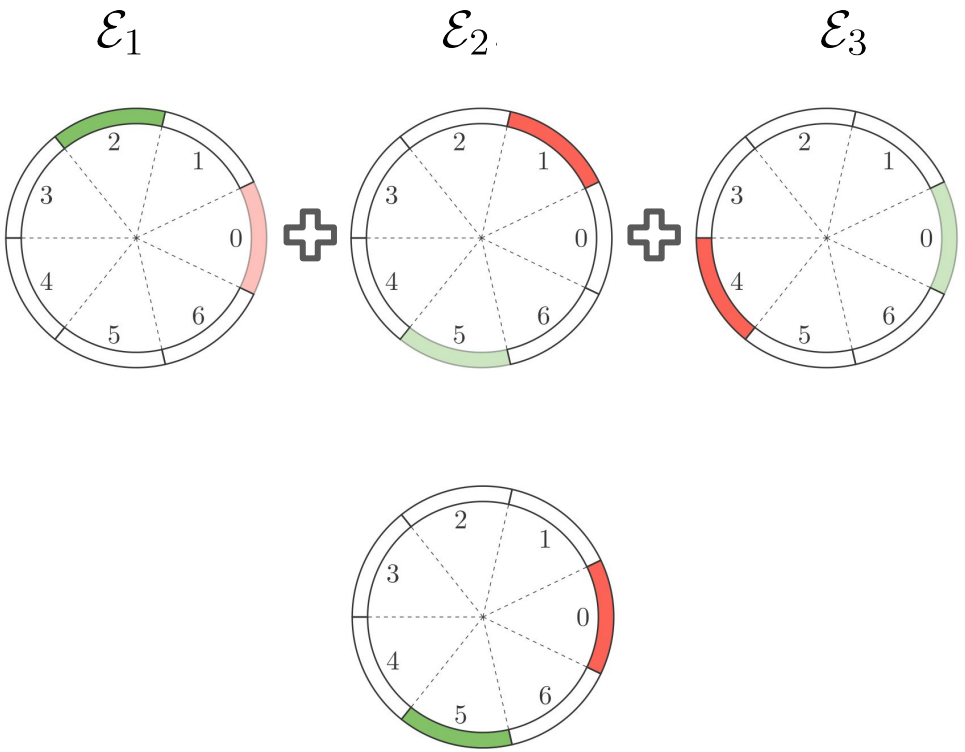
b1	b2	b3	f(b1, b2, b3)
0	0	0	1
1	0	0	
0	1	0	



Toy Example

Truth table of f:

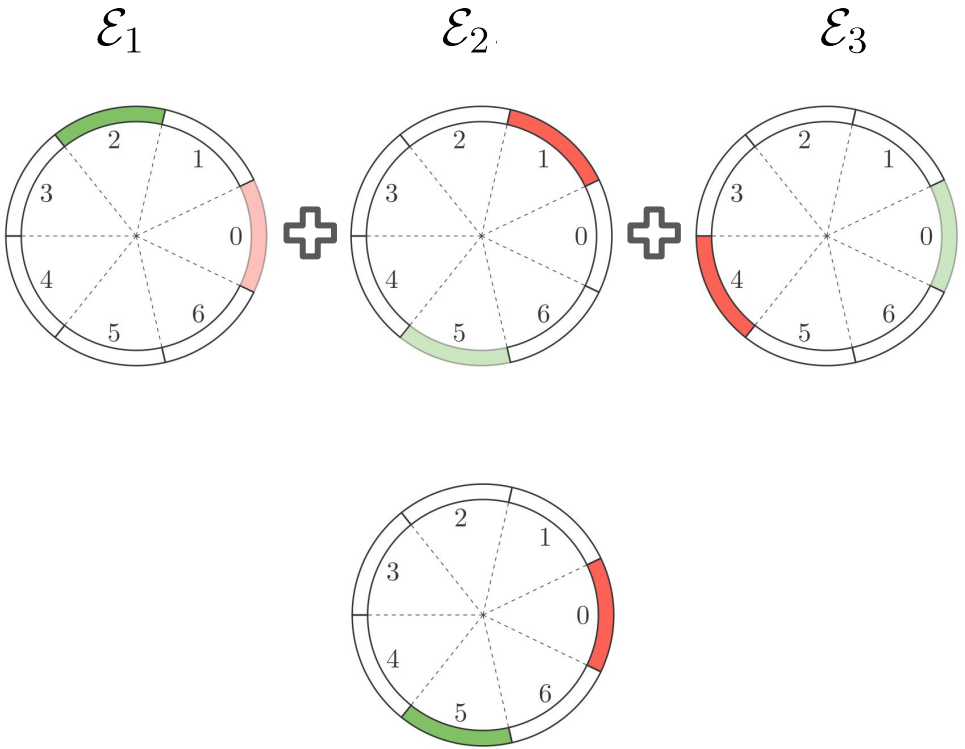
b1	b2	b3	f(b1, b2, b3)
0	0	0	1
1	0	0	0
0	1	0	



Toy Example

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.

Advantages of the method

- One single bootstrapping to evaluate the whole function
- No need to extend the plaintext space to fit more inputs
- Scales much better that the two conventional approaches (gate bootstrapping / packing everything in a big LUT)

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:

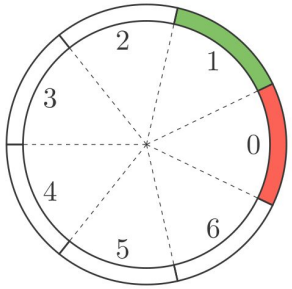
$$\mathcal{E}_i = \begin{cases} 0 \mapsto \{0\} \\ 1 \mapsto \{d_i\} \end{cases} \qquad \text{with: } \mathcal{E}_1 = \begin{cases} 0 \mapsto \{0\} \\ 1 \mapsto \{1\} \end{cases}$$

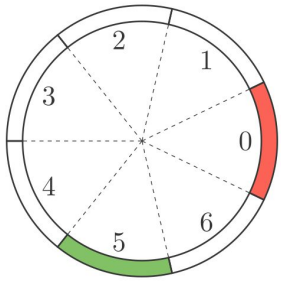
with no loss of generality

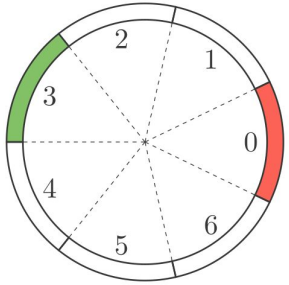
Another point of view on the problem

Truth table of f:

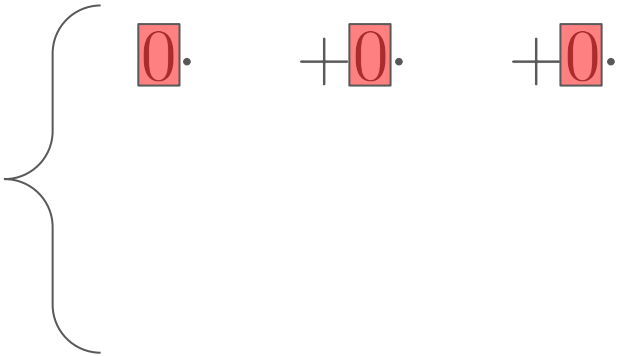
b1	b2	b3	f(b1, b2, b3)
0	0	0	
1	0	0	
0	1	0	





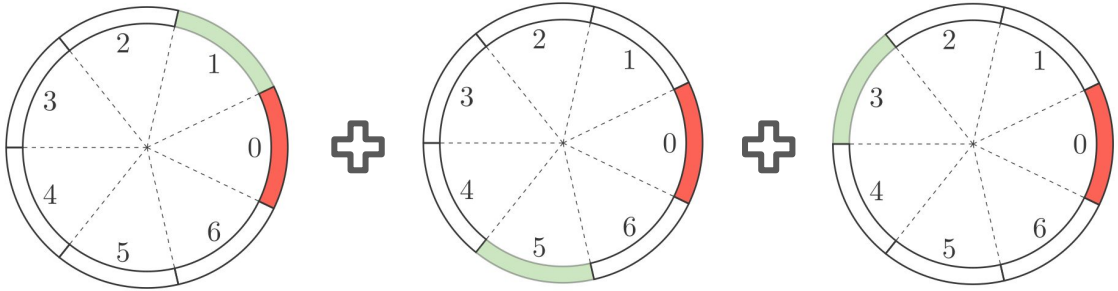


Another point of view on the problem



Truth table of f:

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

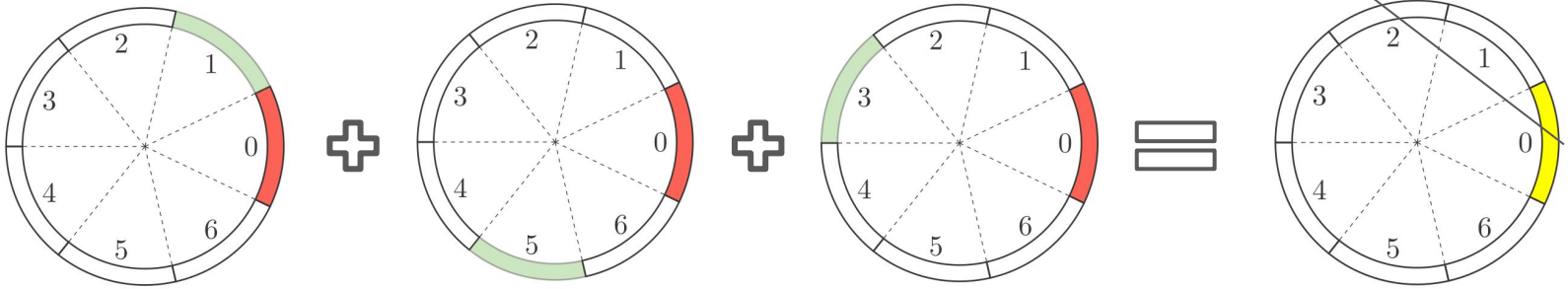


Another point of view on the problem

$$\begin{cases} 0\cdot \quad +0\cdot \quad +0\cdot \quad = r_0 \end{cases}$$

Truth table of f:

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

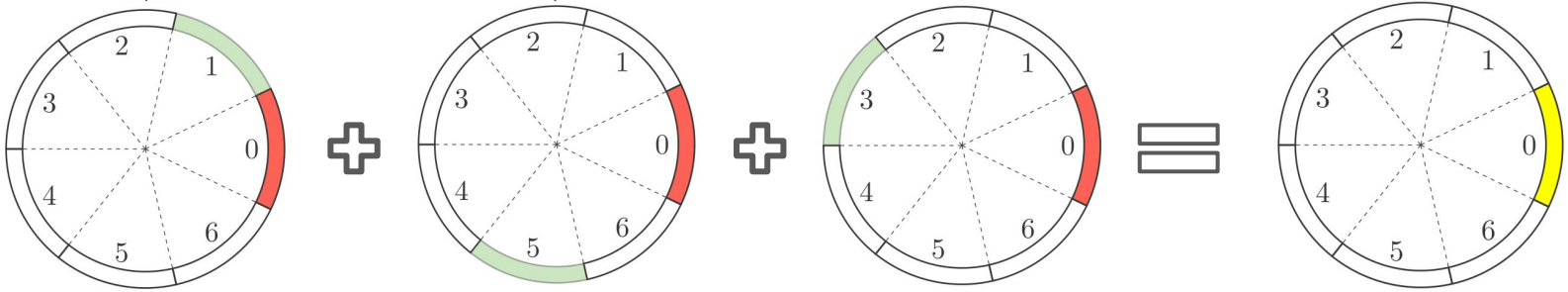


Another point of view on the problem

$$0 \cdot d_1 + 0 \cdot d_2 + 0 \cdot d_3 = r_0$$

Truth table of f:

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

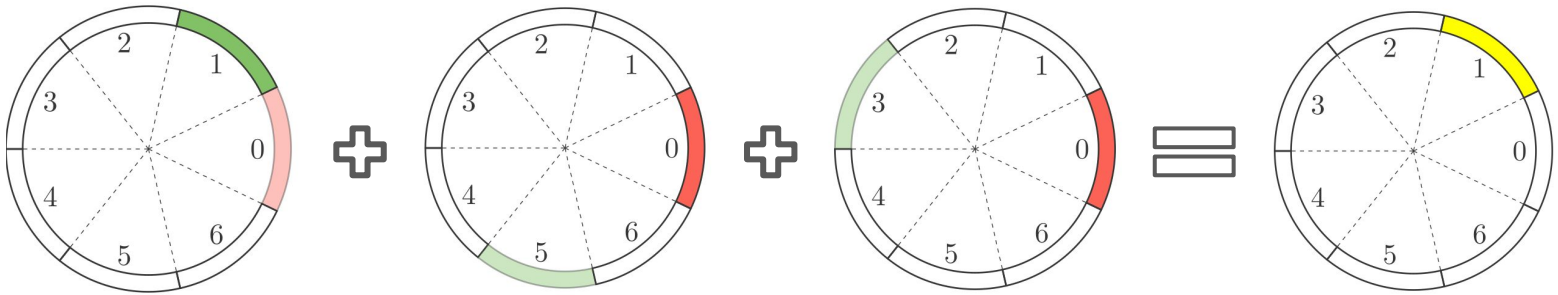


Another point of view on the problem

$$\begin{cases} 0 \cdot d_1 + 0 \cdot d_2 + 0 \cdot d_3 = r_0 \\ 1 \cdot d_1 + 0 \cdot d_2 + 0 \cdot d_3 = r_1 \end{cases}$$

Truth table of f:

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



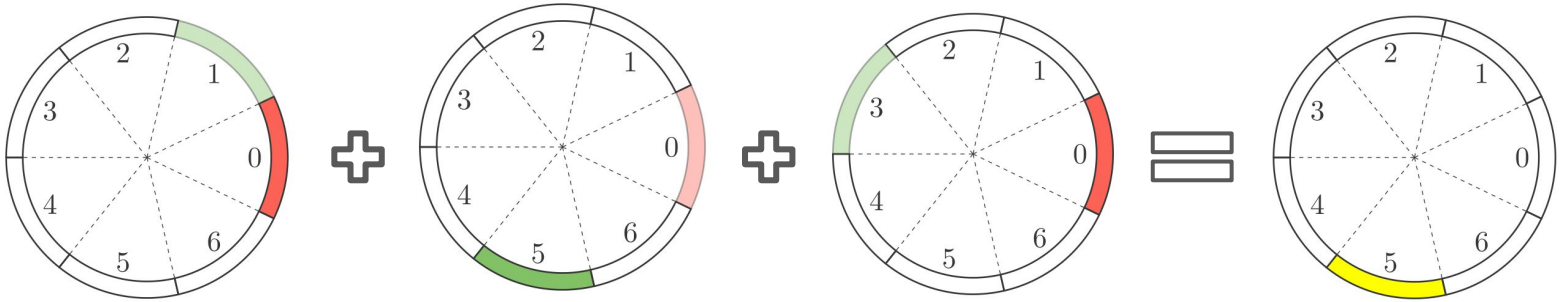
Another point of view on the problem

$$\begin{cases} 0\cdot d_1 + 0\cdot d_2 + 0\cdot d_3 = r_0 \\ 1\cdot d_1 + 0\cdot d_2 + 0\cdot d_3 = r_1 \\ 0\cdot d_1 + 1\cdot d_2 + 1\cdot d_3 = r_2 \\ \dots \end{cases}$$

Truth table of f:

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

...



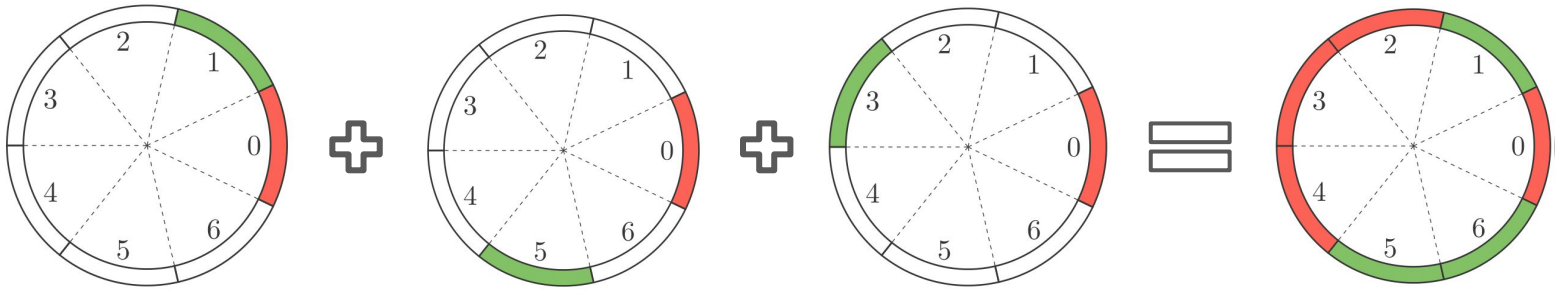
Another point of view on the problem

$$\begin{cases} 0\cdot d_1 + 0\cdot d_2 + 0\cdot d_3 = r_0 \bmod p \\ 1\cdot d_1 + 0\cdot d_2 + 0\cdot d_3 = r_1 \bmod p \\ 0\cdot d_1 + 1\cdot d_2 + 1\cdot d_3 = r_2 \bmod p \\ \dots \end{cases}$$

Truth table of f:

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

...

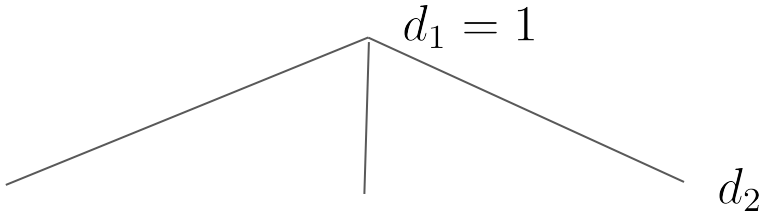


An other point of view on the problem

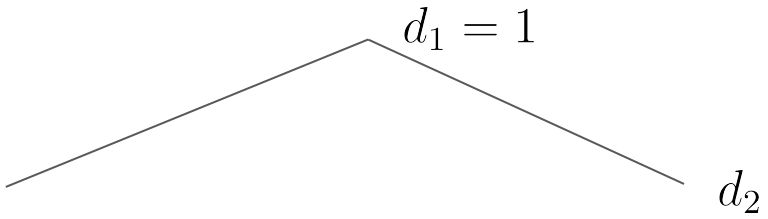
By writing all the inequations  we get:

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

The search algorithm

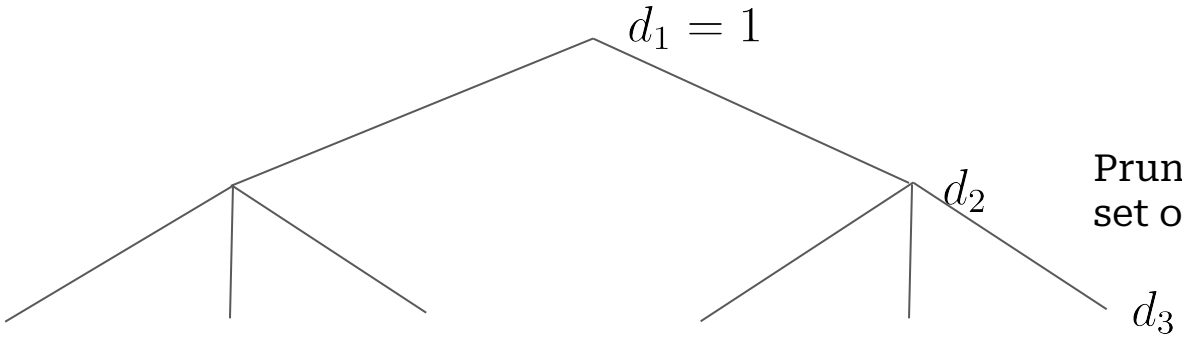


The search algorithm



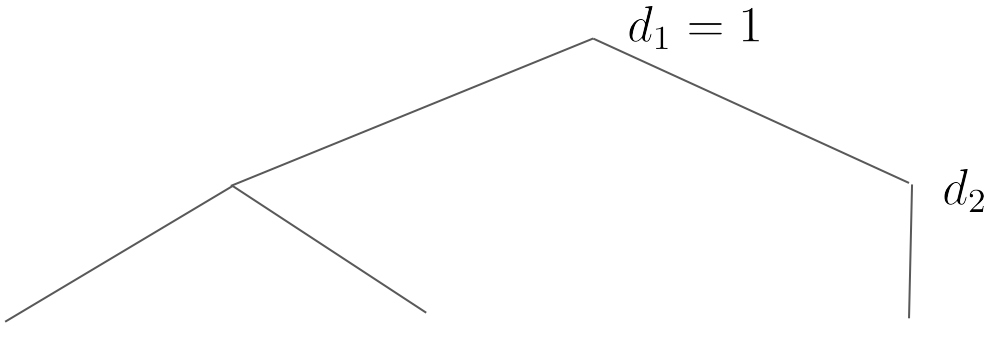
Pruning using the set of constraints

The search algorithm



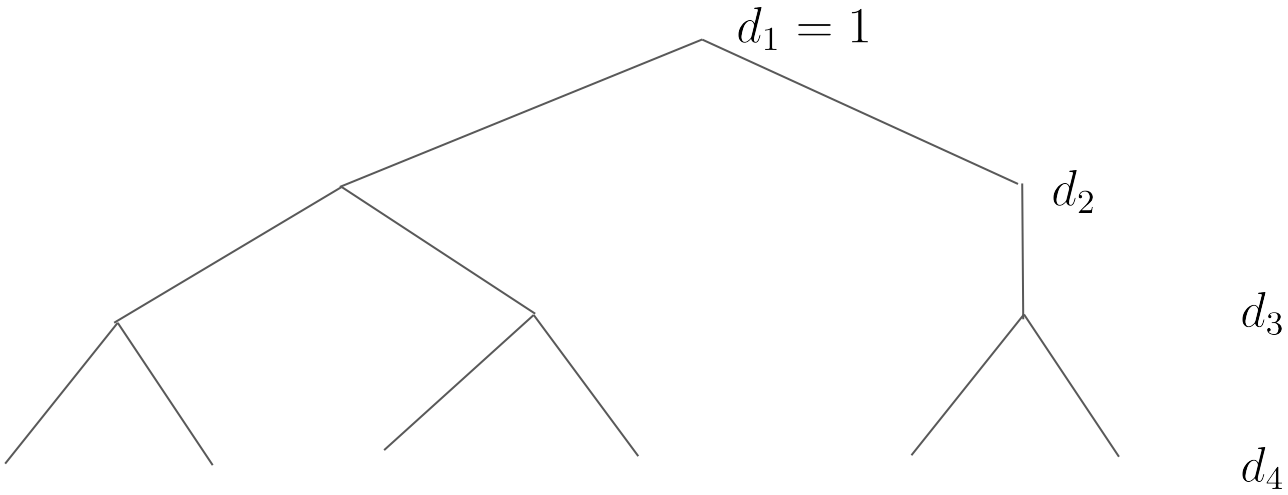
Pruning using the set of constraints

The search algorithm



Pruning using the set of constraints

The search algorithm



... until we find a path of length ℓ

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

Experimental results

Primitive	Section or Other work	Performances
One full run of SIMON	Gate Bootstrapping	174 s
	[BSS+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

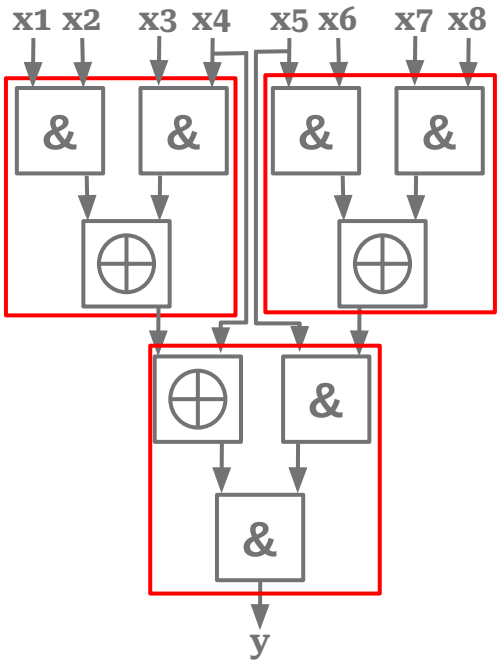
Use-cases includes transciphering, OPRF, ...

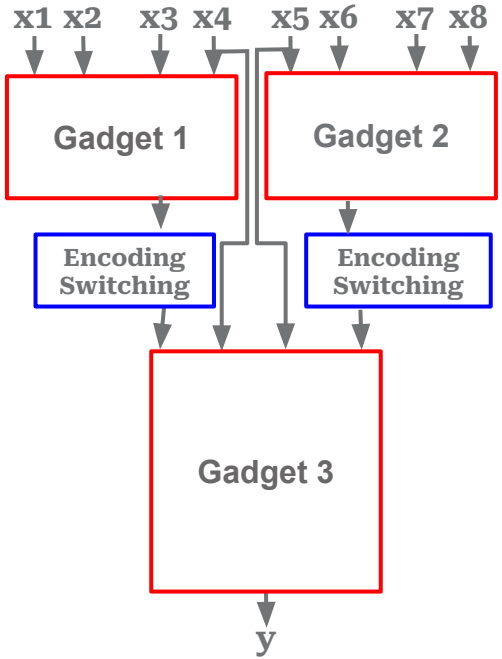
Experimental results

Primitive	Section or Other work	Performances
One full run of SIMON $p = 9$	Gate Bootstrapping	174 s
	[BSS+23] †	128 s
	➡ Our work (Section 7.1)	10 s
One warm-up phase of Trivium (*) $p = 9$	Gate Bootstrapping	1498 s
	[BOS23] (estimation on our machine)	53 s
	➡ Our work (Section 7.2)	32.8 s
One Full Keccak permutation (∗) $p = 3$	Gate Bootstrapping	30.7 min
	➡ Our work (Section 7.3)	8.8 min
One Ascon hashing (∗) $p = 17$	Gate Bootstrapping	200s
	➡ Our work (Section 7.4)	92 s

Use-cases includes transciphering, OPRF, ...

Extension to bigger circuits





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

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

Thank you !

https://eprint.iacr.org/2023/1589

For more details

Slides by Nicolas Bon

Mathematical figures built with Manim
(https://github.com/ManimCommunity/manim)

Icons from Flaticons