Liberate.FHE: A New FHE Library for Bridging the Gap between Theory and Practice with a Focus on Performance and Accuracy

Juwhan Kim Hanyul Ryu Donghoon Yoo

DESILO Inc., Seoul, Republic of Korea

Abstract

A new implementation of the Fully Homomorphic Encryption (FHE) library, namely Liberate.FHE is presented. Its primary focus is performance and accuracy. Despite significant advancements in previous works, the adoption of modern computing resources, such as GPU acceleration, has been sparse, and the accuracy of the homomorphic computations could have been better than the theoretical predictions. This work addresses the sub-optimal issues by filling in the missing details regarding practical engineering. Based on the newly added formulations, a new FHE library is built. Liberate.FHE supports multi-GPU operations natively, and its Application Programming Interface (API) is designed to be simple by employing the most widely used presets. The main idea behind the design decisions is that non-cryptographers can use the library; it should be easily hackable and integrated with more extensive software frameworks. The resulting implementation can be accessed at https://github.com/Desilo/liberate-fhe [1].

Major Algorithms in Liberate.FHE

New Channel Prime Selection Method

Liberate.FHE introduces an error-free rescaling method through a novel channel prime selection approach, leveraging the division by channel primes to mitigate rescaling errors inherent in RNS-based FHE systems which use a scale parameter Δ and a modulus $Q = \prod_{i=0}^{L-1} q_i$. The proposed method, detailed in Equation 1, optimizes the selection of channel primes to minimize quadratic deviation errors, crucial for maintaining the accuracy of homomorphic operations with $\delta_0 = 1$. It employs a strategic search for channel primes that support the Number Theoretic Transform (NTT), adjusting the direction of the search to balance cumulative deviations. This precise prime selection, combined with a pre-scaling strategy that adjusts cipher texts before multiplication, significantly enhances the accuracy and efficiency of Liberate.FHE's cryptographic computations.

$$\delta_i = \Delta^2 \delta_{i-1}^2 / q_{i-1}^2 = \delta_{i-1}^2 \left(\frac{\Delta}{q_{i-1}}\right)^2 \tag{1}$$

Encoding/Decoding using Nega-cyclic FFT

Our packing method innovatively encodes complex numbers into plaintext for encryption, ensuring nega-cyclic behavior for stable array sizes, compatibility with homomorphic operations, and conversion to real numbers, with a focus on minimal accuracy loss. It employs a novel nega-cyclic FFT, transforming complex arrays for encryption within a finite quotient ring, maintaining permutation commutativity. This approach uses a calculated twister to match polynomial coefficients, integrating smoothly into cryptographic algorithms without extra scaling, simplifying encryption while maximizing precision and utility.

Division by the Channel Modulus and Reduction

In the context of Fully Homomorphic Encryption (FHE), a novel method for precise division by a channel prime is introduced, addressing the accuracy issues arising from the approximate division method traditionally used during modulus switching. This method involves simultaneous division and reduction, optimizing the Residue Number System (RNS) based computations by leveraging an exact process along with a necessary rounding technique. This advancement ensures the integrity of the encrypted data's accuracy throughout the computation process, significantly enhancing the practical utility of FHE schemes like CKKS in real-world applications.

$$x_i' = \left| (x_{i+1} - x_0) \left| q_0 \right|_{i+1}^{-1} \right|_{q_{i+1}} \tag{2}$$

Key Switching

Liberate.FHE utilizes a unique key switching approach, inspired by a hybrid method [2, 3], focusing on direct operations on the RNS representation rather than applying the conventional bitwise decomposition technique. This method involves transforming two ciphertexts, ct_1 and ct_2, through a process where ct_1 is switched from secret key sk_1 to sk_1, with ct_2 containing the key switching key (ksk). The transformation is designed to minimize error terms, leveraging the RNS representation for efficient computation. This direct approach simplifies the key switching process, making it more efficient while ensuring the integrity of the encrypted message.

RNS Basis Extension

The decomposition approach for optimizing RNS representation values involves a three-part process

- Initially focusing on generating a cache of fixed parameters for efficiency.
- Followed by the calculation of a state vector using predetermined values.
- Concluding with a basis extension to integrate unknown values.

This method leverages the 'pow' function for integer exponentiation within $\mathbb{Z}_{q_i+1}$, demonstrating its applicability beyond the initial partition to encompass arbitrary fixed partitions, streamlining computations in cryptographic operations.

Liberate.FHE Design Philosophy

Liberate.FHE attempts to straighten up the computations by employing explicit formulae. In other words, formulations in Liberate.FHE is implementation-centric and avoids abstract constructions. Additionally, several design decisions were made to maximize the usability of the developed software:

- Make the number of dependencies minimal.
- Make the software easily hackable.
- Set the usage of multiple GPUs as the default.
- Make the resulting library easily integrated with the pre-existing software, especially the Artificial Intelligence (AI) related ones.

The above decisions led to building the library on top of the popular software package PyTorch. In Liberate.FHE, all the keys and cipher texts are a collection of PyTorch tensors. In that way, the inner structures of the data and even the contents stored in them can be viewed and hacked easily. Also, the very low-level functions such as the Montgomery reduction and the related operations, Number Theoretic Transform (NTT), and Cryptographically Secure Pseudo Random Number Generators (CSPRNG) are built using the NVIDIA CUDA programming language. The Low-level functions are then attached to the higher-level library functions using the C-API provided by PyTorch.

Practical Advancements

Data Partitioning over Multiple GPUs

Liberate.FHE optimizes Ring-LWE cipher text processing across multi-GPU setups, utilizing a tensor chunking strategy that minimizes data movement, balances computational load, and maintains decryption integrity through a tailored tensor shape for efficient, high-performance homomorphic encryption. The chunking strategy is designed to satisfy the following conditions:

- A key switching partition always resides in the same device.
- The base channel is put into the master GPU.
- Special prime channels are replicated in all GPUs.
- The partitions are placed in GPUs as exhaustion of one partition will start the consumption of the next partition at a different GPU.

Exact Rounding in Rescaling

Our study highlights that the accuracy of rescaling results in FHE is significantly influenced by the rounding method employed. By introducing a rounding parameter, $\rho = x_0/q_0$, and incorporating it into the equation 3 . we achieve a more precise solution, enhancing computational accuracy substantially.

$$\left(x_0^\dagger, \ldots x_{L-2}^\dagger\right) \triangleq \lfloor \frac{X \mod Q}{q_0} \equiv X' \mod Q' \rceil \tag{3}$$

Montgomery Reduction-based Integer Arithmetics

Liberate.FHE enhances FHE multiplication with an optimized Montgomery reduction, tailored for GPUs by using bitwise operations to perform simultaneous multiplication and reduction without exceeding 64-bit sizes or wasting memory, significantly increasing computational efficiency.

Cryptographically Secure Pseudo Random Number Generator (CSPRNG)

The new CSPRNG for Liberated.FHE, designed for enhanced security and efficiency in homomorphic encryption, features:

- CHACHA20[4] algorithm base for cryptographic robustness.
- GPU parallelization via index-based generation for high performance [5].
- Branchless binary tree search, optimizing GPU operations by avoiding conditional logic.

This streamlined approach addresses security and computational speed, crucial for encryption/decryption tasks.

Multiparty

Liberate.FHE advances MFHE with an 'n of n' consent model, merging distributed trust and privacy [6]. It offers single-key efficiency and dynamic participant adaptability via individual secret keys and a shared public key, supported by user-friendly APIs for collaborative, real-world applications.

```python
from liberate import fhe
from liberate.fhe import presets

# generate ckks engine
grade = "gold"
params = presets.params[grade]
engine = fhe.ckks_engine(**params)

# generate keys
sks = [engine.create_secret_key()
       for _ in range(num_of_parties)]
pks = [engine.create_public_key(sks[0])]
crs = pks[0].data[1]
for sk in sks[1:]:
 pks.append(engine.multiparty_create_public_key(sk, crs))
cpk = engine.multiparty__collective_public_key(pks=pks)
```

```python
# generate test data
m0 = engine.example(-1, 1)
m1 = engine.example(-10, 10)

# encode & encrypt data
ct0 = engine.encorypt(m0, pk)
ct1 = engine.encorypt(m1, pk)

result = (m0 + m1)  - m0
ct_add = engine.add(ct0, ct1)
ct_result = engine.sub(add, ct0)
# decrypt & decode data
pcts = [engine.multiparty_decrypt_head(ct, sks[0])]
for sk in sks[1:]:
 pcts.append(engine.multiparty_decrypt_partial(pcts))
m_ = engine.multiparty_decrypt_fusion(pcts)
```

Results and Example Codes

Results

Liberate.FHE sets a new benchmark in FHE with key innovations for unparalleled accuracy and speed. It leverages integer-only operations via Montgomery Reduction, avoiding floating-point errors, and optimizes data handling on GPUs for rapid key switching. Its novel 'Rescale Before Multiplication' strategy, alongside message permutation, guarantees cyclic rotation and integrity, with unique formulas mitigating rescale and deviation errors, vastly improving accuracy in homomorphic operations.

Table 1. Liberate.FHE Benchmark* (ms)

Operation		Liberate.FHE
Key Creation	Secret Key	0.330
	Public Key	0.452
	Evaluation Key	5.371
	Galois Key	90.205
HE Operations	Addition	0.074
	Multiplication	11.30
	Rotation	6.110
	Mean	94.900
	Variation	188.000
Round Trip	Encode+Encrypt	1.601
	Decrypt+Decode	1.051

* $logN$=16, scale bits=40, special primes=4, multiplicative levels=34, GPUs=2 (NVIDIA A100)

Example Codes

```python
from liberate import fhe
from liberate.fhe import presets

# generate ckks engine
grade = "gold"
params = presets.params[grade]
engine = fhe.ckks_engine(**params)

# generate Keys
sk = engine.create_secret_key()
pk = engine.create_public_key(sk)
evk = engine.create_evk(sk)

# generate test data
m0 = engine.example(-1, 1)
m1 = engine.example(-10, 10)

# encode & encrypt data
ct0 = engine.encorypt(m0, pk)
ct1 = engine.encorypt(m1, pk)

# (a + b) * b - a
result = (m0 + m1) * m1 - m0
ct_add = engine.add(ct0, ct1)
ct_mult = engine.mult(ct0, ct_add, evk)
ct_result = engine.sub(ct_mult, ct0)

# decrypt & decode data
decrypted = engine.decrode(ct_result, sk)
```

Conclusions

Liberate.FHE bridges the gap between theoretical FHE and practical implementation by prioritizing performance, accuracy, and user-friendliness. Its novel design leverages modern computing resources like GPUs and simplifies API usage for non-cryptographers, encouraging wider adoption and real-world applications of FHE. Further research directions include investigating bootstrapping on multiple GPUs, exploring scheme switching between FHE schemes, and developing flexible and scalable multiparty FHE protocols on Liberate.FHE.

Features in Delivery

- CKKS bootstrapping
- Multiparty bootstrapping
- Liberate.FHE CPU version
- IND-CPA-D security for CKKS

References

[1] DESILO, "Liberate.fhe: A new fhe library for bridging the gap between theory and practice with a focus on performance and accuracy," 2023. https://github.com/Desilo/liberate-fhe.

[2] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption." Cryptology ePrint Archive, Paper 2019/688, 2019. https://eprint.iacr.org/2019/688.

[3] J.-C. Bajard, J. Eynard, A. Hasan, and V. Zucca, "A full rns variant of fv like somewhat homomorphic encryption schemes." Cryptology ePrint Archive, Paper 2016/510, 2016. https://eprint.iacr.org/2016/510.

[4] D. Bernstein, "Chacha, a variant of salsa20," 01 2008.

[5] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw, "Parallel random numbers: As easy as 1, 2, 3," in *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2011.

[6] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, "Multiparty homomorphic encryption from ring-learning-with-errors," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. CONF, pp. 291–311, 2021.

Libertate.FHE Documentation Website

Scan this QR code