CRYPTOEXPERTS

WE INNOVATE TO SECURE YOUR BUSINESS

1794

ENS

ÉCOLE NORMALE SUPÉRIEURE

Optimized homomorphic evaluation of Boolean functions

Journées C2 2023

Nicolas Bon

What is FHE ?

Client

Server

$sk, evk \leftarrow \texttt{KeyGen()}$

evk

$c \leftarrow \texttt{EncFHE}(m, sk)$

c

$c' \leftarrow f_{FHE}(c, evk)$

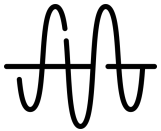
c'

$m' \leftarrow \texttt{DecFHE}(c, sk)$

Correctness property: $f(m) = m'$

Main constraints of FHE

Performances: Overhead in time and in size

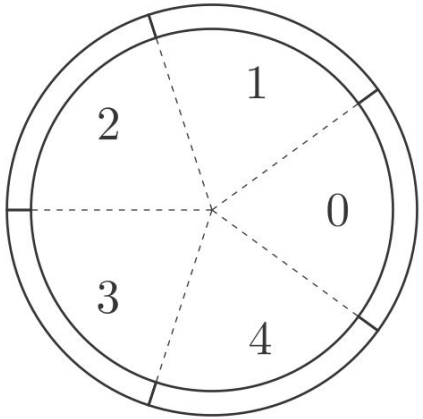


Noise control: risk of losing correctness

Limited set of supported homomorphic operations

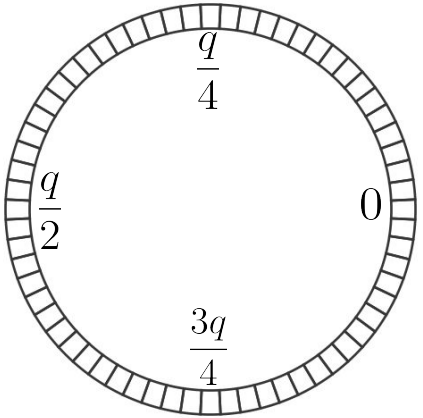
TFHE : description of the scheme

Clear space: $\mathbb{T}_p$



p has a size of few bits.

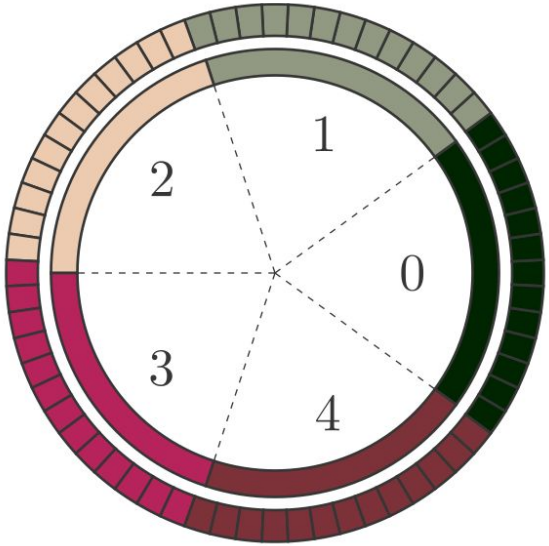
Encrypted space: $\mathbb{T}_q$



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

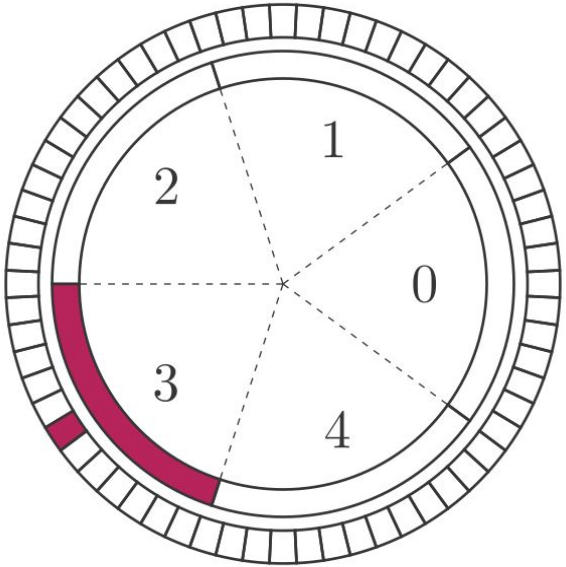
TFHE : description of the scheme

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



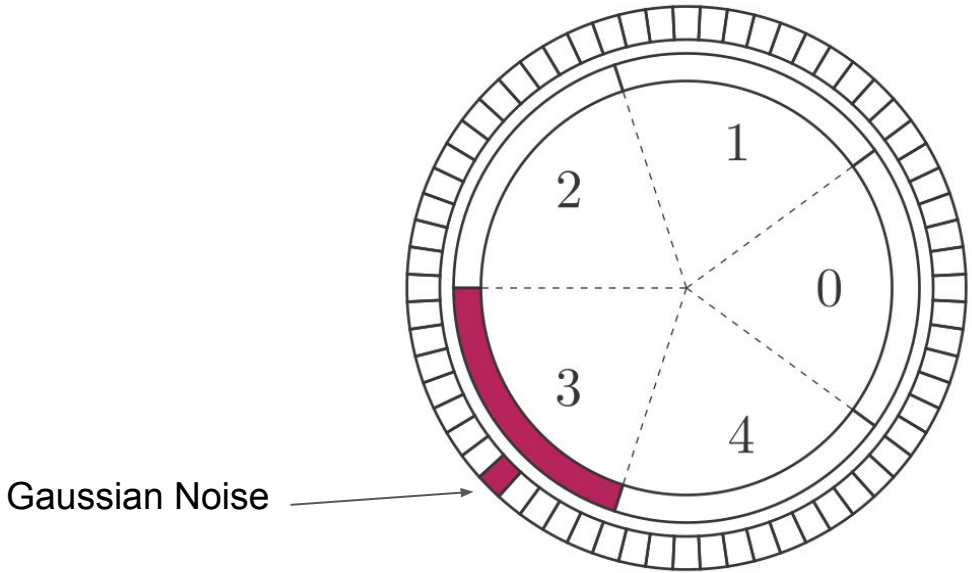
TFHE : description of the scheme

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



TFHE : description of the scheme

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

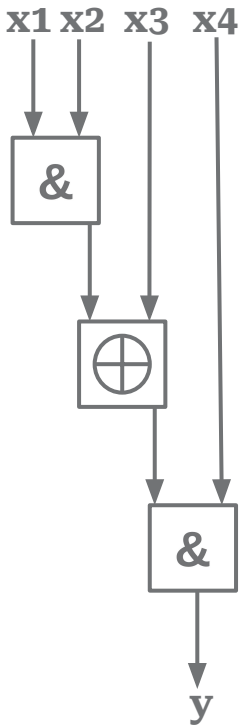


TFHE: available operations

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

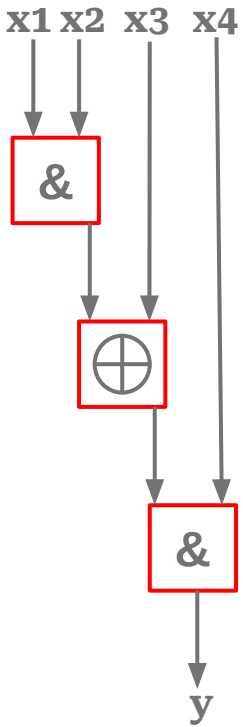
- Programmable Bootstrapping
  - Reset the noise level 
  - Evaluate any Look-up table on the ciphertext 
  - **BUT** slow and heavy operation 

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

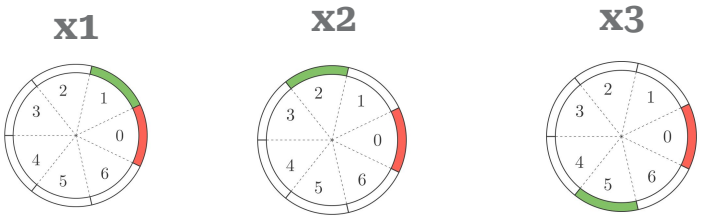


- See Boolean functions as Boolean circuits
- Each bit is a ciphertext
- Each gate is a 2-input Look-up table

Problem: each gate costs 1 Programmable Bootstrapping

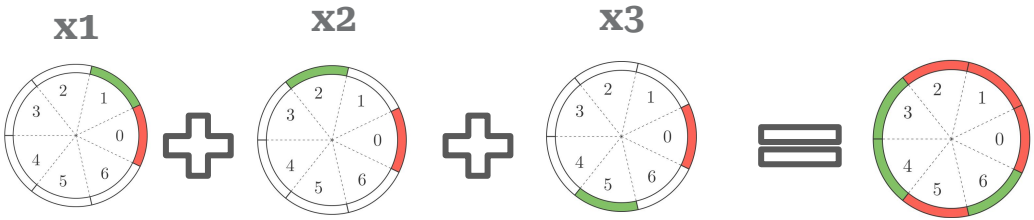
Our strategy

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



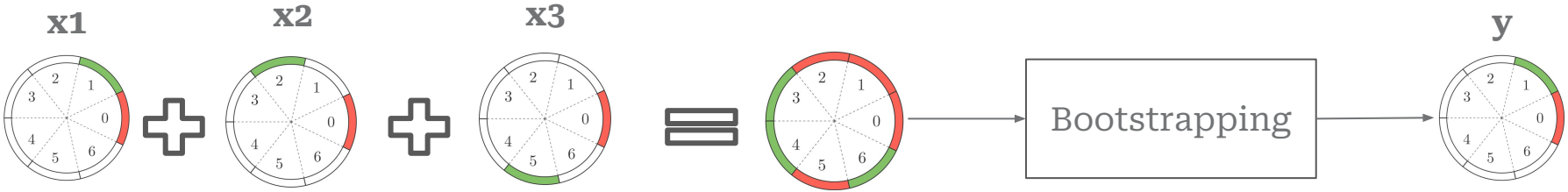
Our strategy

- 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



Our strategy

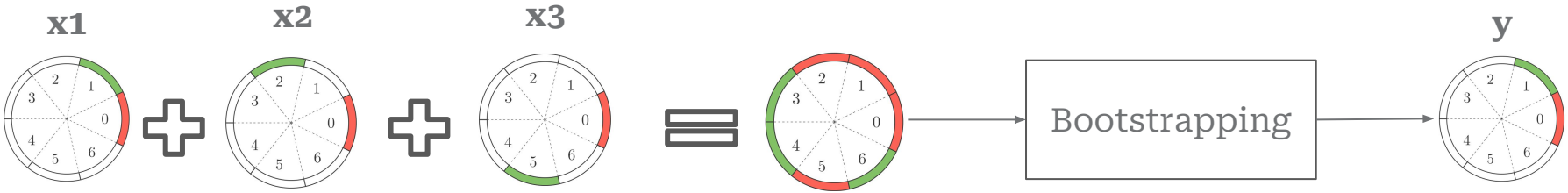
- Pick a p(better if prime) and embed each bit in $\mathbb{T}_p$
- Compute the sum (fast !) and label the sectors according to the function we want to evaluate
- Compute a Bootstrapping on the sum and get a fresh ciphertext



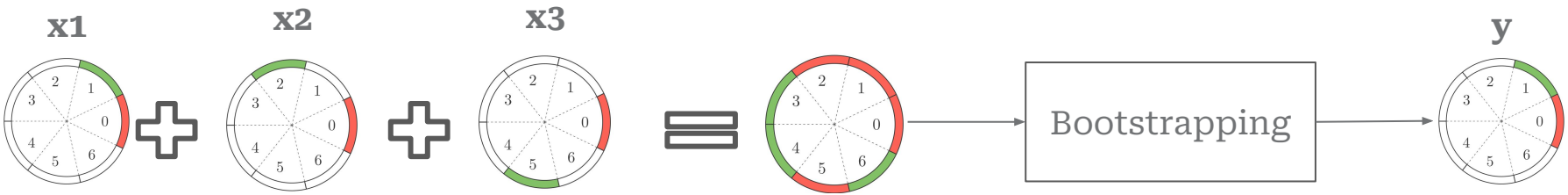
Our strategy

- 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



Construction of our solution



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)

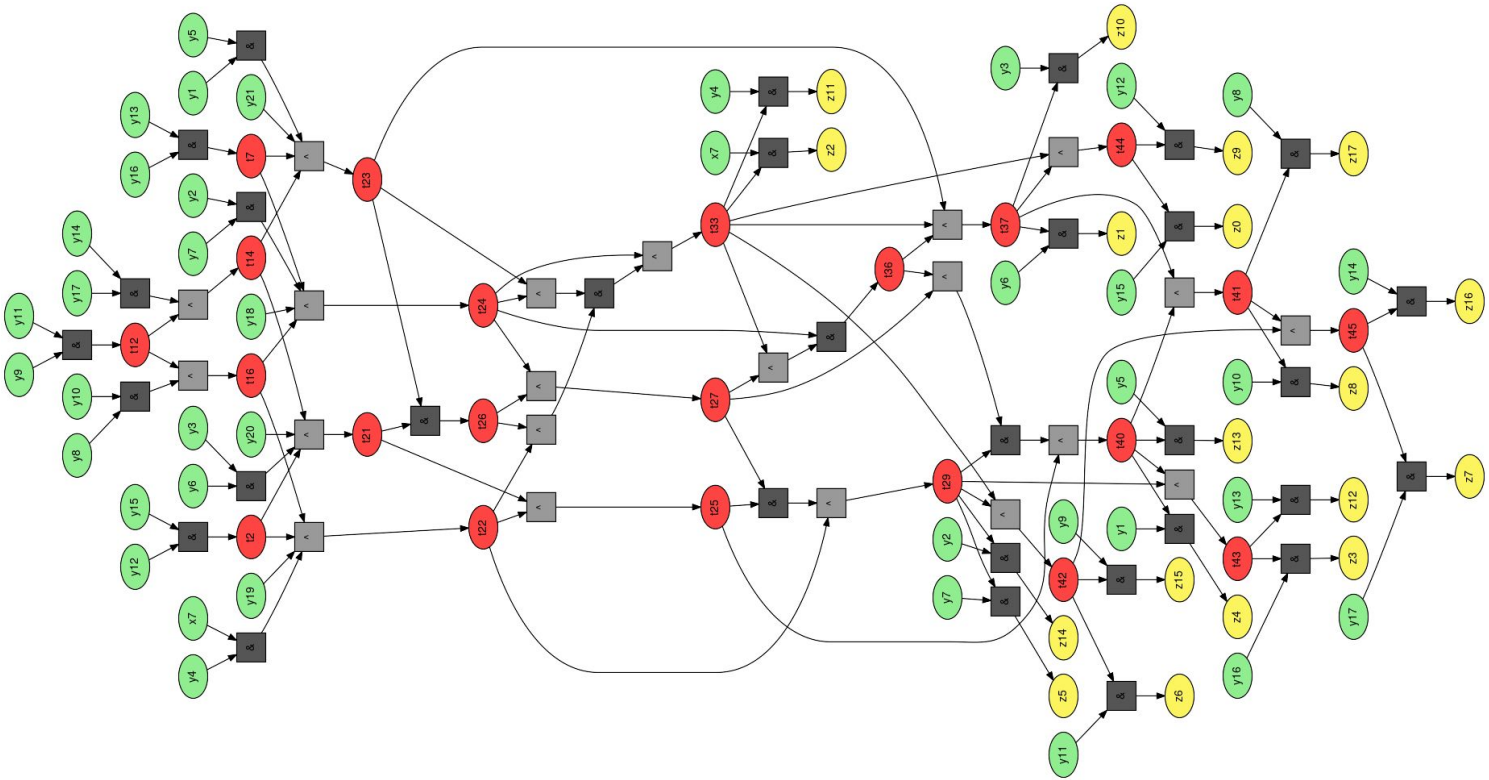
Our search algorithm finds the optimal solution to this problem

Application to cryptographic primitives

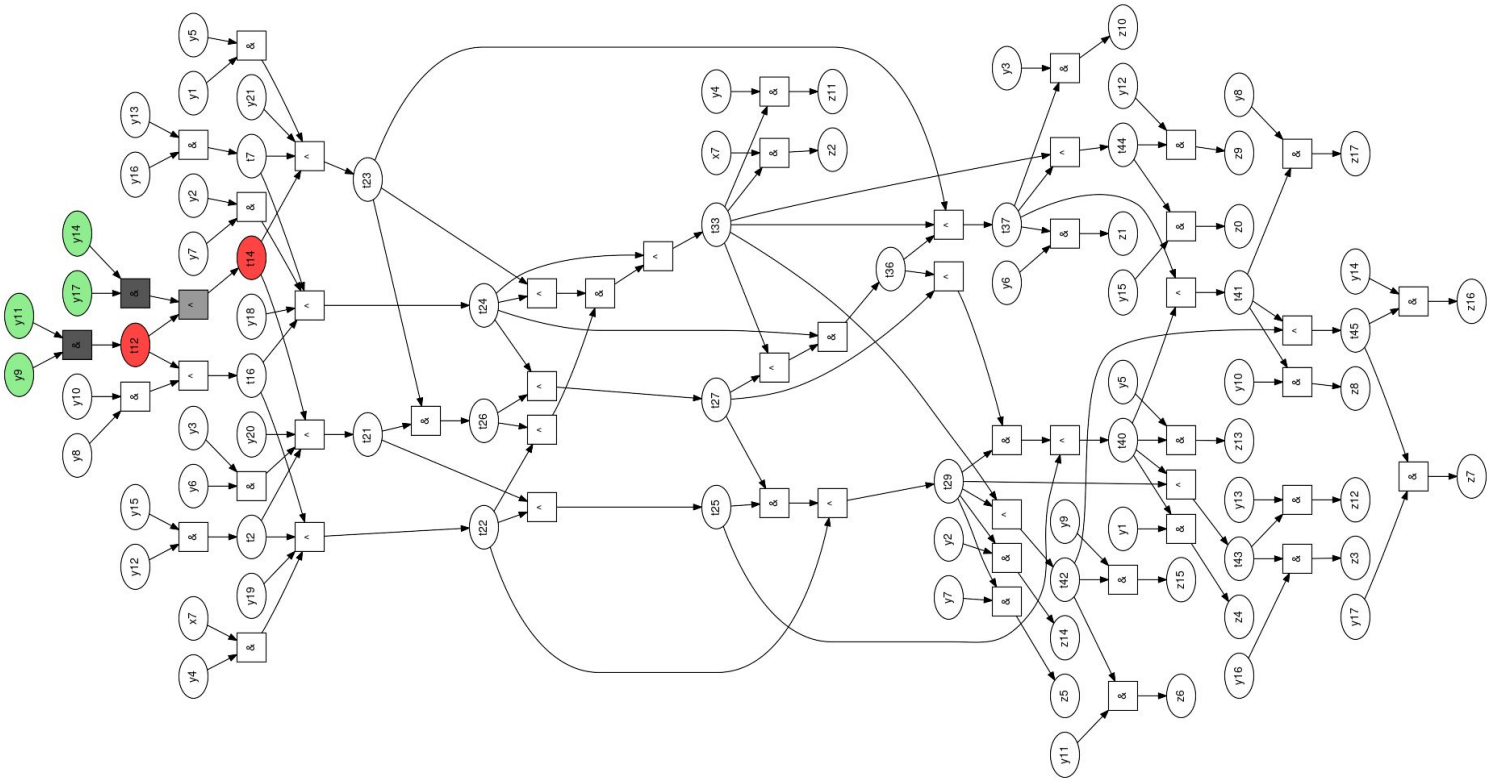
- For use-cases such as transciphering, OPRF, ...

- Efficient solutions for acceptable modulus for some lightweight block ciphers and hash functions
- Our implementation beats the state of the art
- But no solution for AES !

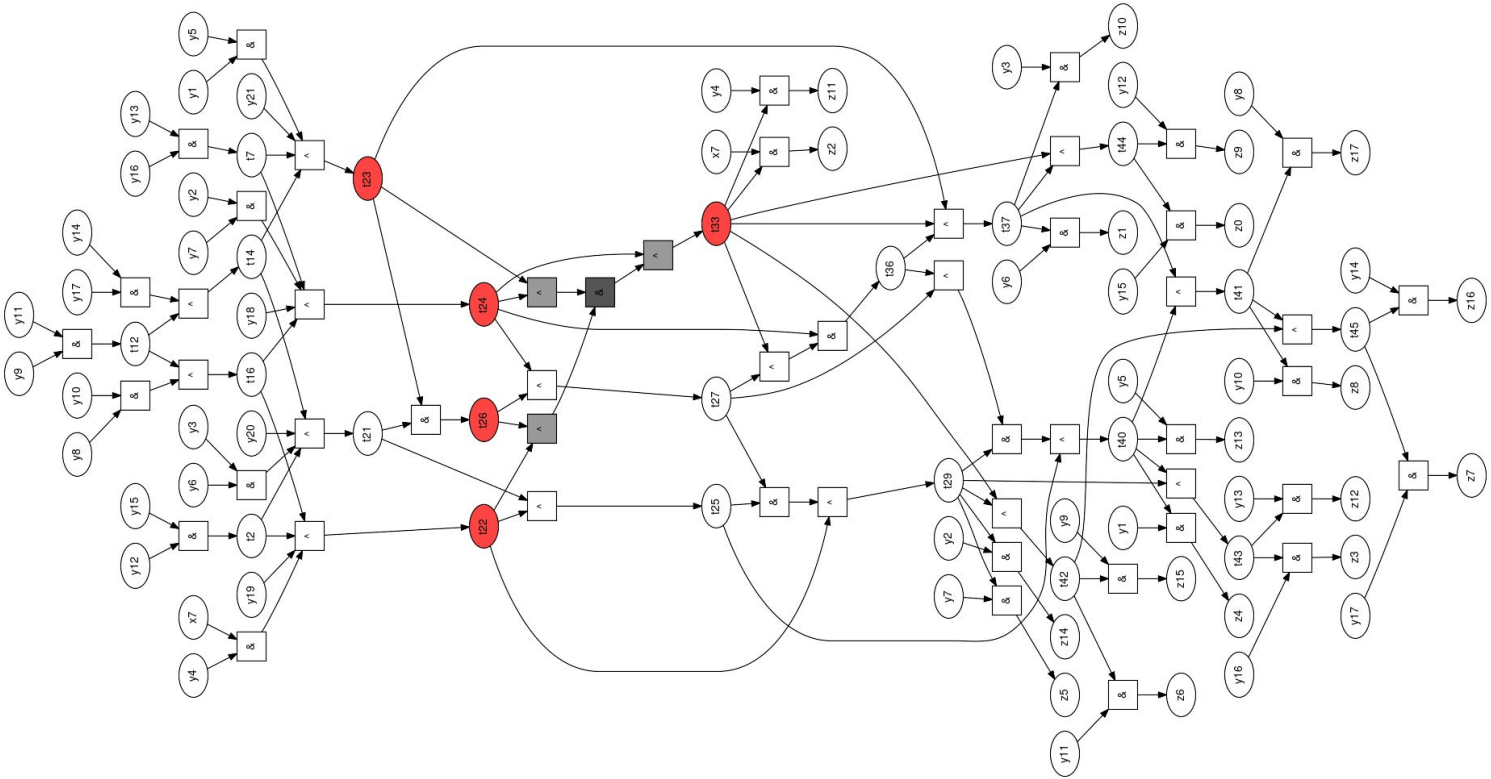
Extension to bigger circuits (e.g. AES)



Extension to AES



Extension to AES



AES: performances

- 210 seconds on one thread on a laptop (beats state of the art). Highly parallelizable
- Total of 7040 Bootstrappings (with p=11).

Thank you !

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

For (much more) details