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]
- Multiplications with linear error growth
- The Functional Bootstrap 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. Figure 1. TFHE Bootstrap Example, evaluating a floor function over the Torus discretized in multiples of 1/8.

The Functional Bootstrap in TFHE
- Problem
  - Large look-up tables require large parameters.
  - TFHE efficiency comes from using small parameters.

Two methods for evaluating large Look-Up Tables
- The message is decomposed in d digits. Each of them is encrypted in a ciphertext ci.
- Tree-based method
  - Each ciphertext ci is used to evaluate 2i LUTs.
  - The results of the evaluation using ci are used to create new LUTs for ci+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.

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]

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

<table>
<thead>
<tr>
<th>Source</th>
<th>λ</th>
<th>Key Size</th>
<th>Error Rate</th>
<th>Time (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bourse et al.[2]</td>
<td>90</td>
<td>1.2</td>
<td>-50°</td>
<td>2232°</td>
<td>1.75</td>
</tr>
<tr>
<td>Zhou et al.[7]</td>
<td>211</td>
<td>4.6</td>
<td>-89°</td>
<td>3840°</td>
<td>1.02</td>
</tr>
<tr>
<td>This work (1)</td>
<td>80</td>
<td>0.3</td>
<td>negl.</td>
<td>1143.2</td>
<td>0.93</td>
</tr>
<tr>
<td>This work (2)</td>
<td>127</td>
<td>0.3</td>
<td>negl.</td>
<td>1867.2</td>
<td>0.57</td>
</tr>
</tbody>
</table>

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

- 8-bit ReLU

<table>
<thead>
<tr>
<th>Source</th>
<th>λ</th>
<th>Key Size</th>
<th>Error Rate</th>
<th>Time (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhou et al.[7]</td>
<td>80</td>
<td>0.3</td>
<td>negl.</td>
<td>380</td>
<td>1.59</td>
</tr>
<tr>
<td>This work (1)</td>
<td>127</td>
<td>0.3</td>
<td>negl.</td>
<td>603.1</td>
<td>1.00</td>
</tr>
<tr>
<td>This work (2)</td>
<td>80</td>
<td>0.3</td>
<td>negl.</td>
<td>64.8</td>
<td>9.31</td>
</tr>
<tr>
<td>This work (3)</td>
<td>127</td>
<td>0.3</td>
<td>negl.</td>
<td>103.1</td>
<td>5.85</td>
</tr>
</tbody>
</table>

Table 2. 8-bit Rectified Linear Unit (ReLU). Key size in GB, error rate in logλ.

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

<table>
<thead>
<tr>
<th>Source</th>
<th>λ</th>
<th>Key Size</th>
<th>Error Rate</th>
<th>Time (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carpov et al.[3]</td>
<td>≥128</td>
<td>8</td>
<td>negl.</td>
<td>2232</td>
<td>1.00</td>
</tr>
<tr>
<td>This work (1)</td>
<td>127</td>
<td>4.3</td>
<td>-59.59</td>
<td>378.2</td>
<td>2.49</td>
</tr>
<tr>
<td>This work (2)</td>
<td>127</td>
<td>6.5</td>
<td>-129.58</td>
<td>396.4</td>
<td>2.68</td>
</tr>
</tbody>
</table>

Table 3. 6-bit-to-6-bit generic function. Key size in GB, error rate in logλ. * 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/FST-TFHE](https://github.com/antoniocgj/FST-TFHE)
- [https://doi.org/10.46586/tches.v2021.i2.229-253](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://github.com/antoniocgj/MOSFHET)
  - [https://eprint.iacr.org/2022/515](https://eprint.iacr.org/2022/515)

antonio.guimaraes@ic.unicamp.br