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

Rongxin Guan[1(✉)], Ji Qi[1], Tianxiang Shen[1], Heming Cui[1], Sen Wang[2], and Gong Zhang[2]

[1] The University of Hong Kong
rxguan@cs.hku.hk
[2] Huawei Technologies Co., Ltd.

Abstract. Data confidentiality is essential for safety-critical blockchain applications such as digital payment. A promising approach for preserving confidentiality is to encrypt transaction data using homomorphic encryption (HE) and prove the correctness of transaction execution through non-interactive zero-knowledge proofs. However, prior work on this approach is restricted by the use of HE schemes that only support addition or multiplication, making it challenging to implement business logic involving both types of arithmetic operations. In addition, prior work suffers from poor performance caused by the costly HE computation, hindering their adoption for real-world applications.
We present Gafe[3], a high-performance confidentiality-preserving blockchain that incorporates a novel GPU-accelerated transaction execution workflow. Gafe encrypts transaction data with FHE, which allows both addition and multiplication on ciphertexts. For high performance, Gafe leverages parallel execution on GPUs to accelerate FHE computations. To ensure result correctness, Gafe generates lightweight NIZKPs that incur low overhead. Evaluations show that Gafe is highly performant, achieving a 3.1× increase in throughput (258 transactions per second) and a 37% reduction in latency (1.61 seconds), surpassing the baseline.

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

1 Introduction

Blockchain has been an appealing solution for both industry and academia [3] due to its exceptional characteristics, such as immutability [21]. However, notable blockchains like Hyperledger Fabric [2] face a serious confidentiality issue as they process and store transaction data in plaintext, exposing sensitive information to anyone with access to the blockchain. This confidentiality issue impedes the

[3] Gafe stands for GPU-Accelerated Fully Homomorphic Encryption Blockchain.

widespread adoption of blockchain, especially in safety-critical applications such as finance [33], where the confidentiality of transaction data is crucial.

A promising approach to addressing the confidentiality issue is using homomorphic encryption [1] (HE) and non-interactive zero-knowledge proofs [27] (NIZKPs). HE enables arithmetic computation to be performed directly on ciphertexts. NIZKPs allow a prover to prove that a statement is true without revealing any information beyond the validity of the statement itself. In this approach, clients encrypt transaction input data using an HE scheme, and nodes execute transactions directly on ciphertexts. Then, clients generate NIZKPs to prove the correctness of transaction execution, without disclosing the plaintexts of the execution results.

However, prior work on this approach faces two prominent deficiencies. Firstly, they suffer from low performance due to the intensive computational overhead of HE. For instance, the notable confidentiality-preserving blockchain ZeeStar [26] achieves a long commit latency of tens of seconds. As a result, prior work fails to meet the high-performance requirements of many blockchain applications, such as digital payment [33]. Secondly, prior work has serious limitations in expressing complex business logic due to their reliance on partly homomorphic encryption (PHE) [1]. PHE only supports either addition or multiplication on ciphertexts, rendering prior work unsuitable for applications like supply chain management [13] that involve both types of arithmetic operations. Although fully homomorphic encryption (FHE) offers a promising alternative that allows arbitrary computation on ciphertexts, the adoption of FHE is hindered by its even higher computation costs compared to PHE.

Our key insight to address the deficiencies is that, *we can enable the efficient integration of FHE and blockchains by introducing GPU acceleration for transaction execution and FHE computation.* Blockchain transactions are highly parallelizable as they invoke the same smart contract, enabling the parallel computations of FHE. These parallel computations are well-suited for GPUs, which offer superior parallel processing capabilities and are widely available in modern commodity machines. Hence, employing GPU-accelerated FHE is both beneficial and feasible for blockchains, as it enables high performance, confidentiality preservation, and the implementation of complex business logic.

Nonetheless, the trivial combination of blockchain and FHE leads to the problem of inconsistent ciphertext results. Specifically, when given identical inputs, different nodes may generate inconsistent ciphertexts for the same plaintext result. This inconsistency occurs because FHE schemes intentionally introduce random noise to ciphertexts. This noise serves as a protection against attacks aimed at extracting information from the ciphertext [28].

To solve the problem of inconsistency, our second insight is *to integrate lightweight NIZKPs with the trusted execution mechanism of the execute-order-validation blockchain workflow* [2]. This mechanism first executes a transaction on multiple nodes and considers it correct iff a majority of nodes produce consistent results. Inspired by this mechanism, client can generate lightweight NIZKPs that decrypts all ciphertext results and checks the consistency of the majority

System	FHE Support	GPU Acceleration	High Performance
❖ ZeeStar [26]	✗	✗	✗
❖ Ekiden [10]	✗	✗	✓
❖ Hawk [20]	✗	✗	✓
❖ Arbitrum [18]	✗	✗	✓
◆ Zcash [16]	✗	✗	✗
◆ Monero [23]	✗	✗	✗
◆ FabZK [19]	✗	✗	✓
◆ Zether [8]	✗	✗	✗
◆ RFPB [30]	✗	✗	✓
◆ **GAFE**	✓	✓	✓

Table 1: Comparison of GAFE and related confidentiality-preserving blockchains. "❖/◆" represent general-purpose and specific-purpose blockchains, respectively.

of plaintexts. The generation of NIZKPs incurs minimal overhead as it does not perform costly HE arithmetic computations like existing studies [26,30].

These two insights lead to GAFE, a high-performance confidentiality-preserving blockchain. GAFE carries a *GPU-accelerated transaction execution workflow*, comprising four phases. First, the client encrypts transaction data using FHE and sends the encrypted transaction to all executor nodes. Second, each executor uses a GPU to execute FHE computation for multiple transactions in parallel, while the client generates NIZKPs to prove the correctness of execution. Third, the orderer nodes run a consensus protocol to determine the transaction order within each block. Finally, the executors validate and commit the transactions in the predetermined order. In short, this workflow achieves high performance while ensuring the provably correct execution of confidentiality-preserving transactions.

We built GAFE on the codebase of Hyperledger Fabric and compared GAFE with a baseline that performs FHE computation solely on CPU. The results show that GAFE achieves high performance, with an effective throughput of 258 transactions per second and a commit latency of 1.61 seconds. Compared to the baseline, GAFE exhibited a 3.1× increase in throughput and a 37% reduction in latency.

In summary, we make the following contributions: (1) We propose a GPU-accelerated transaction execution workflow that integrates GPU-accelerated FHE into blockchain and ensures the execution correctness through lightweight NIZKPs. (2) We implement GAFE, a high-performance confidentiality-preserving blockchain that incorporates the aforementioned workflow. (3) We conduct evaluations on GAFE, demonstrating its effectiveness and high performance.

2 Related work

2.1 Confidentiality-Preserving Blockchain

We categorize prior work on confidentiality-preserving blockchains into two groups, as shown in Table 1. Note that GAFE is a general-purpose blockchain.

General-purpose blockchains. ZeeStar [26] uses ElGamal encryption [22], which only supports HE addition. While ZeeStar extends addition to emulate multiplication, this extension is inefficient and not applicable to ciphertexts encrypted by different keys. Ekiden [10] relies on hardware with trusted execution environments to execute transactions involving private data, but such hardware may not always be available for commodity machines. Hawk [20] and Arbitrum [18] delegate transaction execution to trusted managers, which have the potential to maliciously expose sensitive transaction data.
Specific-purpose blockchains. To conceal digital payment information such as the payment amount, Zcash [16] employs NIZKPs, while Monero [23] uses ring signatures and stealth addresses. However, their poor performance has impeded their broader adoption [25,8]. FabZK [19] combines NIZKPs with a specialized tabular data structure to realize confidentiality, but this data structure restricts FabZK to performing only HE additions. Zether [8] and RFPB [30][4] also support only HE additions due to the use of PHE schemes.

2.2 GPU-Accelerated Fully Homomorphic Encryption

Much prior work has been proposed to leverage GPU acceleration for FHE computations in order to unlock the potential of FHE in real-world applications. HE-Booster [29] accelerates 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. [31] provides GPU implementations of three notable FHE schemes (BGV [7], BFV [6,14], and CKKS [11]) along with various theoretical and engineering optimizations, including a hybrid key-switching technique and several kernel fusing strategies.

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, as well as 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, each type of participant 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 string *id* and owns a public-private key set for performing FHE computations.

[4] We refer to the system proposed in [30] as RFPB for convenience.

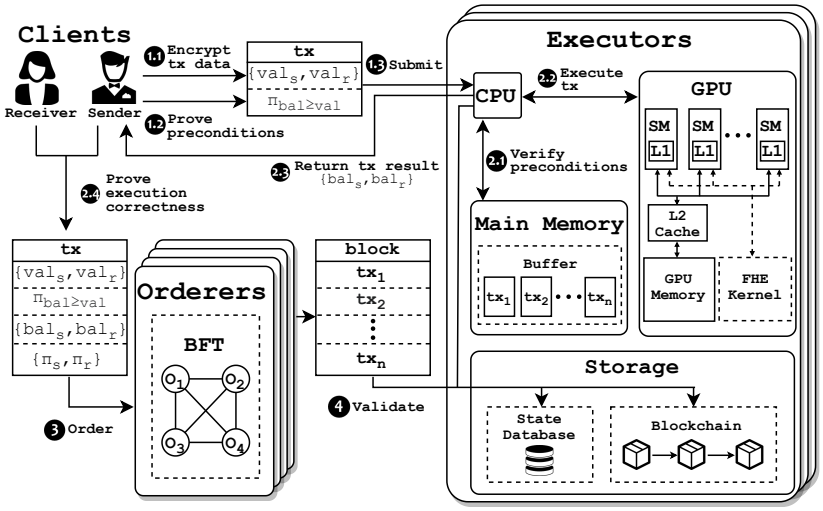


Fig. 1: GAFE's GPU-accelerated transaction execution workflow. Each executor utilizes multiple Streaming Multiprocessors (SM) of a GPU to concurrently execute FHE computations for multiple transactions.

In addition, each client is associated with a floating-point number *bal* that represents the client's balance and is encrypted using an 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. Each executor is equipped with a GPU to accelerate FHE computations during transaction execution. We provide a detailed discussion of GAFE's transaction execution workflow in §4.2.

Orderer. Orderers determines the order of transactions within each block via the BFT-SMaRT [5] protocol, a Byzantine-Fault-Tolerant consensus protocol.

3.2 Threat Model

GAFE adopts the Byzantine failure model [4,9], which tolerates up to N malicious orderers out of a total of $3N + 1$ orderers. Participants within an organization trust each other, but not participants from other organizations. We make standard assumptions on FHE and NIZKP.

3.3 GAFE's Workflow Overview

GAFE consists of two sub-workflows: *client key generation* (§4.1) and *GPU-accelerated transaction execution* (§4.2). Specifically, the 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) and generating 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 tx 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 moved to a GPU for concurrent FHE computations, and the execution results are subsequently returned to the corresponding clients. Upon receiving the results from all executors, the client generates NIZKPs to prove the correctness of transaction execution. Finally, the client sends both the execution 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 disseminate the generated block to all executors for validation.

Phase 4: Validation. On receiving a block from the orderers, the executor sequentially validates each transaction within the block in the determined order. The executor commits a transaction only if the transaction has valid correctness NIZKPs and has no write conflict with previously 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 computation. While both pk and ek are accessible to all participants in the GAFE network, the decryption key sk must remain private to the client.

During the creation of a client, GAFE executes the key generation algorithm associated with the chosen FHE scheme (e.g., CKKS [11]). This algorithm initially generates the encryption key sk, and then derives pk and 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 permissioned blockchains [2] with GPU-accelerated FHE schemes. It consists of four phases, as outlined in Figure 1 and Algorithm 1.
Phase 1: Construction. For confidentiality preservation, GAFE processes and stores transaction data in the form of ciphertext. Thus, 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 involving clients. In the context of digital payment, a transaction involves a sender client c_s and a receiver client c_r. Therefore, c_s encrypts the payment amount val into two ciphertexts, val_s and val_r, using c_s's and c_r's encryption keys, respectively. The reason why two different ciphertexts are generated for val is because GAFE employs single-key FHE,

Algorithm 1: Transaction execution workflow of the client.

input: Plaintext transaction data `val`
// Phase 1: Construction
1 $\mathtt{pk_s} \leftarrow \mathtt{GetEncryptionKey(id_s)}$; $\mathtt{val_s} \leftarrow \mathtt{Encrypt(val,pk_s)}$;
2 $\mathtt{pk_r} \leftarrow \mathtt{GetEncryptionKey(id_r)}$; $\mathtt{val_r} \leftarrow \mathtt{Encrypt(val,pk_r)}$;
3 $\mathtt{bal_s} \leftarrow \mathtt{GetBalance(id_s)}$;
4 $\pi_{\mathtt{bal} \geq \mathtt{val}} \leftarrow \mathtt{GenerateNIZKP(bal_s} \geq \mathtt{val_s}, \mathtt{sk_s})$;
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 $\quad \mathtt{bal'_s}, \mathtt{bal'_r} \leftarrow \mathtt{SendForExecution(e,tx)}$;
9 $\quad \mathtt{results} \leftarrow \mathtt{results} \cup \{\mathtt{bal'_s}, \mathtt{bal'_r}\}$;
10 $\pi_{\mathtt{s}} \leftarrow \mathtt{GenerateNIZKP(a\ majority\ of\ bal'_s\ in\ results\ are\ consistent)}$;
11 $\pi_{\mathtt{r}} \leftarrow \mathtt{GenerateNIZKP(a\ majority\ of\ bal'_r\ in\ results\ are\ consistent)}$;
// 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)`;

which restricts computation to be performed on two ciphertexts only if they are encrypted using the same encryption key. GAFE does not use multi-key FHE, which enables computation on ciphertexts encrypted with different encryption keys. This is due to the extremely high cost associated with multi-key FHE, rendering it impractical for real-world applications [32].

Phase 1.2: Proving preconditions. In certain applications, the client generates 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 confidentiality for c_s's decryption key and the two resulting 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 steps. 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 first verifies the precondition NIZKPs associated with the transaction, such as $\pi_{bal \geq val}$. GAFE proceeds to the subsequent steps iff $\pi_{bal \geq val}$ is valid. Otherwise, GAFE immediately terminates the transaction execution.

Phase 2.2: Executing transactions with GPU-accelerated FHE. Similar to Hyperledger Fabric, each GAFE executor independently executes transactions. Internally, the executor maintains a buffer that temporarily stores the transactions

Algorithm 2: Execution phase of the executor.

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

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 carries out the following three steps: (1) moving the encrypted data of all buffered transactions from main memory to GPU memory, (2) launching the corresponding GPU kernels 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 sender's and receiver's balances $\{bal_s, bal_r\}$. 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 submits the transaction. In the case of digital payment, the executor returns the updated balances $\{bal'_s, bal'_r\}$ to the sender client.

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 de-

cryption 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 [5], 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. When receiving a block from the orderers, the executor follows the pre-determined order to sequentially validate each transaction within the block. The executor will only commit transactions that satisfy two conditions. First, the transaction must not have any write conflict with previously committed transactions within the same block. Second, the transaction must be accompanied by valid correctness NIZKPs, such as $\{\pi_s, \pi_r\}$, which serve as proofs of the transaction's execution correctness. If a transaction fails to meet either of these conditions, the executor aborts the transaction and does not commit it to the state database. Once the executor has validated all transactions, the executor permanently appends the block to the local copy of the blockchain.

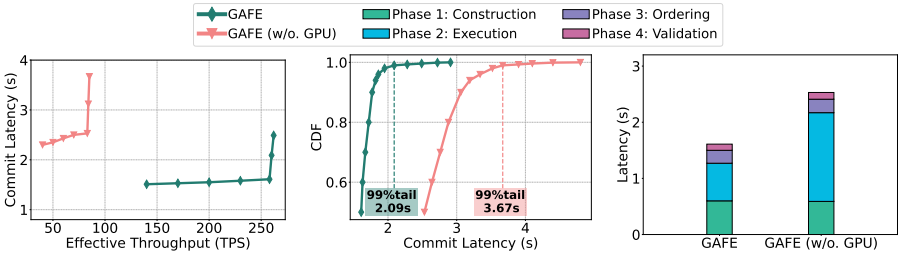
5 Evaluation

5.1 Settings

Implementation. We built a prototype system of GAFE based on Hyperledger Fabric [2] and simulated the business logic of digital payment. We implemented the CKKS scheme [11] for GAFE based on the state-of-the-art studies on GPU-accelerated FHE schemes [24,29,31]. GAFE adopted the gnark [12] library's implementation for the Groth16 NIZKP system [15] and employed the BFT-SMaRT [5] 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.

Metrics. We evaluated two metrics: (1) *effective throughput*, which indicates the average number of 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 (CDF) 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.



(a) Effective throughput and commit latency. (b) CDF of commit latency. (c) Latency of each phase.

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

5.2 End-to-End Performance

We evaluated the end-to-end performance of GAFE and GAFE (W/O. GPU). For each evaluation, we created three executors, four orderers, and one thousand clients. Next, we construct and submit 100,000 digital payment transactions using Tape [17], an efficient benchmark tool for blockchain. In order to prevent transaction aborts caused by write conflicts, we explicitly ensured that no two transactions in the same block shared identical involving clients. 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 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 computations on GPU. Figure 2c 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

We present Gafe, a confidentiality-preserving blockchain that achieves high performance via the novel GPU-accelerated transaction execution workflow. Gafe protects data confidentiality through the use of FHE-encrypted transaction data, ensures the correctness of execution results by generating lightweight NIZKPs, and obtains high performance by leveraging GPUs to execute transactions concurrently. We implemented Gafe on top of Hyperledger Fabric and the latest advancements in GPU-accelerated FHE. Our evaluations demonstrated the superior performance of Gafe compared to the baseline, with a significant 3.1× increase in effective throughput (258 TPS) and a notable 37% decrease in commit latency (1.61 seconds).

References

1. Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys **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. Aste, T., Tasca, P., Di Matteo, T.: Blockchain technologies: The foreseeable impact on society and industry. computer **50**(9), 18–28 (2017)
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. IEEE, IEEE Computer Society, 1730 Massachusetts Ave., NW Washington, DCUnited States (2014)
5. 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
6. 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)
7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
8. 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)
9. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI. vol. 99, pp. 173–186. ACM, New York, NY, USA (1999)
10. 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)

11. 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)
12. consensys: Gnark (2023), https://docs.gnark.consensys.net/
13. Fabric, H.: Case Study:How Walmart brought unprecedented transparency to the food supply chain with Hyperledger Fabric. https://www.hyperledger.org/learn/publications/walmart-case-study (2022)
14. 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
15. 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)
16. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N., et al.: Zcash protocol specification. GitHub: San Francisco, CA, USA **4**(220), 32 (2016)
17. Hyperledger-TWGC: Hyperledger-twgc/tape: A simple traffic generator for hyperledger fabric (2023), https://github.com/Hyperledger-TWGC/tape
18. 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)
19. 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)
20. 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)
21. Landerreche, E., Stevens, M.: On immutability of blockchains. In: Proceedings of 1st ERCIM Blockchain Workshop 2018. European Society for Socially Embedded Technologies (EUSSET) (2018)
22. Meier, A.V.: The elgamal cryptosystem. In: Joint Advanced Students Seminar (2005)
23. Monero: The monero project, https://www.getmonero.org/
24. Özcan, A.Ş., Ayduman, C., Türkoğlu, E.R., Savaş, E.: Homomorphic encryption on gpu. IEEE Access (2023)
25. 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/
26. 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)
27. 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)
28. Viand, A., Jattke, P., Hithnawi, A.: Sok: Fully homomorphic encryption compilers. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1092–1108. IEEE, San Francisco, CA, USA (2021). https://doi.org/10.1109/SP40001.2021.00068
29. 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)

30. 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)
31. 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)
32. Yuan, M., Wang, D., Zhang, F., Wang, S., Ji, S., Ren, Y.: An examination of multikey fully homomorphic encryption and its applications. Mathematics **10**(24), 4678 (2022)
33. Zhang, T., Huang, Z.: Blockchain and central bank digital currency. ICT Express **8**(2), 264–270 (2022)