Revisiting the functional bootstrap in TFHE

Antonio Guimarães, Edson Borin, and Diego F. Aranha

University of Campinas, Brazil, and Aarhus University, Denmark

Context

Efficiently evaluating **non-linear** functions with **high precision** is a **challenge** for Fully Homomorphic Encryption schemes.

CKKS [4]

- Approximations using Taylor, Fourier, and Chebyshev series.
- Good performance for low-precision approximations.

TFHE [5]

- Circuits are implemented using binary logic gates.
- Low throughput of operations.
- New approach: Functional Bootstrap [1]

The Functional Boostrap in TFHE

- All known FHE schemes are noisy.
  - The noise increases with the arithmetic operations.
  - Eventually it would affect correctness.
- Bootstrap
  - A usually expensive process that resets the noise.
- Functional Bootstrap [1]
  - Evaluates a function within the bootstrap at (almost) no additional cost.
  - In TFHE, the bootstrap is a Lookup Table (LUT) evaluation, which is great for non-linear functions.
- Problem
  - Large look-up tables require large parameters.
  - TFHE efficiency comes from using small parameters.

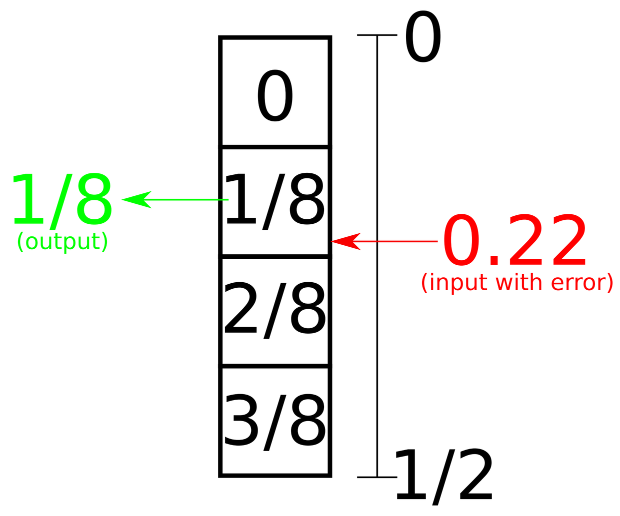

Figure 1. *TFHE Bootstrap Example, evaluating a floor function over the Torus discretized in multiples of* $1/8$.

Two methods for evaluating large Look-Up Tables

- The message is decomposed in d digits. Each of them is encrypted in a ciphertext c_i.
- Tree-based method
  - Each ciphertext c_i is used to evaluate 2^{d-i} LUTs
  - The results of the evaluation using c_i are used to create new LUTs for c_{i+1}.
  - Figure 2 shows an example. Each rectangle is a small LUT evaluation.
- Chaining-method
  - A more functionally restricted method, which presents better error growth behavior.
  - Suitable for carry-like functions.
  - More details in the paper.

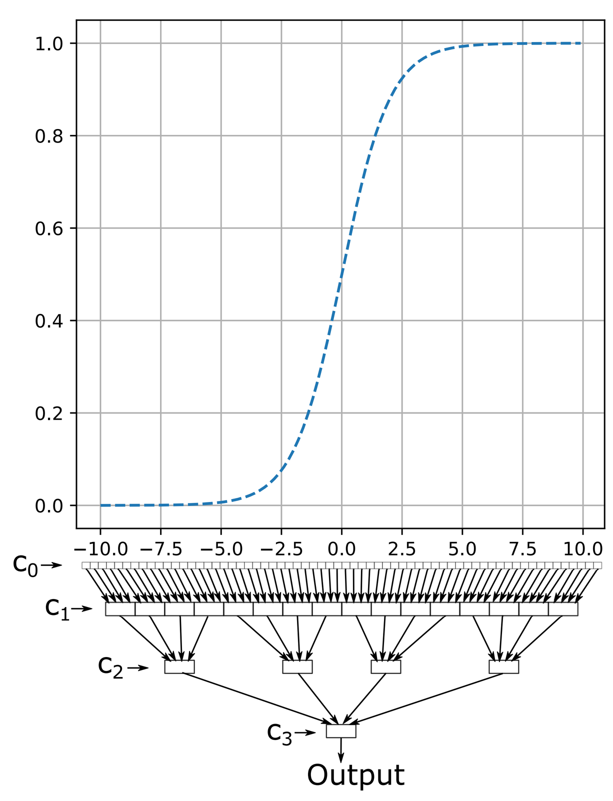

Figure 2. *Tree-PBS Example. 8-bit sigmoid.*

Multiplications with linear error growth

- Typically, multiplications increase the error variance **quadratically**.
- Multi-value extract method:
  - Allows obtaining multiple copies of the same ciphertext, with **independent** error.
  - One can perform multiplications with linear error growth by adding these copies.
  - The method is computationally inexpensive and introduces a very low probability of error.
  - We introduce it to improve the batch bootstrapping technique of Carpov *et al.*[3]

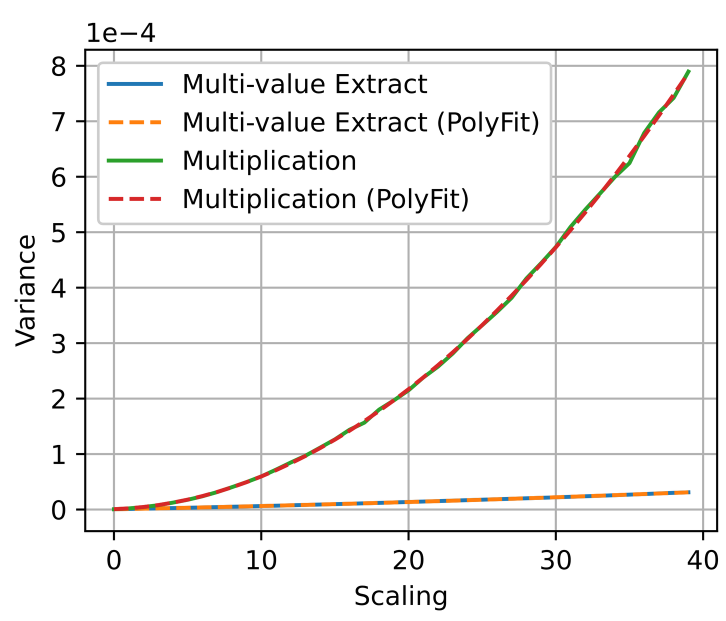

Figure 3. *Multi-value extract after 25-bit precision gadget decomposition ($\ell = 5$ and $\log_2(B_g) = 5$)*

Practical Results

Compared to previous literature, our methods are **faster** and have a **lower probability of error** for similar or **higher security levels**. On the other hand, they might require **larger keys** in some cases.

- 32-bit Integer comparison

Source	λ	Key Size	Error Rate	Time (ms)	Speedup
Bourse *et al.*[2]	90	1.2	-50*	2232*	1.75
	109	3.4	-47*	3902*	1.00
	211	4.6	-89*	3840*	1.02
Zhou *et al.*[7]	80	0.3	negl.	1143.2	0.93
	127	0.3	negl.	1867.2	0.57
This work (1)	127	4.3	-26.51	334.1	**3.19**
This work (2)	127	6.5	**-129.58**	396.4	**2.68**

Table 1. *32-bit integer comparison. Key size in GB, error rate in* $\log_2$. ** Data provided by the authors. We adjusted the speedup according to the differences in execution environments.*

- 8-bit ReLU

Source	λ	Key Size	Error Rate	Time (ms)	Speedup
Lou *et al.*[6]	80	0.3	negl.	380	1.59
	127	0.3	negl.	603.1	1.00
Zhou *et al.*[7]	80	0.3	negl.	64.8	9.31
	127	0.3	negl.	103.1	5.85
This work (1)	127	4.3	**-137.1**	86.4	**6.98**
This work (2)	127	6.5	-181.0	103.6	5.82

Table 2. *8-bit Rectified Linear Unit (ReLU). Key size in GB, error rate in* $\log_2$.

- 6-bit-to-6-bit Look-Up Table

Source	λ	Key Size	Error Rate	Time (ms)	Speedup
Carpov *et al.*[3]	≥128	8	-26.94	1570*	1.00
This work (1)	127	4.3	-59.59	378.2	2.49
This work (2)	127	6.5	-134.84	457.9	2.06

Table 3. *6-bit-to-6-bit generic function. Key size in GB, error rate in* $\log_2$. ** Data provided by the authors. We adjusted the speedup according to the differences in execution environments.*

Implementation

To reproduce the results of this poster, using the original TFHE library, see:

- `https://github.com/antoniocgj/FBT-TFHE`
- `https://doi.org/10.46586/tches.v2021.i2.229-253`

For an **updated** implementation containing all the techniques presented in this paper and **many others**, see:

- MOSFHET: Optimized Software for FHE over the Torus
- `https://github.com/antoniocgj/MOSFHET`
- `https://eprint.iacr.org/2022/515`

antonio.guimaraes@ic.unicamp.br