High-Performance Privacy-Preserving Blockchain via GPU-Accelerated Fully Homomorphic Encryption

Anonymous Authors

Anonymous Institute

Abstract. Data privacy is essential for safety-critical blockchain applications such as digital payment. A promising approach for preserving privacy is to encrypt transaction data with homomorphic encryption (HE) and prove the correctness of transaction execution through non-interactive zero-knowledge proofs. However, previous works on this approach are restricted by the use of HE schemes that support either addition or multiplication, preventing them from implementing business logic involving both types of arithmetic operations. Additionally, previous works are plagued by the compute-intensive HE computation, leading to poor performance.

We present GAFE[1], an efficient privacy-preserving blockchain that is co-designed with GPU-accelerated fully homomorphic encryption (FHE). GAFE processes and stores transaction data that are encrypted with FHE, which allows both addition and multiplication on ciphertexts. Moreover, GAFE leverages GPU to accelerate FHE computation, which enhances the overall throughput and reduces the commit latency. We implemented GAFE on top of Hyperledger Fabric. Our evaluations show that GAFE is highly efficient, with a 3.1× increase in throughput (258 transactions per second) and a 37% decrease in latency (1.61 seconds), surpassing the baseline.

Keywords: Blockchain · Privacy Preserving · Fully Homomorphic Encryption · GPU Acceleration

1 Introduction

Blockchain has emerged as a transformative force across various industries, such as finance [25] and healthcare [19] Its decentralized nature, immutability, and transparency make it an appealing solution for ensuring trust and security in transaction data. However, notable blockchains such as Bitcoin [23] and Hyperledger Fabric [2] process and store transaction data in the form of plaintext, which raises deep concerns about the privacy of transaction data. While blockchain provides protection against unauthorized modifications, the transparency of the ledger means that anyone with access to the network can view

[1] GAFE stands for GPU-Accelerated Fully Homomorphic Encryption Blockchain.

the sensitive information stored in plaintext. This poses a significant challenge, particularly in safety-critical applications where the privacy of transactional data is crucial. The exposure of plaintext data on blockchains can lead to privacy breaches, identity theft, and financial fraud [27]. Therefore, addressing blockchain's privacy issue is important for promoting its adoption in safety-critical applications and safeguarding the privacy of sensitive information.

A promising approach to addressing the privacy issue is leveraging cryptographic primitives such as homomorphic encryption [1] (HE) and non-interactive zero-knowledge proofs [32] (NIZKPs). HE enables computation to be performed directly on ciphertexts, eliminating the need to decrypt the underlying plaintexts. NIZKPs allow a prover to prove to a verifier that a statement is true without revealing the involving private inputs. Following this approach, transaction data are first encrypted with a specific HE scheme. Subsequently, the transaction is executed directly on the ciphertext. Finally, NIZKPs are generated to prove the correctness of the transaction execution, without revealing the plaintext of the execution result.

However, previous works on this approach face two non-trivial challenges. Firstly, some works [31,30,34] are limited in their ability to express complex business logic. This limitation arises from their use of partly homomorphic encryption (PHE) to encrypt transaction data, which only supports a single type of arithmetic operation (i.e., either addition or multiplication). Consequently, these works are not applicable to applications that involve both addition and multiplication [25]. Secondly, previous works suffers from intensive HE computation overhead. For example, Microsoft SEAL [28], a highly optimized HE library, takes hundreds of milliseconds to perform a HE multiplication [22]. As a result, these works fail to meet the high-performance requirements expected in many blockchain applications like digital payment.

In this paper, we present Gafe, an efficient privacy-preserving blockchain system. Gafe tackles the challenges of the aforementioned approach based on our key insight: a considerable portion of the arithmetic computations involved in fully homomorphic encryption (FHE) schemes can be performed concurrently. This concurrent computation is well-suited for GPUs, which possess superior parallel processing capabilities. Therefore, we empower Gafe with GPU-accelerated FHE, which brings two benefits. Firstly, FHE enables Gafe to preserve the privacy of transaction data such as the amount of digital payment, as well as implement business logic involving both addition and multiplication. Secondly, GPU acceleration significantly enhances Gafe's performance, making Gafe highly suitable for blockchain applications that demand both privacy preservation and high performance.

In the rest of the paper: §2 discusses the related work of Gafe; §3 gives an overview of Gafe; §4 delves into the details of Gafe's workflow; §5 shows our evaluation, and §6 concludes.

System	FHE Support	GPU Acceleration	High Performance
❖ ZeeStar [30]	✗	✗	✗
❖ SmartFHE [29]	✓	✗	✗
❖ Ekiden [9]	✗	✗	✓
❖ Hawk [18]	✗	✗	✓
❖ Arbitrum [16]	✗	✗	✓
◆ Zcash [14]	✗	✗	✗
◆ Monero [21]	✗	✗	✗
◆ FabZK [17]	✗	✗	✓
◆ Zether [7]	✗	✗	✗
◆ RFPB [34]	✗	✗	✓
◆ **Gafe**	✓	✓	✓

Table 1: Comparison of Gafe and related privacy-preserving blockchains. Symbols ❖ and ◆ represent general-purpose blockchains and specific-purpose blockchains, respectively.

2 Related work

2.1 Privacy-Preserving Blockchain

Privacy-preserving blockchains have gained significant attention in both academia and industry. As shown in Table 1, we summarize previous works into two categories:

General-purpose blockchains. ZeeStar [30] uses ElGamal encryption [20], which only supports addition. Although ZeeStar extends addition to support multiplication, this extension is inefficient and cannot be applied to ciphertexts encrypted by different keys. SmartFHE [29] employs FHE schemes, but it does not support computation on ciphertexts encrypted by different keys. Ekiden [9] demands hardware that supports trusted execution environments to execute transactions over private data off the blockchain, which may not always be available for commodity desktop machines. Hawk [18] and Arbitrum [16] delegate trusted managers for transaction execution, but this introduces a substantial attack surface for malicious managers to expose private inputs.

Specific-purpose blockchains. In order to preserve the privacy of digital payment information (e.g., payment amount), Zcash [14] employs NIZKPs, while Monero [21] utilizes ring signatures and stealth addresses. However, their low throughput and long latency have hindered broader adoption [26,7]. FabZK [17] combines NIZKPs with a specialized tabular ledger data structure to preserve data privacy, but this data structure limits FabZK to performing only addition. Zether [7] and RFPB [34][2] adopt PHE schemes, posing the same limitation as ZeeStar: they only support on-ciphertext addition.

[2] We refer to the system proposed in [34] as RFPB for convenience.

2.2 GPU-Accelerated Fully Homomorphic Encryption

Many works have been proposed to leverage GPU acceleration for FHE in order to unlock the potential of FHE in practical applications. HE-Booster [33] achieves acceleration of polynomial arithmetic computation by mapping five common phases of typical FHE schemes to the GPU parallel architecture. HE-Booster also introduces thread-level and data-level parallelism to enable acceleration on single-GPU and multi-GPU setups, respectively. Ozcan et al. [24] presents a library that makes efficient use of the GPU memory hierarchy and minimizes the number of GPU kernel function calls. Yang et al. [35] provides GPU implementations of three notable FHE schemes (BGV [6], BFV [5,12], and CKKS [10]) along with various theoretical and engineering optimizations, including a hybrid key-switching technique and several kernel fusing strategies. Note that GAFE is independent of specific implementations of GPU-accelerated FHE schemes.

3 Overview

3.1 System Model

GAFE comprises three types of participants: *client*, *executor*, and *orderer*. Executors and orderers are commonly referred to as *nodes*. GAFE uses permissioned settings, where all participants are organized into distinct *organizations*. Each organization runs multiple executors and orderers, and also possesses a set of clients. We built GAFE on top of Hyperledger Fabric [2] and developed a prototype system implementing the business logic of digital payment. Specifically, the role of each participant type is described as follows:

Client. Clients encrypt transaction data and submit the encrypted transactions to the nodes for execution and commitment. Each client is identified by a unique fixed-length string *id* and possesses a randomly generated a public-private key set that is used for performing FHE operations. Additionally, each client is associated with a floating-point number *bal*, which represents the balance owned by the client and is encrypted using the FHE scheme.

Executor. Each executor is responsible for three main tasks: (1) executing encrypted transactions, (2) validating transaction results, and (3) maintaining the latest local copy of blockchain and state database. Specifically, each executor is equipped with a GPU and employs GPU acceleration for FHE computation during transaction execution. We provide a detailed discussion of GAFE's transaction execution workflow in §4.2.

Orderer. GAFE orderers determines the order of transactions within each block. GAFE orderers leverage the orderer implementation of Hyperledger Fabric, therefore inheriting the mature security guarantees. Specifically, GAFE orderers run an instance of the BFT-SMaRT [4] protocol, a fast Byzantine-Fault-Tolerant consensus protocol. This eliminates the performance bottleneck associated with time-consuming consensus protocols like Proof-of-Work [23].

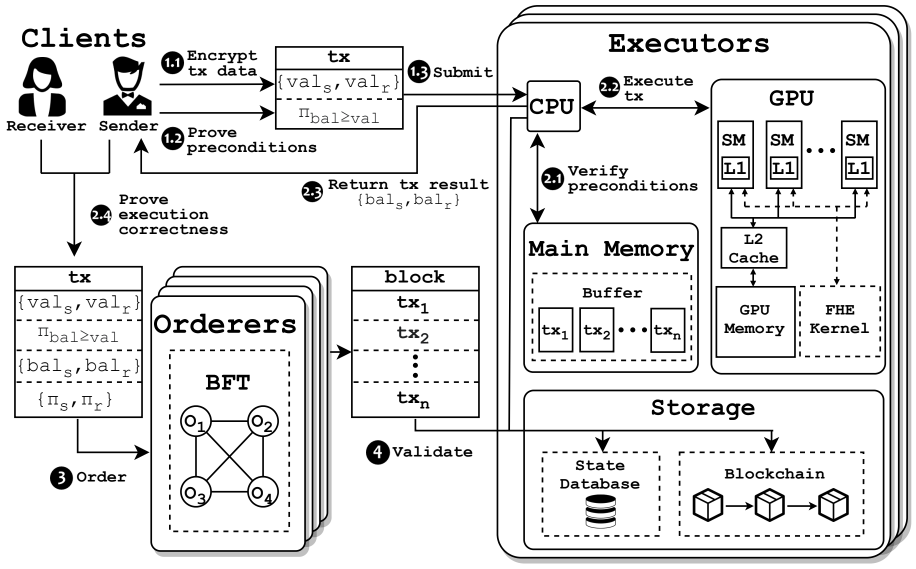


Fig. 1: GAFE's GPU-accelerated transaction execution workflow. Each executor utilizes multiple Streaming Multiprocessors (SM) of a GPU to concurrently execute FHE computation for multiple transactions.

3.2 Threat Model

GAFE adopts the Byzantine failure model [3,8], which tolerates up to N malicious orderers out of a total of $3N + 1$ orderers. Same as Hyperledger Fabric, participants within GAFE trust all participants within their own organization but do not trust participants from other organizations. We make standard assumptions on FHE and NIZKP.

3.3 GAFE's Workflow Overview

GAFE's workflow consists of two sub-workflows: *client key generation* (§4.1) and *GPU-accelerated transaction execution* (§4.2). Specifically, the online transaction execution workflow is divided into four phases, as illustrated in Figure 1.

Phase 1: Construction. The client constructs a transaction tx by encrypting the transaction data (e.g., payment amount val) using the client's encryption key. The client also generates an NIZKP $\pi_{bal \geq val}$ to demonstrate that the client's balance bal is greater than or equal to val. Finally, the client submits the encrypted transaction and $\pi_{bal \geq val}$ to all executors for execution.

Phase 2: Execution: The executor verifies $\pi_{bal \geq val}$ and proceeds with the remaining procedure only if $\pi_{bal \geq val}$ is valid. Next, the executor buffers tx along with other transactions received within a specific time frame. These buffered transactions are delivered to a GPU for performing concurrent FHE computations, and their execution results are returned to the corresponding clients. Upon receiving the results from all executors, the client generates correctness NIZKPs to prove the correctness of transaction execution. Finally, the client sends both the consistent result and the correctness NIZKPs to the orderers.

Phase 3: Ordering. All orderers run a BFT consensus protocol to determine the order of transactions within each block. Once the order is determined, the orderers deliver the block to the executors for validation.

Phase 4: Validation. On receiving a block from the orderers, the executor sequentially validates each transaction within the block following the determined order. The executor commits a transaction only if the transaction has valid correctness NIZKPs and has no write conflict with previous commited transactions within the same block. Otherwise, the executor aborts the transaction.

4 Workflow Description

4.1 Client Key Generation

In GAFE, each client possesses a unique public-private key set for FHE operations. This key set consists of (1) an encryption key pk, (2) a decryption key sk, and (3) an evaluation key ek used for on-ciphertext arithmetic evaluation. While the encryption key pk and the evaluation key ek are accessible to all participants in the GAFE network, the decryption key sk must remain private to the client.

Upon the creation of a client, GAFE runs an instance of the key generation algorithm associated with the chosen FHE scheme (e.g., CKKS [10]). This algorithm initially generates the encryption key sk, and then derives the decryption key pk and the evaluation key ek from sk. Additionally, GAFE generates a unique fixed-length string id based on sk to serve as the client's unique identifier.

4.2 GPU-Accelerated Transaction Execution

GAFE's *GPU-accelerated transaction execution* workflow co-designs the execute-order-validate workflow of Hyperledger Fabric with GPU-accelerated FHE scheme. This workflow consists of four consecutive phases, as outlined in Figure 1 and Algorithm 1.

Phase 1: Construction. For privacy preservation, GAFE processes and stores transaction data in the form of ciphertext. Consequently, the client is required to construct an encrypted transaction and submit it to GAFE nodes for execution.

Phase 1.1: Encrypting transaction data. The client encrypts transaction data using the encryption keys of the clients involved. In the context of digital payment, a transaction involves a sender client c_s and a receiver client c_r. Therefore, c_s is responsible for encrypting the payment amount val into two ciphertexts val_s and val_r using c_s's and c_r's encryption keys, respectively. In the execution phase, executors use val_s and val_r to perform arithmetic evaluation. The reason for generating two different ciphertexts for val is that GAFE employs single-key FHE, which allows computation on two ciphertexts only if they are encrypted using the same encryption key. GAFE does not use multi-key FHE, which supports

Algorithm 1: Transaction execution workflow of the client.

```
   input: Plaintext transaction data val
   // Phase 1: Construction
1  $\mathtt{pk_s} \leftarrow$ GetEncryptionKey($\mathtt{id_s}$); $\mathtt{val_s} \leftarrow$ Encrypt(val,$\mathtt{pk_s}$);
2  $\mathtt{pk_r} \leftarrow$ GetEncryptionKey($\mathtt{id_r}$); $\mathtt{val_r} \leftarrow$ Encrypt(val,$\mathtt{pk_r}$);
3  $\mathtt{bal_s} \leftarrow$ GetBalance($\mathtt{id_s}$);
4  $\pi_{\mathtt{bal} \geq \mathtt{val}} \leftarrow$ GenerateNIZKP($\mathtt{bal_s} \geq \mathtt{val_s}$, $\mathtt{sk_s}$);
5  $\mathtt{tx} \leftarrow \{\mathtt{val_s}, \mathtt{val_r}, \pi_{\mathtt{bal} \geq \mathtt{val}}\}$;
   // Phase 2: Execution
6  $\mathtt{results} \leftarrow \varnothing$;
7  For every executor e; do in parallel
8      $\mathtt{bal'_s}, \mathtt{bal'_r} \leftarrow$ SendForExecution(e,tx);
9      $\mathtt{results} \leftarrow \mathtt{results} \cup \{\mathtt{bal'_s}, \mathtt{bal'_r}\}$;
10 $\pi_{\mathtt{s}} \leftarrow$ GenerateNIZKP(a majority of $\mathtt{bal'_s}$ in results are consistent);
11 $\pi_{\mathtt{r}} \leftarrow$ GenerateNIZKP(a majority of $\mathtt{bal'_r}$ in results are consistent);
   // Phase 3: Ordering & Phase 4: Validation
12 $\mathtt{tx} \leftarrow \mathtt{tx} \cup \{\mathtt{bal'_s}, \mathtt{bal'_r}, \pi_{\mathtt{s}}, \pi_{\mathtt{r}}\}$;
13 SendForOrderingAndValidation(tx);
```

computation on ciphertexts encrypted with different encryption keys, because multi-key FHE is currently prohibitively expensive and impractical for real-world applications [36].

Phase 1.2: Proving preconditions. In certain applications, the client generates precondition NIZKPs to prove the satisfaction of preconditions. The correct transaction execution of these applications is conditional upon on these conditions. For instance, digital payment demands that the sender client's balance bal_s must be greater than or equal to the payment amount val. To prove this condition, c_s generates an NIZKP $\pi_{bal \geq val}$ that takes c_s's decryption key as private input, decrypts the two ciphertexts bal_s and val_s, and compares the resulting plaintexts. Thanks to the zero-knowledge properties of NIZKPs, Gafe ensures the preservation of privacy for c_s's decryption key and the two plaintexts.

Phase 1.3: Submitting transactions. As the final step of the construction phase, the client submits the transaction, including the encrypted data and precondition NIZKPs, to all executors for execution.

Phase 2: Execution. As shown in Figure 1, the execution phase comprises four sub-phases. Algorithm 2 outlines Phase 2.1, 2.2, and 2.3 from the executor's viewpoint.

Phase 2.1: Verifying preconditions. Upon receiving a transaction from a client, the executor initially verifies the validity of the precondition NIZKPs associated with the transaction, such as $\pi_{bal \geq val}$. Gafe proceeds to the subsequent phases iff $\pi_{bal \geq val}$ is deemed valid. Otherwise, Gafe immediately terminates the transaction execution.

Algorithm 2: Execution phase of the executor.

```
1  buffer $\leftarrow \varnothing$;
   // Phase 2.1: Verifying preconditions
2  Upon reception of transaction tx from client; do
3      $\{\mathtt{val_s}, \mathtt{val_r}, \pi_{\mathtt{bal} \geq \mathtt{val}}\} \leftarrow$ tx;
4      $\mathtt{bal_s} \leftarrow$ GetBalance($\mathtt{id_s}$); $\mathtt{ek_s} \leftarrow$ GetEvaluationKey($\mathtt{id_s}$);
5      $\mathtt{bal_r} \leftarrow$ GetBalance($\mathtt{id_r}$); $\mathtt{ek_r} \leftarrow$ GetEvaluationKey($\mathtt{id_r}$);
6      if VerifyNIZKP($\pi_{\mathtt{bal} \geq \mathtt{val}}$,$\mathtt{bal_s}$,$\mathtt{val_s}$) $= \mathit{true}$ then
7          buffer $\leftarrow$ buffer $\cup \{\mathtt{val_s}, \mathtt{val_r}, \mathtt{bal_s}, \mathtt{bal_r}, \mathtt{ek_s}, \mathtt{ek_r}\}$;
8      else
9          Terminate();
   // Phase 2.2: Executing transactions with GPU-accelerated FHE
10 Upon timeout or |buffer| $\geq$ bufferSize
11     MoveFromMainMeoryToGPUMemory(buffer);
12     For every transaction tx in buffer; do in parallel on GPU
13         $\mathtt{bal}'_\mathtt{s} \leftarrow$ FHESub($\mathtt{bal_s}$,$\mathtt{val_s}$,$\mathtt{ek_s}$);
14         $\mathtt{bal}'_\mathtt{r} \leftarrow$ FHEAdd($\mathtt{bal_r}$,$\mathtt{val_r}$,$\mathtt{ek_r}$);
15         buffer $\leftarrow$ buffer $\cup \{\mathtt{tx}, \mathtt{bal}'_\mathtt{s}, \mathtt{bal}'_\mathtt{r}\}$;
16     MoveFromGPUMeoryToMainMemory(buffer);
17     buffer $\leftarrow \varnothing$;
   // Phase 2.3: Returning transaction results
18 For every transaction tx in buffer; do in parallel
19     ReturnResultToClient(tx,$\mathtt{id_s}$)
```

Phase 2.2: Executing transactions with GPU-accelerated FHE. Similar to Hyperledger Fabric, each Gafe executor independently executes transaction. For the on-GPU FHE computation, the executor performs concurrency control on the transaction level. Internally, the executor maintains a buffer that temporarily store the transactions received within a predefined time frame (e.g., 500 milliseconds). When a timeout occurs or the number of buffered transactions exceeds a specific threshold, the executor performs the following three steps: (1) moving the encrypted data of all buffered transactions from main memory to GPU memory, (2) launching the corresponding GPU kernel of FHE computation, and (3) copying back the resulting ciphertexts back to main memory. Taking the example of digital payment, the executor first moves the following data to GPU memory: the two ciphertexts representing the payment amount $\{val_s, val_r\}$, and the balances of the sender and the receiver clients $\{bal_s, bal_r\}$. Next, the executor launches the GPU kernel for FHE addition to compute the updated balances: $bal'_s = bal_s - val_s$ for the sender, and $bal'_r = bal_r - val_r$ for the receiver. After completing the FHE computation, the executor copies back the updated balances $\{bal'_s, bal'_r\}$ to main memory.

Phase 2.3: Returning transaction results. After the completion of transaction execution, the executor returns the transaction results to the client who initiated the transaction. In the case of digital payment transactions, the executor returns the updated balances bal'_s, bal'_r to the sender client c_s.

Phase 2.4: Proving execution correctness. Upon receiving the results of a transaction from all executors, the involving clients of the transaction are responsible for proving the execution correctness. To achieve this, the involving clients generates NIZKPs to prove the consistency of the majority of the results. In the case of digital payment, the sender c_s generates an NIZKP π_s that takes c_s's decryption key as private input, decrypts c_s's updated balance ciphertexts from all the results, and checks the consistency of the resulting plaintexts. Similarly, to ensure the consistency of the receiver client c_r's updated balance, c_r follows the same procedure and generates an NIZKP π_r using c_r's decryption key. Lastly, c_s sends the necessary materials to the orderers for ordering, including the transaction data $\{val_s, val_r\}$, the precondition NIZKP $\pi_{bal \geq val}$, the consistent results $\{bal_s, bal_r\}$, and the correctness NIZKPs $\{\pi_s, \pi_r\}$.

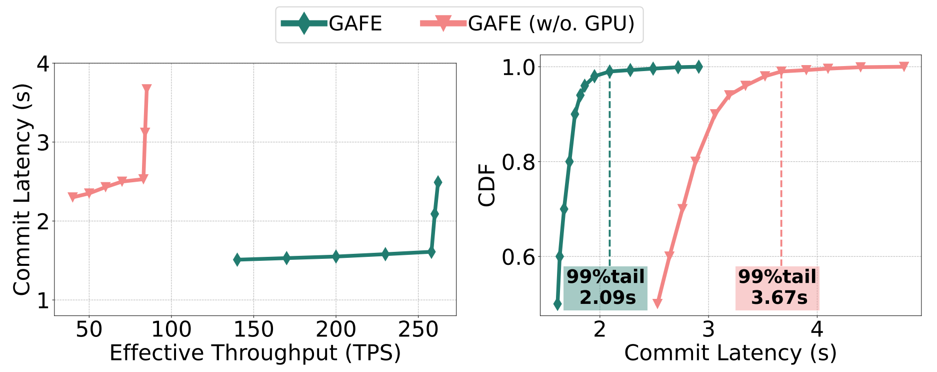
Phase 3: Ordering. GAFE orderers run an instance of a BFT consensus protocol, such as BFT-SMaRT [4]), to collectively determine the transaction order within each block. Once a consensus is reached among all orderers, they proceed to generate the block and disseminate the block to all executors for validation.

Phase 4: Validation. Upon receiving a block from the orderers, the executor follows the pre-determined order to sequentially validate each transaction within the block. The executor commits only those transactions that meet two conditions. First, the transaction must not have any write-conflicts with previously committed transactions within the same block. This prevents conflicts between concurrent transactions. Second, the transaction must be accompanied by valid correctness NIZKPs $\{\pi_s, \pi_r\}$. These NIZKPs serve as proofs of the transaction's execution correctness. If a transaction fails to meet either of these conditions, the executor aborts the transaction. Once the executor has validated all transactions, the executor permanently appends the block to the local copy of the blockchain.

5 Evaluation

5.1 Settings

Implementation. We built a prototype system of GAFE based on Hyperledger Fabric [2] and simulates the scenario of digital payment. We implemented the CKKS scheme [10] for GAFE based on the state-of-the-art studies on GPU-accelerated FHE schemes [24,33,35]. GAFE adopted the gnark [11] library's implementation for the Groth16 NIZKP system [13] and employed the BFT-SMaRT [4] consensus protocol. To evaluate the performance gain of GAFE, we also developed a baseline system called GAFE (W/O. GPU). The baseline follows a similar transaction execution workflow as GAFE, except that the baseline does not buffer transactions for concurrent execution and performs all FHE computations exclusively on the CPU.



(a) Effective throughput and commit latency.

(b) The cumulative distribution function of commit latency.

Fig. 2: End-to-end performance of Gafe and the baseline.

Metrics. We evaluated two metrics: (1) *effective throughput*, which represents the average number of client transactions per seconds (TPS) committed to the blockchain; and (2) *commit latency*, which measures the time duration from transaction construction to commitment. Additionally, we also reported the cumulative distribution function of commit latency for all committed transactions, and the latency of each phase in the transaction execution workflow (§4.2).
Testbed. We ran all evaluations on a cluster consisting of 4 machines. Each machine was equipped with an Nvidia RTX 3090 GPU, a 3.1GHz AMD EPYC 9754 CPU, and 64GB of main memory.

5.2 End-to-End Performance

We evaluated the end-to-end performance of Gafe and Gafe (w/o. GPU). For each evaluation, we initially instantiated three executors, four orderers, and one thousand clients. Next, we construct and submit 100,000 digital payment transactions using Tape [15], an efficient benchmark tool for blockchain. We explicitly ensured that no two transactions shared identical involving clients in order to prevent transaction aborts caused by write conflicts. We ran the evaluation ten times and reported the average values of the metrics.

Gafe exhibited exceptional end-to-end performance, as shown in Figure 2a. Gafe achieved a high throughput of 258 TPS and a low average latency of 1.61 seconds. In contrast, Gafe (w/o. GPU) displayed a significantly lower average throughput of 83 TPS and a notably longer average latency of 2.53 seconds. These results highlight the substantial performance advantage of Gafe over Gafe (w/o. GPU), with a 3.1× increase in effective throughput and a 37% reduction in commit latency. Additionally, Figure 2b illustrates that Gafe achieved a shorter 99% tail latency (2.09 seconds) compared to the baseline (3.67

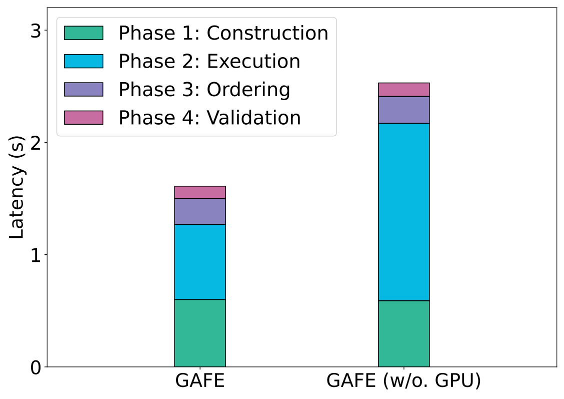


Fig. 3: Latency of each phase on Gafe and the baseline.

seconds). This suggests that, even in worst-case scenarios, Gafe is capable of completing transaction execution in significantly less time.

Gafe's high performance is attributed to the concurrent execution of FHE computation on GPU. Figure 3 compares the latency of each phase between Gafe and the baseline. Note that Gafe achieved significantly lower latency in the execution phase while achieving similar latencies in other phases. This latency reduction is made possible by the optimized parallel processing capability of the GPU, allowing for the concurrent execution of a substantial portion of the arithmetic computations involved in typical FHE schemes. As a result, Gafe avoids performing FHE computation on the CPU, which has fewer cores and is less efficient in executing a large number of compute-intensive computations in parallel. This capability enables Gafe to achieve higher effective throughput and shorter commit latency.

6 Conclusion

TODO

References

1. Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys (Csur) **51**(4), 1–35 (2018)
2. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the thirteenth EuroSys conference. pp. 1–15. ACM, New York, NY, USA (2018)
3. Bessani, A., Sousa, J., Alchieri, E.E.: State machine replication for the masses with bft-smart. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 355–362. IEEE, IEEE Computer Society, 1730 Massachusetts Ave., NW Washington, DCUnited States (2014)

4. Bessani, A., Sousa, J., Alchieri, E.E.: State machine replication for the masses with bft-smart. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 355–362 (2014). https://doi.org/10.1109/DSN.2014.43
5. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. pp. 868–886. Springer, Springer Berlin, Heidelberg (2012)
6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
7. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International Conference on Financial Cryptography and Data Security. pp. 423–443. Springer (2020)
8. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI. vol. 99, pp. 173–186. ACM, New York, NY, USA (1999)
9. Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., Song, D.: Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 185–200. IEEE (2019)
10. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23. pp. 409–437. Springer (2017)
11. consensys: Gnark (2023), https://docs.gnark.consensys.net/
12. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144 (2012), https://eprint.iacr.org/2012/144, https://eprint.iacr.org/2012/144
13. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
14. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N., et al.: Zcash protocol specification. GitHub: San Francisco, CA, USA **4**(220), 32 (2016)
15. Hyperledger-TWGC: Hyperledger-twgc/tape: A simple traffic generator for hyperledger fabric (2023), https://github.com/Hyperledger-TWGC/tape
16. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 1353–1370 (2018)
17. Kang, H., Dai, T., Jean-Louis, N., Tao, S., Gu, X.: Fabzk: Supporting privacy-preserving, auditable smart contracts in hyperledger fabric. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 543–555. IEEE, IEEE, Portland, Oregon, USA (2019)
18. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE symposium on security and privacy (SP). pp. 839–858. IEEE (2016)
19. Medicalchain: Medicalchain. https://medicalchain.com (2022)
20. Meier, A.V.: The elgamal cryptosystem. In: Joint Advanced Students Seminar (2005)
21. Monero: The monero project, https://www.getmonero.org/

22. Mouchet, C.V., Bossuat, J.P., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Lattigo: A multiparty homomorphic encryption library in go. In: Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography. pp. 64–70 (2020)
23. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
24. Özcan, A.Ş., Ayduman, C., Türkoğlu, E.R., Savaş, E.: Homomorphic encryption on gpu. IEEE Access (2023)
25. Patel, R., Migliavacca, M., Oriani, M.E.: Blockchain in banking and finance: A bibliometric review. Research in International Business and Finance **62**, 101718 (2022)
26. Raczyński, M.: What is the fastest blockchain and why? analysis of 43 blockchains (Jan 2021), https://alephzero.org/blog/what-is-the-fastest-blockchain-and-why-analysis-of-43-blockchains/
27. Saad, M., Spaulding, J., Njilla, L., Kamhoua, C., Shetty, S., Nyang, D., Mohaisen, D.: Exploring the attack surface of blockchain: A comprehensive survey. IEEE Communications Surveys & Tutorials **22**(3), 1977–2008 (2020)
28. Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL (Mar 2022), microsoft Research, Redmond, WA.
29. Solomon, R., Weber, R., Almashaqbeh, G.: smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption. Cryptology ePrint Archive (2021)
30. Steffen, S., Bichsel, B., Baumgartner, R., Vechev, M.: Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 179–197. IEEE (2022)
31. Steffen, S., Bichsel, B., Gersbach, M., Melchior, N., Tsankov, P., Vechev, M.: zkay: Specifying and enforcing data privacy in smart contracts. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security. pp. 1759–1776 (2019)
32. Sun, X., Yu, F.R., Zhang, P., Sun, Z., Xie, W., Peng, X.: A survey on zero-knowledge proof in blockchain. IEEE network **35**(4), 198–205 (2021)
33. Wang, Z., Li, P., Hou, R., Li, Z., Cao, J., Wang, X., Meng, D.: He-booster: An efficient polynomial arithmetic acceleration on gpus for fully homomorphic encryption. IEEE Transactions on Parallel and Distributed Systems **34**(4), 1067–1081 (2023)
34. Xu, L., Zhang, Y., Zhu, L.: Regulation-friendly privacy-preserving blockchain based on zk-snark. In: International Conference on Advanced Information Systems Engineering. pp. 167–177. Springer (2023)
35. Yang, H., Shen, S., Dai, W., Zhou, L., Liu, Z., Zhao, Y.: Implementing and benchmarking word-wise homomorphic encryption schemes on gpu. Cryptology ePrint Archive (2023)
36. Yuan, M., Wang, D., Zhang, F., Wang, S., Ji, S., Ren, Y.: An examination of multi-key fully homomorphic encryption and its applications. Mathematics **10**(24), 4678 (2022)