Short Talk: Best Solution for FHERMA Lookup Table Challenge

Jules Dumezy

Workshop on Encrypted Computing & Applied Homomorphic Cryptography

13 Octobre 2025

Table of contents

Table of contents

The FHERMA Lookup Table Challenge

The FHERMA Lookup Table Challenge

Challenge setting [AAAB+25]:

The FHERMA Lookup Table Challenge

Challenge setting [AAAB⁺25]:

- FHE scheme: BFV

The FHERMA Lookup Table Challenge

Challenge setting [AAAB+25]:

- FHE scheme: BFV
- Ring dimension: $N = 32768 \quad (2^{15})$

The FHERMA Lookup Table Challenge

Challenge setting [AAAB$^+$25]:

- FHE scheme: BFV
- Ring dimension: $N = 32768 \quad (2^{15})$
- Plaintext modulus: $t = 65537 \quad$ (plaintext space $\mathcal{P} = \mathbb{F}_t$ with t a prime)

The FHERMA Lookup Table Challenge

Challenge setting [AAAB+25]:

- FHE scheme: BFV
- Ring dimension: $N = 32768 \quad (2^{15})$
- Plaintext modulus: $t = 65537 \quad$ (plaintext space $\mathcal{P} = \mathbb{F}_t$ with t a prime)
- Number of slots/LUT input size: $n = 2048 \quad$ (2^{11}, can be extended to larger sizes $\leq N$)

The FHERMA Lookup Table Challenge

Challenge setting [AAAB$^+$25]:

- FHE scheme: BFV
- Ring dimension: $N = 32768 \quad (2^{15})$
- Plaintext modulus: $t = 65537 \quad$ (plaintext space $\mathcal{P} = \mathbb{F}_t$ with t a prime)
- Number of slots/LUT input size: $n = 2048 \quad$ (2^{11}, can be extended to larger sizes $\leq N$)

Goal: Given a vector $(y_0, \ldots, y_{n-1}) \in \mathbb{F}_t^n$, retrieve the x-th element for $x \in [\![0, n-1]\!]$

The FHERMA Lookup Table Challenge

Challenge setting [AAAB+25]:

- FHE scheme: BFV
- Ring dimension: $N = 32768 \quad (2^{15})$
- Plaintext modulus: $t = 65537 \quad$ (plaintext space $\mathcal{P} = \mathbb{F}_t$ with t a prime)
- Number of slots/LUT input size: $n = 2048 \quad$ (2^{11}, can be extended to larger sizes $\leq N$)

Goal: Given a vector $(y_0, \ldots, y_{n-1}) \in \mathbb{F}_t^n$, retrieve the x-th element for $x \in [\![0, n-1]\!]$

This is equivalent to computing $f(x)$ for $f : \begin{cases} [\![0, n-1]\!] \to \mathbb{F}_t \\ x \mapsto y_x \end{cases}$

The FHERMA Lookup Table Challenge

Challenge setting [AAAB$^+$25]:

- FHE scheme: BFV
- Ring dimension: $N = 32768 \quad (2^{15})$
- Plaintext modulus: $t = 65537$ (plaintext space $\mathcal{P} = \mathbb{F}_t$ with t a prime)
- Number of slots/LUT input size: $n = 2048$ (2^{11}, can be extended to larger sizes $\leq N$)

Goal: Given a vector $(y_0, \ldots, y_{n-1}) \in \mathbb{F}_t^n$, retrieve the x-th element for $x \in [\![0, n-1]\!]$

This is equivalent to computing $f(x)$ for $f : \begin{cases} [\![0, n-1]\!] \to \mathbb{F}_t \\ x \mapsto y_x \end{cases}$

Allowed operations: additions and multiplications on $\mathbb{F}_t$, vector rotations

The FHERMA Lookup Table Challenge

Challenge setting [AAAB$^+$25]:

- FHE scheme: BFV
- Ring dimension: $N = 32768 \quad (2^{15})$
- Plaintext modulus: $t = 65537$ (plaintext space $\mathcal{P} = \mathbb{F}_t$ with t a prime)
- Number of slots/LUT input size: $n = 2048$ (2^{11}, can be extended to larger sizes $\leq N$)

Goal: Given a vector $(y_0, \ldots, y_{n-1}) \in \mathbb{F}_t^n$, retrieve the x-th element for $x \in [\![0, n-1]\!]$

This is equivalent to computing $f(x)$ for $f : \begin{cases} [\![0, n-1]\!] \to \mathbb{F}_t \\ x \mapsto y_x \end{cases}$

Allowed operations: additions and multiplications on $\mathbb{F}_t$, vector rotations
$\to$ Inspired by Iliashenko et al. [IZ21]

Table of contents

Broadcasting the index

Broadcasting the index

The index is given as a vector $(x, 0, \ldots, 0) \in \mathbb{F}_t^n$

Broadcasting the index

The index is given as a vector $(x, 0, \dots, 0) \in \mathbb{F}_t^n$
We first want to obtain the vector $(x, \dots, x) \in \mathbb{F}_t^n$

Broadcasting the index

The index is given as a vector $(x, 0, \dots, 0) \in \mathbb{F}_t^n$
We first want to obtain the vector $(x, \dots, x) \in \mathbb{F}_t^n$
We can compute it only with vector rotations and additions as

$$(x, \dots, x) = (x, 0, \dots, 0) + (0, x, 0, \dots, 0) + \cdots + (0, \dots, 0, x).$$

Broadcasting the index

The index is given as a vector $(x, 0, \ldots, 0) \in \mathbb{F}_t^n$
We first want to obtain the vector $(x, \ldots, x) \in \mathbb{F}_t^n$
We can compute it only with vector rotations and additions as

$$(x, \ldots, x) = (x, 0, \ldots, 0) + (0, x, 0, \ldots, 0) + \cdots + (0, \ldots, 0, x).$$

This can be efficiently computed with $O(\log_2(n))$ operations with the fast rotation algorithm

Broadcasting the index

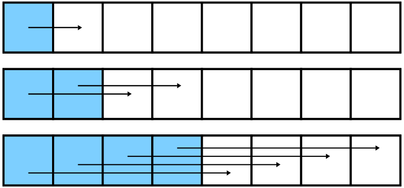
The index is given as a vector $(x, 0, \ldots, 0) \in \mathbb{F}_t^n$
We first want to obtain the vector $(x, \ldots, x) \in \mathbb{F}_t^n$
We can compute it only with vector rotations and additions as

$$(x, \ldots, x) = (x, 0, \ldots, 0) + (0, x, 0, \ldots, 0) + \cdots + (0, \ldots, 0, x).$$

This can be efficiently computed with $O(\log_2(n))$ operations with the fast rotation algorithm

Algorithm Fast Rotation and Add

procedure FastRotAdd(ct)
  for $i \in [\![0, \log_2(n) - 1]\!]$ **do**
    ct $\leftarrow$ Add(ct, Rotate(ct, -2^i))
  return ct



Creating a one-hot indicator with Fermat's little theorem

Creating a one-hot indicator with Fermat's little theorem

We subtract $(0, 1, 2, \ldots, n-1) \in \mathbb{F}_t^n$ to the previously broadcasted index:

$$(x, \ldots, x) - (0, 1, 2, \ldots, n-1) = (x, x-1, x-2, \ldots, x-(n-2), x-(n-1))$$

Creating a one-hot indicator with Fermat's little theorem

We subtract $(0, 1, 2, \ldots, n-1) \in \mathbb{F}_t^n$ to the previously broadcasted index:

$$(x, \ldots, x) - (0, 1, 2, \ldots, n-1) = (x, x-1, x-2, \ldots, x-(n-2), x-(n-1))$$

This new vector contains exactly one zero at the x-th position

Creating a one-hot indicator with Fermat's little theorem

We subtract $(0, 1, 2, \ldots, n-1) \in \mathbb{F}_t^n$ to the previously broadcasted index:

$$(x, \ldots, x) - (0, 1, 2, \ldots, n-1) = (x, x-1, x-2, \ldots, x-(n-2), x-(n-1))$$

This new vector contains exactly one zero at the x-th position

Theorem

For a prime t, and any integer a not divisible by t, we have:

$$a^{t-1} = 1 \mod p$$

Creating a one-hot indicator with Fermat's little theorem

We subtract $(0, 1, 2, \ldots, n-1) \in \mathbb{F}_t^n$ to the previously broadcasted index:

$$(x, \ldots, x) - (0, 1, 2, \ldots, n-1) = (x, x-1, x-2, \ldots, x-(n-2), x-(n-1))$$

This new vector contains exactly one zero at the x-th position

Theorem

For a prime t, and any integer a not divisible by t, we have:

$$a^{t-1} = 1 \mod p$$

Noticing that $t - 1 = 2^{16}$, we can use fast exponentiation with 16 repetitive squaring

Creating a one-hot indicator with Fermat's little theorem

We subtract $(0,1,2,\ldots,n-1) \in \mathbb{F}_t^n$ to the previously broadcasted index:

$$(x,\ldots,x) - (0,1,2,\ldots,n-1) = (x, x-1, x-2, \ldots, x-(n-2), x-(n-1))$$

This new vector contains exactly one zero at the x-th position

Theorem

For a prime t, and any integer a not divisible by t, we have:

$$a^{t-1} = 1 \mod p$$

Noticing that $t-1 = 2^{16}$, we can use fast exponentiation with 16 repetitive squaring
This result in the vector $(1,\ldots,1,\underbrace{0}_{x\text{-th element}},1,\ldots,1)$

Extraction and final summation

Extraction and final summation

By subtracting to $(1, \ldots, 1)$, we obtain the one-hot encoding $(0, \ldots, 0, \underbrace{1}_{x\text{-th element}}, 0, \ldots, 0)$

Extraction and final summation

By subtracting to $(1, \ldots, 1)$, we obtain the one-hot encoding $(0, \ldots, 0, \underbrace{1}_{x\text{-th element}}, 0, \ldots, 0)$

Then, we multiply it by the LUT $(y_0, \ldots, y_{n-1})$ to obtain $(0, \ldots, 0, y_x, 0 \ldots 0)$

Extraction and final summation

By subtracting to $(1, \ldots, 1)$, we obtain the one-hot encoding $(0, \ldots, 0, \underbrace{1}_{x\text{-th element}}, 0, \ldots, 0)$

Then, we multiply it by the LUT $(y_0, \ldots, y_{n-1})$ to obtain $(0, \ldots, 0, y_x, 0 \ldots 0)$

We can then reuse the FastRotAdd algorithm (with reversed rotation indices) to bring back y_x to the first slot

Extraction and final summation

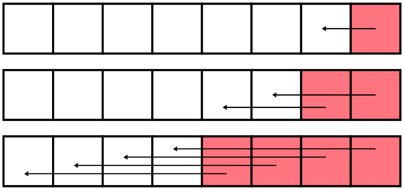
By subtracting to $(1, \ldots, 1)$, we obtain the one-hot encoding $(0, \ldots, 0, \underbrace{1}_{x\text{-th element}}, 0, \ldots, 0)$

Then, we multiply it by the LUT $(y_0, \ldots, y_{n-1})$ to obtain $(0, \ldots, 0, y_x, 0 \ldots 0)$

We can then reuse the FastRotAdd algorithm (with reversed rotation indices) to bring back y_x to the first slot

Algorithm Fast Rotation and Add

procedure FastRotAdd(ct)
1: **for** $i \in [\![0, \log_2(n) - 1]\!]$ **do**
2: ct $\leftarrow$ Add(ct, Rotate(ct, 2^i))
return ct



Algorithm workflow

Algorithm workflow

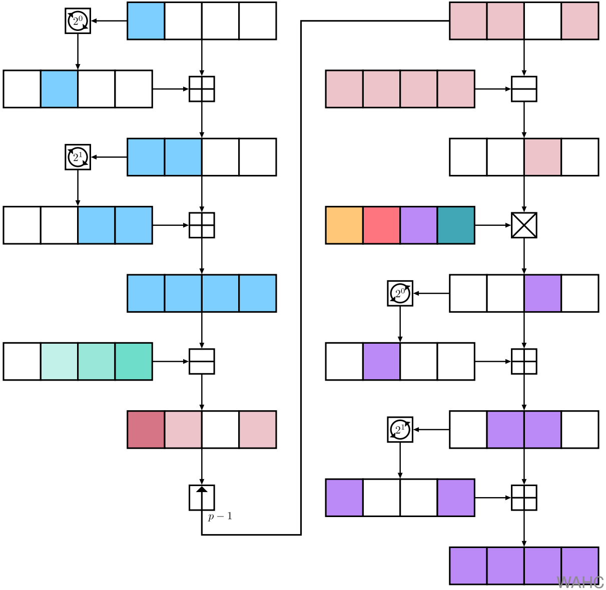


Table of contents

Implementation optimizations

Implementation optimizations

Here are several optimizations that we use to achieve the best performance:

Implementation optimizations

Here are several optimizations that we use to achieve the best performance:

- Use the right compilation flags (NATIVEOPT...)

Implementation optimizations

Here are several optimizations that we use to achieve the best performance:

- Use the right compilation flags (NATIVEOPT...)
- Use the optimized EvalSquare API of OpenFHE

Implementation optimizations

Here are several optimizations that we use to achieve the best performance:

- Use the right compilation flags (NATIVEOPT...)
- Use the optimized EvalSquare API of OpenFHE
- Reduce new ciphertext declaration

Implementation optimizations

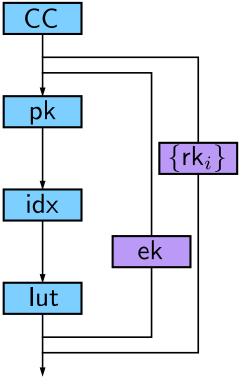
Here are several optimizations that we use to achieve the best performance:

- Use the right compilation flags (NATIVEOPT...)
- Use the optimized EvalSquare API of OpenFHE
- Reduce new ciphertext declaration
- Limit the number of rotation keys: instead of generating the keys for indexes $(1, 2, 4, \ldots, 2^{\log_2(n)-1}, -1, -2, -4, \ldots, -2^{\log_2(n)-1})$, we generate the keys for $(1, 2, 4, \ldots, 2^{\log_2(n)-1}, -(n-1))$
  Doing so, we need only an additional rotation, but halve the number of keys

Implementation optimizations

Here are several optimizations that we use to achieve the best performance:

- Use the right compilation flags (NATIVEOPT...)
- Use the optimized EvalSquare API of OpenFHE
- Reduce new ciphertext declaration
- Limit the number of rotation keys: instead of generating the keys for indexes $(1, 2, 4, \ldots, 2^{\log_2(n)-1}, -1, -2, -4, \ldots, -2^{\log_2(n)-1})$, we generate the keys for $(1, 2, 4, \ldots, 2^{\log_2(n)-1}, -(n-1))$
  Doing so, we need only an additional rotation, but halve the number of keys
- Asynchronous key deserialization:



Performance

Performance

Multiplicative size depends on LUT output size, that is the plaintext modulus t

Performance

Multiplicative size depends on LUT output size, that is the plaintext modulus t
We perform $O(\log_2(n))$ operations with n the size of the LUT/batch size

Performance

Multiplicative size depends on LUT output size, that is the plaintext modulus t
We perform $O(\log_2(n))$ operations with n the size of the LUT/batch size

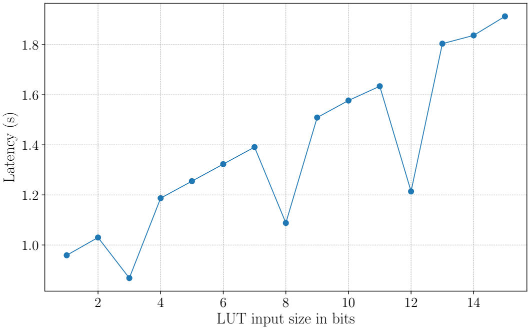


Figure: Performance of the proposed algorithm on a commodity laptop on OpenFHE v1.4.0

Table of contents

CKKS functional bootstrapping for LUT evaluation

CKKS functional bootstrapping for LUT evaluation

- Efficient LUT evaluation with FBT
  [AKP24, BKSS24]

CKKS functional bootstrapping for LUT evaluation

- Efficient LUT evaluation with FBT
  [AKP24, BKSS24]
- Higher latency, but (much) better throughput,
  and *compatible with BFV*

CKKS functional bootstrapping for LUT evaluation

- Efficient LUT evaluation with FBT [AKP24, BKSS24]
- Higher latency, but (much) better throughput, and *compatible with BFV*
- Joint work with the team of OpenFHE: [DAP^{+}25] accepted at CHES '26

CKKS functional bootstrapping for LUT evaluation

- Efficient LUT evaluation with FBT [AKP24, BKSS24]
- Higher latency, but (much) better throughput, and *compatible with BFV*
- Joint work with the team of OpenFHE: [DAP$^+$25] accepted at CHES '26
- Large LUTs evaluation: higher latency compared to BFV, but *3 orders of magnitude faster* in throughput

CKKS functional bootstrapping for LUT evaluation

- Efficient LUT evaluation with FBT [AKP24, BKSS24]
- Higher latency, but (much) better throughput, and *compatible with BFV*
- Joint work with the team of OpenFHE: [DAP$^+$25] accepted at CHES '26
- Large LUTs evaluation: higher latency compared to BFV, but *3 orders of magnitude faster* in throughput

$\log_2 p$	Latency (s)	Amz. time (ms)
10	46.17	0.70
12	52.46	0.80
14	98.94	1.51
16	112.17	1.71
20	1,014.8	7.74

Table of contents

Conclusion

Conclusion

- Low latency (encrypted) LUT evaluation with BFV

Conclusion

- Low latency (encrypted) LUT evaluation with BFV
- Between 2-bit and 15-bit LUTs, could be further extended

Conclusion

- Low latency (encrypted) LUT evaluation with BFV
- Between 2-bit and 15-bit LUTs, could be further extended
- One of many components of the FHERMA cookbook (scan QR code for more details)

Conclusion

- Low latency (encrypted) LUT evaluation with BFV
- Between 2-bit and 15-bit LUTs, could be further extended
- One of many components of the FHERMA cookbook (scan QR code for more details)
- Building blocks for easily creating FHE applications

Thank you !

CEA Nano-INNOV
91 120 Palaiseau Cedex
France
jules.dumezy@cea.fr

References I

Janis Adamek, Aikata Aikata, Ahmad Al Badawi, Andreea Alexandru, Armen Arakelov, Gurgen Arakelov, Philipp Binfet, Victor Correa, Jules Dumezy, Sergey Gomenyuk, Valentina Kononova, Dmitrii Lekomtsev, Vivian Maloney, Chi-Hieu Nguyen, Yuriy Polyakov, Daria Pianykh, Hayim Shaul, Moritz Schulze Darup, Dieter Teichrib, and Dmitry Tronin.
Fherma cookbook: Fhe components for privacy-preserving applications.
In *Proceedings of the 13th Workshop on Encrypted Computing & Applied Homomorphic Computing*, WAHC '25, page 68–76, New York, NY, USA, 2025. Association for Computing Machinery.

Andreea Alexandru, Andrey Kim, and Yuriy Polyakov.
General functional bootstrapping using ckks.
In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025*, pages 304–337, Cham, 2024. Springer Nature Switzerland.

References II

- Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto.
  Bootstrapping small integers with ckks.
  In *Advances in Cryptology – ASIACRYPT 2024*, pages 330–360. Springer, 2024.
- Jules Dumezy, Andreea Alexandru, Yuriy Polyakov, Pierre-Emmanuel Clet, Olive Chakraborty, and Aymen Boudguiga.
  Evaluating larger lookup tables using CKKS.
  To appear in CHES '26, 2025.
- Ilia Iliashenko and Vincent Zucca.
  Faster homomorphic comparison operations for bgv and bfv.
  Proceedings on Privacy Enhancing Technologies, 2021(3):246–264, 2021.