

Circuit optimization problems in the context of homomorphic encryption

Sergiu Carpov

Arcium

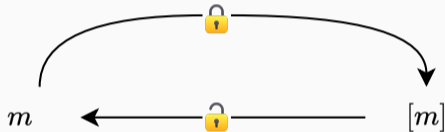
Privacy-preserving computations

- Anonymization (weak privacy guaranties)
- Differential privacy (somewhat better, limited functionalities)
- Secure enclaves (must trust the “hardware”)
- Multi-party computation (communication bound)
- *Homomorphic encryption* (computation bound)

What is homomorphic encryption?

Classical encryption

- Protect data confidentiality:
 - in storage / in transit
- Computationally difficult to decrypt without the secret
 - Or even impossible
- High performance:
 - e.g. AES-NI throughput $> 2\text{GB/s}$ on a single CPU core

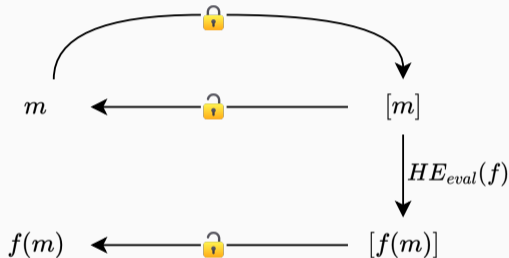


Homomorphic encryption (HE)

- Protect data confidentiality:
 - in storage / in transit / **in use**
- Impressive progress in the efficiency of homomorphic schemes
 - Hours to milliseconds per operation

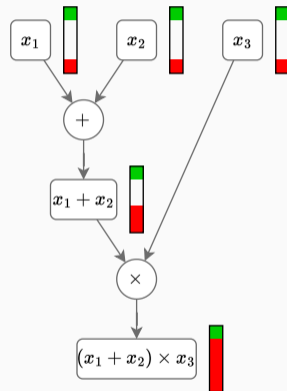
Plaintext operations

- Arithmetic over rings/fields
 - Additions, multiplications, ...
 - $\mathbb{Z}_p[X] \text{ mod } X^n + 1, F_p$



Inherent to each ciphertext

- Ensures scheme security
- **Noise** increases after each operation
 - Multiplication \gg addition
- **Message** will be lost if noise overlaps



Ring LWE scheme example

- Scheme security given by:
 - Polynomial ring size
 - Noise level
- Homomorphic operations budget:
 - Ciphertext coefficient size to noise ratio

LWE problem

- Find $\mathbf{s} \in \mathbb{Z}_q^n$ given polynomial many samples of

$$(\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e)$$

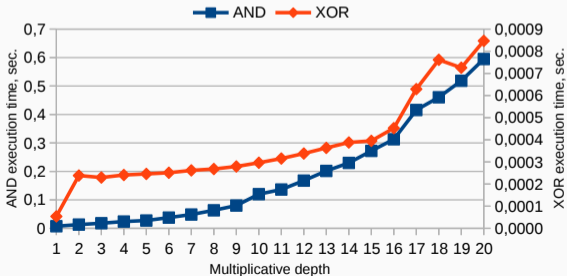
Ring LWE scheme example

- Scheme security given by:
 - Polynomial ring size
 - Noise level
- Homomorphic operations budget:
 - Ciphertext coefficient size to noise ratio

LWE problem

- Find $\mathbf{s} \in \mathbb{Z}_q^n$ given polynomial many samples of

$$(\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e)$$



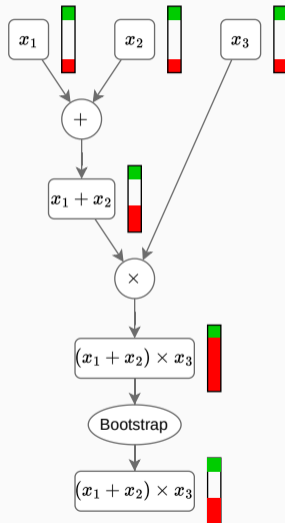
Ciphertext noise

Inherent to each ciphertext

- Ensures scheme security
- **Noise** increases after each operation
 - Multiplication \gg addition
- **Message** will be lost if noise overlaps

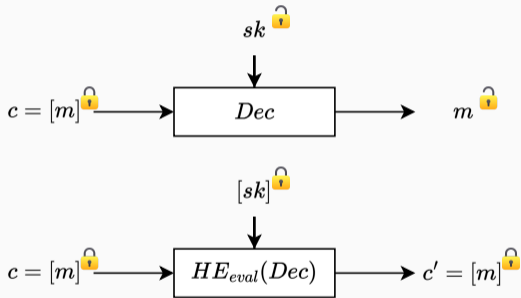
Bootstrapping procedure

- Decreases ciphertext noise



Bootstrapping

- Introduced by Gentry in 2009
- Evaluate decryption algorithm “homomorphically”
- Bootstrapping noise \equiv decryption circuit noise



Features

- Many messages packed into a single ciphertext
- One instruction over multiple data (aka SIMD)
- Bootstrap is slow, but multiple messages at once

BFV/BGV

- Modular ring plaintext:
 - slots mod p
 - slot add/multiply/rotate

CKKS

- Real ring plaintext:
 - fixed-point slots
 - slot add/multiply/rotate

Fast bootstrapping schemes

Features

- Encrypt a single message per ciphertext
- Bootstrap is fast
- Arbitrary function evaluation in addition to noise reduction
 - *Functional Bootstrapping (FBS)*

FHEW

- Focus on 2-input NAND gates
 - functionally complete
- Extension to multi-input Boolean gates

TFHE

- All 2-input gates and 3-input MUX, $10mS$ per gate
- Binary-decision diagrams and deterministic automata
- Arbitrary multi-output gates

Why do we need HE compilers?

- Homomorphic encryption schemes are low-level by construction
 - Additions and multiplications, more or less
 - Difficult to implement “efficient” applications in this context
- Evaluation time depends on the structure of the evaluated circuit:
 - E.g. sum of 2 binary numbers: ripple carry or carry-lookahead?
- *Circuit optimization tools are needed*

Existing HE compilers

- Cingulata [CDS15], Marble [VS18], Ramparts [ACTD⁺19], HEIR ...

Bootstrap number minimization

Bootstrap problem

Input

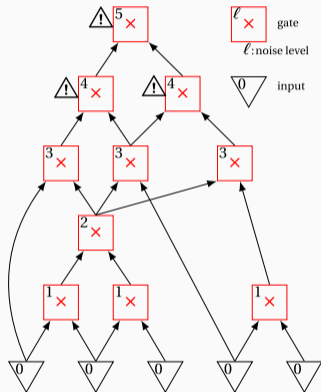
- A leveled HE scheme which supports up to L multiplications
 - Multiplicative depth L
- An arithmetic circuit of multiplicative depth $> L$ to evaluate

Problem

- Minimize the number of bootstraps needed to evaluate the circuit
 - Find a bootstrap placement

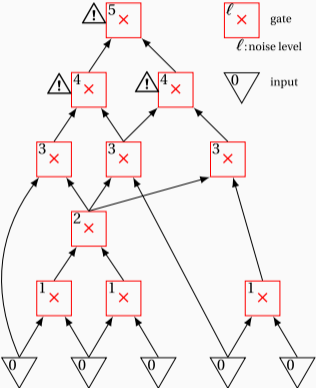
Bootstrap problem solution example

Input circuit, $L = 3$

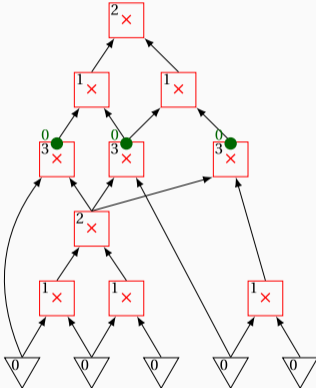


Bootstrap problem solution example

Input circuit, $L = 3$

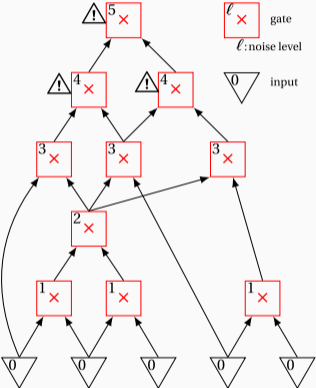


Naive map

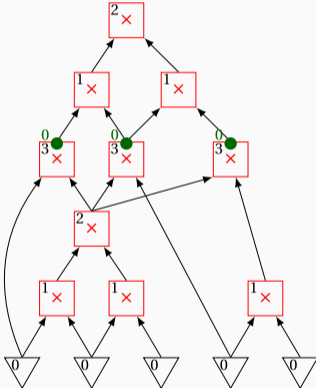


Bootstrap problem solution example

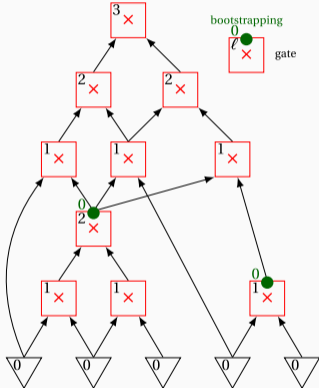
Input circuit, $L = 3$



Naive map



Optimal map



Complexity

- Polynomial for $L = 1$
 - Reduction to min-cut problem
- NP-hard for $L \geq 2$

Solutions

- Mixed-integer linear programming [PV15]
- Polynomial-time L -approximation algorithm [BLMZ17]

Particularities

- Noise budget asymmetry between fresh and bootstrapped ciphertexts
- What is the optimal L for a circuit
 - i.e minimize execution time instead of bootstrap count

AES homomorphic evaluation [GHS12]

- Uses HElib¹ BGV scheme
- Plaintext slots: 120 (no bootstrap) or 60 (bootstrap)
- Bootstrapped version is slower but allows further computations
 - 2 bootstraps \approx 80% of execution

Test	m	$\phi(m)$	lvls	$ Q $	security	params/key-gen	Encrypt	Decrypt	memory
no bootstrap	53261	46080	40	886	150-bit	26.45 / 73.03	245.1	394.3	3GB
bootstrap	28679	23040	23	493	123-bit	148.2 / 37.2	1049.9	1630.5	3.7GB

¹<https://github.com/homenc/HElib>

Multiplicative depth minimization

Multiplicative depth minimization problem

Problem

- Minimize the multiplicative depth of a circuit
 - Arithmetic or Boolean

Goal

- Decrease circuit multiplicative depth
 - Faster homomorphic evaluation
 - Smaller ciphertext sizes and parameters
- Orthogonal to bootstrap number minimization

Main idea

- Replace *critical* subcircuits with functionally equivalent counterparts with lower multiplicative depth

Some existing works

- Rewrite critical paths [CAS17] or cones [ACS20]

Critical nodes

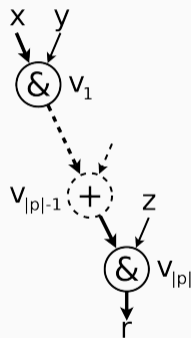
- Nodes which belong to circuit paths with longest multiplicative depth
- Rewriting critical nodes allows to reduce the overall multiplicative depth

Main idea

- Rewrite all multiplicative depth-2 critical paths

Two step rewrite

Multiplicative depth-2 critical path



Multiplicative depth of r is $\ell(x) + 2$

Critical path rewrite heuristic

Main idea

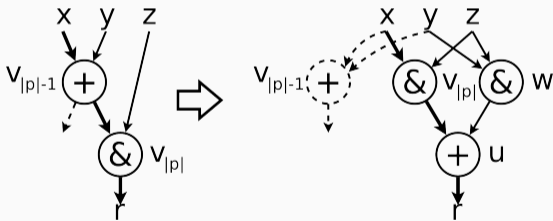
- Rewrite all multiplicative depth-2 critical paths

Two step rewrite

- Move multiplications up
 - One more multiplication

Multiplication move up operator

$$(x + y) \cdot z = x \cdot z + yz$$



Critical path rewrite heuristic

Main idea

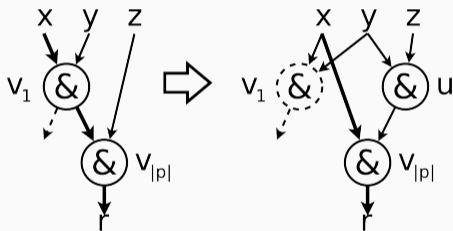
- Rewrite all multiplicative depth-2 critical paths

Two step rewrite

- Move multiplications up
 - One more multiplication
- Depth-2 to 1 transformation
 - Potentially one more multiplication

Multiplicative depth reduce operator

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$



If $\ell(y) < \ell(x)$ and $\ell(z) < \ell(x)$ then the multiplicative depth of r decreases from $\ell(x) + 2$ to $\ell(x) + 1$

Boolean circuits

- EPFL combinational benchmark suite
- Circuits were optimized and mapped to {AND, XOR} gates beforehand

Experiments

- Critical path rewrite heuristic
- Executed 10 times with random seeds (get unique rewrite orders)
- Output circuit with minimum multiplicative depth

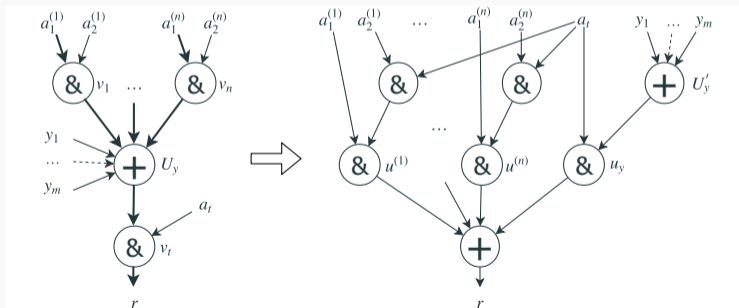
Critical path rewrite results

circuit	initial				heuristic		MD
	#inps	#outs	MD	#AND	MD	#AND	init/heur
adder	256	129	255	509	11	1125	23.2
div	128	128	4253	25219	1463	31645	2.9
max	512	130	204	2832	27	4660	7.6
multiplier	128	128	254	14389	59	17942	4.3
square	64	128	247	9147	28	10478	8.8
arbiter	256	129	87	11839	42	8582	2.1
i2c	147	142	15	1161	8	1185	1.9
mem_ctrl	1204	1231	110	44795	45	49175	2.4
priority	128	8	203	676	102	1106	2.0
router	60	30	21	167	11	204	1.9

MD - multiplicative depth

Critical cones

- Generalization of depth 2 critical paths
- Rewriting a cone is equivalent to rewriting n critical paths
 - More optimization possibilities



Critical cone rewrite results

circuit	critical path rewrite			critical cone rewrite		
	MD	#AND	MD init/heur	MD	#AND	MD init/heur
adder	11	1125	23.2	9	16378	28.3
div	1463	31645	2.9	532	190855	8
max	27	4660	7.6	26	7666	7.8
multiplier	59	17942	4.3	57	23059	4.5
square	28	10478	8.8	26	11306	9.3
arbiter	42	8582	2.1	10	5183	8.7
i2c	8	1185	1.9	7	1213	2.1
mem_ctrl	45	49175	2.4	40	54816	2.4
priority	102	1106	2.0	102	876	2.0
router	11	204	1.9	11	198	1.9

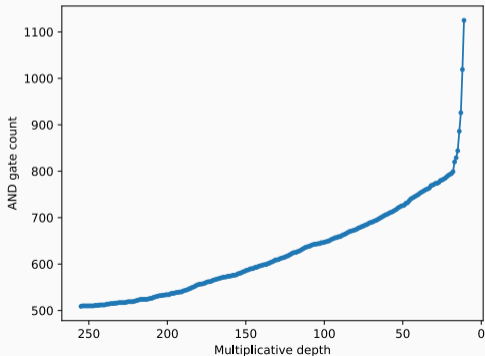
MD - multiplicative depth

Is multiplicative depth minimization always useful?

Trade-off between AND count and multiplicative depth

- At some point the gain from multiplicative depth decrease is canceled out by the number of additional AND gates

Adder benchmark intermediary circuits



Considering HE evaluation

- Cone rewrite heuristic
- Store intermediary circuits with distinct multiplicative depths
- Minimum multiplicative depth speedup (“min MD”)
- Highest speedup (“best”)

circuit	speedup	
	min MD	best
adder	44.9	408.3
div	11.0	40.3
max	32.0	61.0
multiplier	15.7	17.5
square	105.8	109.3
arbiter	257.9	257.9
i2c	5.2	5.2
mem_ctrl	7.4	7.4
priority	3.4	3.4
router	3.5	3.5

MD - multiplicative depth

Circuit mapping to functional bootstrappings

Fast bootstrapping schemes

Features

- Encrypt a single message per ciphertext
- Bootstrap is fast
- Arbitrary function evaluation in addition to noise reduction
 - *Functional Bootstrapping (FBS)*

FHEW

- Focus on 2-input NAND gates
 - functionally complete
- Extension to multi-input Boolean gates

TFHE

- All 2-input gates and 3-input MUX, $10mS$ per gate
- Binary-decision diagrams and deterministic automata
- Arbitrary multi-output gates

Functional bootstrapping

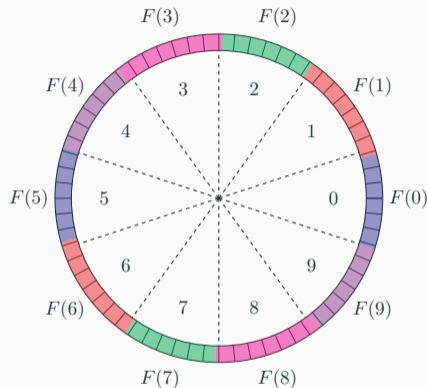
FBS

- Evaluate any function $F : \mathbb{Z}_p \rightarrow \mathbb{Z}_p$

Cost

- Depends mainly on the precision of the plaintext space p
- FBS execution time, library `tfhe-rs`:

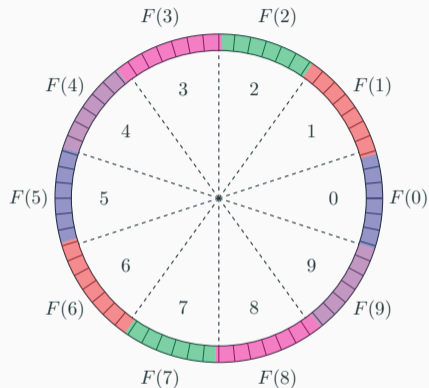
p	4	16	64	256
execution time (ms)	6	11	99	458



Torus split for plaintext space \mathbb{Z}_{10}

Non-power-of-two FBS

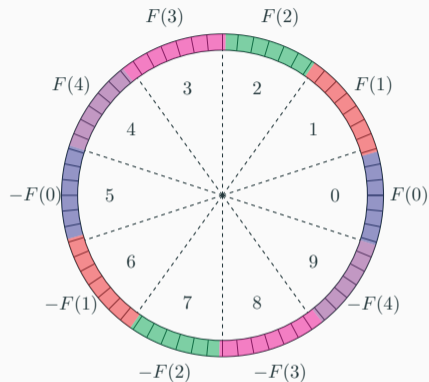
- FBS supports any plaintext space \mathbb{Z}_p
- Slightly slower when $p \nmid N$
 - N is the RLWE polynomial ring size



Torus split for plaintext space \mathbb{Z}_{10}

Negacyclic FBS

- Evaluate any function $F : \mathbb{Z}_{2p} \rightarrow \mathbb{Z}_p$ which verifies:
 - $F(x) = -F(x + p)$
- Applies to negacyclic RLWE polynomial rings:
 - Such as TFHE's ring $\mathbb{T} \bmod X^N + 1$



Torus split for negacyclic plaintext space \mathbb{Z}_5

Generalize FBS to n -input functions

$$f(x_0, \dots, x_{n-1}) = F \circ \phi(x_0, \dots, x_{n-1})$$

Steps:

1. Combine LWE samples x_0, \dots, x_{n-1} using a linear combination ϕ
 - Cheap, linear combination with public values
2. Apply FBS to evaluate F

Valid linear combination

- A linear combination ϕ is *valid* for a function f if it can unambiguously distinguish its image:
 - More formally, $\forall \mathbf{x}, \mathbf{x}'$ such that $f(\mathbf{x}) \neq f(\mathbf{x}') \implies \phi(\mathbf{x}) \neq \phi(\mathbf{x}')$

Linear combination size

- The *image size* of a linear combination ϕ is the smallest plaintext space needed to evaluate ϕ
 - E.g. the image of $2 \cdot x + 3 \cdot y$ is $\{0, 2, 3, 5\}$ and its image size is \mathbb{Z}_6

Linear combination search problem

- Given an n -input function f find a *valid* linear combination ϕ with minimal image size
 - Smaller image sizes mean smaller plaintext spaces, thus cheaper FBS computations

Hard problem

- Exact methods, intractable for large n
- Heuristics

Symmetric Boolean functions

- $\phi(\mathbf{x}) = \sum_i x_i$
- The output depends on the number of set inputs, not their position
 - n -input AND/OR/XOR gates, majority gate, etc.
- Image size linear in n

Arbitrary Boolean functions

- $\phi(\mathbf{x}) = \sum_i x_i \cdot 2^i$
- Functionally complete
- Exponential image size \mathbb{Z}_{2^n}
- Expensive multi-input FBS

Definition

Partition a Boolean circuit so that each partition is executed by one FBS

- The FBS precision (plaintext size) is fixed
- A valid linear combination is outputted for each partition

Goals

- Reduce the number of FBSs in the mapped circuit
- Ideally, it will also minimize the execution time

Hand-optimized FBS circuits

- Cryptographic algorithms (used for FHE trans-ciphering)
 - Trivium/Kreyvium [BOS23]
 - AES [TCBS23]
- Use efficiency tricks
 - Negacyclic functions, large plaintext spaces (\mathbb{Z}_{2^k}), ..
- Drawbacks:
 - Difficult and time-consuming
 - Not always best solution found as we shall see later

Mapping Boolean circuits to FBS

Input

- A Boolean circuit with 2-input gates
- A maximal plaintext space size (FBS precision)

Fast heuristic

- Traverse circuit gates in topological order
- Construct the linear combination of a gate from the linear combinations of its 2 inputs
 - Lazy bootstrap gate inputs if linear combination size is too large

Why is it fast?

- Circuit nodes are visited only once
- Linear combinations are built incrementally
 - Search only 2-coefficient linear combinations

Exhaustive search

- Test all linear combinations $\alpha \cdot x + \beta \cdot y$ and keep the smallest valid one
 - $|\alpha| \leq \|\mathbf{vt}_y\|_\infty$ and $|\beta| \leq \|\mathbf{vt}_x\|_\infty$
- Optimal solution always found
 - Faster than integer linear programming
- Test vector validity in case of negacyclic rings

Example

tt_{AND}	$vt_{a+b} \times vt_c$	(α, β)		
		(1, 1)	(2, 1)	(1, 2)
0	0,0	0	0	0
0	0,1	1	1	2
0	1,0	1	2	1
1	1,1	2	3	3
0	1,0	1	2	1
1	1,1	2	3	3
0	2,0	2	4	2
0	2,1	3	5	4
Lincomb size			6	5
Test vector		invalid	000100	00010
Negacyclic			0001	0001

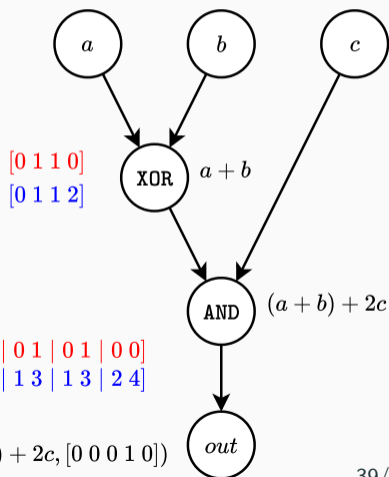
truth table

multi-value table

[0 1] [0 1]

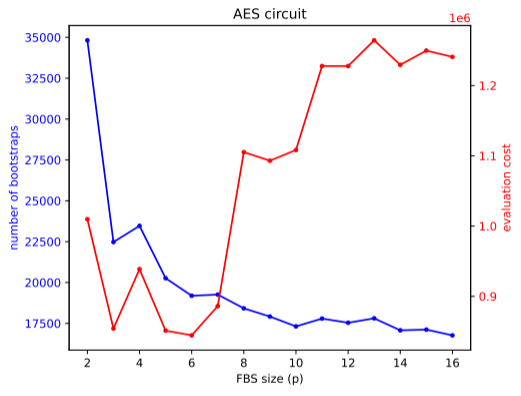
[0 1] [0 1]

[0 1] [0 1]



Map AES-128 circuit

- Best solution for FBS size 6
 - 45% less bootstrappings
 - 17% faster execution
 - In comparison to the naive 1 gate - 1 FBS approach
- Bootstrap count vs FBS size
 - One would have expected a monotonic decrease
 - Node visit order influences heuristic solution quality
 - Lazy bootstrapping strategy results in more FBS



Bristol fashion circuit aes_128.txt²

²github.com/mkskeller/bristol-fashion

EPFL benchmarks results

- Output solution with smallest execution cost for FBS sizes 2..15
- On average:
 - 37% decrease in execution cost
 - 58% less bootstrappings
- Solutions have FBS sizes < 8 in most cases

bench	cost	#boots.	FBS size
adder	-64%	-75%	5 (7)
hyp	-41%	-63%	7 (14)
log2	-38%	-57%	5 (10)
multiplier	-50%	-68%	7 (14)
sin	-37%	-60%	7 (14)
arbiter	-48%	-64%	5 (8)
ctrl	-40%	-61%	7 (12)
int2float	-49%	-67%	7 (13)
priority	-40%	-60%	6 (11)
router	-42%	-63%	7 (14)
avg.	-37%	-58%	

Kreyvium stream cipher

- 128 bits of security
- Implement 2 versions
 - Thwart heuristic “greediness”
 - Changed operations order in out_*
- Compare with hand-optimized versions from [BOS23]

```
t1 = s66 ^ s93
```

```
t2 = s162 ^ s177
```

```
t3 = s243 ^ s288 ^ k127
```

```
out = t1 ^ t2 ^ t3
```

```
out_t1 = t1 ^ (s91 & s92) ^ s171 ^ iv127
```

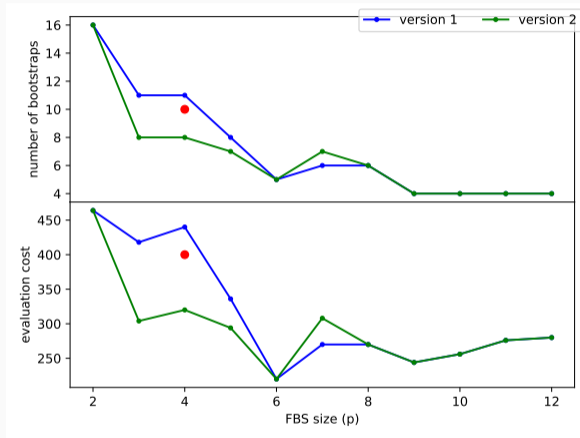
```
out_t2 = t2 ^ (s175 & s176) ^ s264
```

```
out_t3 = t3 ^ (s286 & s287) ^ s69
```

One iteration of Kreyvium

Kreyvium results

- Best solution for $p = 6$
 - 5 FBS
 - 45% faster than $p = 3$
 - $2\times$ faster than input circuit
- Better than the hand-optimized solution
 - 8 FBS instead of 10
 - Smaller FBS size $p = 3$



Red dots are [BOS23] result

FBS mapped Kreyvium



- FBS size 3
- Heavy use of negacyclic property

```
m1 = 2 - s66 + s93 - s162 + s177
m2 = Bootstrap(m1, [0, 1, 0, 1, 0])
m3 = 1 - s66 + s93 + s171 + iv127
m4 = Bootstrap(m3, [1, 0, 1, 0, 1])
m5 = 1 - s162 + s177 + s264
m6 = Bootstrap(m5, [1, 0, 1, 0])
m7 = 1 - s243 + s288 + k127 + s69
m8 = Bootstrap(m7, [1, 0, 1, 0, 1])
m9 = 1 + m2 - s243 + s288 + k127
out = Bootstrap(m9, [1, 0, 1, 0, 1])
m10 = 3 * m4 + s91 + s92
out_t1 = Bootstrap(m10, [0, 0, 1, 1, 1, 0])
m11 = 3 * m6 + s175 + s176
out_t2 = Bootstrap(m11, [0, 0, 1, 1, 1, 0])
m12 = 3 * m8 + s286 + s287
out_t3 = Bootstrap(m12, [0, 0, 1, 1, 1, 0])
```

Key takeaways

- Homomorphic encryption is still a young area of research in the field of cryptography
- Many open optimization problems in the HE “compute model”
 - Key-switch placement in leveled HE schemes
 - Bootstrap placement + multiplicative depth reduction
 - FBS + leveled HE operations
- Similarities with multi-party computation
 - Multiplicative depth in arithmetic circuits
 - Similarities between FBS mapping and arithmetic garbling circuits

Questions?

-  Pascal Aubry, Sergiu Carpov, and Renaud Sirdey.
Faster homomorphic encryption is not enough: Improved heuristic for multiplicative depth minimization of boolean circuits.
In *Topics in Cryptology–CT-RSA 2020: The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24–28, 2020, Proceedings*, pages 345–363. Springer, 2020.
-  David W Archer, José Manuel Calderón Trilla, Jason Dagit, Alex Malozemoff, Yuriy Polyakov, Kurt Rohloff, and Gerard Ryan.
Ramparts: A programmer-friendly system for building homomorphic encryption applications.

In *Proceedings of the 7th acm workshop on encrypted computing & applied homomorphic cryptography*, pages 57–68, 2019.

 Fabrice Benhamouda, Tancrede Lepoint, Claire Mathieu, and Hang Zhou.


Optimization of bootstrapping in circuits.




In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2423–2433. SIAM, 2017.

 Thibault Balenbois, Jean-Baptiste Orfila, and Nigel Smart.

Trivial transciphering with trivium and tfhe.

In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 69–78, 2023.

-  Sergiu Carpov, Pascal Aubry, and Renaud Sirdey.
A multi-start heuristic for multiplicative depth minimization of boolean circuits.
In *International Workshop on Combinatorial Algorithms*, pages 275–286. Springer, 2017.
-  Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey.
Armadillo: a compilation chain for privacy preserving applications.
In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, pages 13–19, 2015.

-  Craig Gentry, Shai Halevi, and Nigel P Smart.
Homomorphic evaluation of the aes circuit.
In *Annual Cryptology Conference*, pages 850–867. Springer, 2012.
-  Marie Paindavoine and Bastien Vialla.
Minimizing the number of bootstrappings in fully homomorphic encryption.
In *International Conference on Selected Areas in Cryptography*, pages 25–43.
Springer, 2015.
-  Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey.
A homomorphic aes evaluation in less than 30 seconds by means of tfhe.
In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 79–90, 2023.



Alexander Viand and Hossein Shafagh.

Marble: Making fully homomorphic encryption accessible to all.

In Proceedings of the 6th workshop on encrypted computing & applied homomorphic cryptography, pages 49–60, 2018.