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

Rongxin Guan[1], Ji Qi[2(✉)], Tianxiang Shen[1], Sen Wang[4], Gong Zhang[4], and Heming Cui[1,3]

[1] The University of Hong Kong, Hong Kong, China
rxguan@cs.hku.hk
[2] Institute of Software, Chinese Academy of Sciences, Beijing, China
qiji@iscas.ac.cn
[3] Shanghai AI Laboratory, Shanghai, China
[4] Huawei Technologies, Hong Kong, China

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 (NIZKPs). However, prior work on this approach suffers from poor performance caused by the costly HE computation, hindering their adoption for real-world applications. In addition, prior work is restricted by the use of HE schemes that only support either addition or multiplication, making it challenging to implement business logic involving both types of arithmetic operations.
We present Gafe[5], a high-performance confidentiality-preserving blockchain that carries a GPU-accelerated transaction execution workflow. Gafe encrypts transaction data with FHE, allowing both addition and multiplication on ciphertexts. For high performance, Gafe leverages parallel execution on GPUs to accelerate FHE computations. For 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 without GPU acceleration.

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

1 Introduction

Blockchain has been a transformative force for both industry and academia [3] due to its exceptional characteristics, such as immutability [18]. However, notable blockchains like Hyperledger Fabric [2] (HLF) face a serious confidentiality

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

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 widespread adoption of blockchain, especially in safety-critical applications such as finance [30], where data confidentiality is crucial.

A promising approach to addressing the confidentiality issue is using homomorphic encryption [1] (HE) and non-interactive zero-knowledge proofs [24] (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 execution correctness, without disclosing the plaintexts of the results.

However, prior work on this approach faces two prominent deficiencies. Firstly, they suffer from low performance due to the intensive computational costs of HE. For instance, the notable confidentiality-preserving blockchain ZeeStar [23] exhibits 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 [30]. Secondly, prior work has serious limitations in expressing complex business logic due to its reliance on partially homomorphic encryption [1] (PHE). PHE supports either addition or multiplication on ciphertexts, making prior work unsuitable for applications like finance [30] that involve both types of arithmetic operations. Although fully homomorphic encryption (FHE) offers a promising alternative that allows arbitrary computation on ciphertexts, its adoption is impeded by its even higher computation costs.

Our key insight to address the deficiencies is that, *we can efficiently integrate FHE and blockchains by introducing GPU acceleration for transaction execution and FHE computation.* Blockchain transactions that invoke the same smart contract are highly parallelizable, thus enabling parallel computations of FHE across multiple transactions. 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 FHE and blockchains 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 [25].

To solve the problem of inconsistency, our second insight is *to integrate lightweight NIZKPs with the trusted execution mechanism of the execute-order-validate 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, clients can generate NIZKPs that decrypt all ciphertext results and check the consistency of the majority of plaintexts. The

System	FHE Support	GPU Acceleration	High Performance
❖ ZeeStar [23]	✗	✗	✗
❖ Ekiden [9]	✗	✗	✓
❖ Hawk [17]	✗	✗	✓
❖ Arbitrum [15]	✗	✗	✓
◆ Zcash [14]	✗	✗	✗
◆ Monero [20]	✗	✗	✗
◆ FabZK [16]	✗	✗	✓
◆ Zether [7]	✗	✗	✗
◆ RFPB [27]	✗	✗	✓
◆ **Gafe**	✓	✓	✓

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

generation of NIZKPs is lightweight as it does not perform costly HE arithmetic computations like existing studies [23,27].

These two insights lead to Gafe, a high-performance confidentiality-preserving blockchain. Gafe carries a *GPU-accelerated transaction execution workflow* in four phases. First, the client encrypts transaction input 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, then the client generates NIZKPs to prove the execution correctness. 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 determined order. In short, this workflow achieves high performance while ensuring the provably correct execution of confidentiality-preserving transactions.

We built Gafe on top of HLF [2], a notable execute-order-validate blockchain framework. We compared Gafe with a baseline that performs FHE computation solely on the 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 achieved 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 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 Blockchains

We categorize prior work on confidentiality-preserving blockchains into two groups, as shown in Table 1.

General-purpose blockchains. ZeeStar [23] uses ElGamal encryption [19], 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 [9], Hawk [17], and Arbitrum [15] present substantial vulnerabilities due to their reliance on trusted managers or hardware. Note that Gafe is a general-purpose blockchain.

Specific-purpose blockchains. To conceal sensitive digital payment information, Zcash [14] employs NIZKPs, while Monero [20] uses ring signatures and stealth addresses. However, their poor performance has impeded their broader adoption [22,7]. FabZK [16] combines NIZKPs with a specialized tabular data structure to realize confidentiality, but this data structure restricts FabZK to performing only HE addition. Zether [7] and RFPB [27][6] also support only HE addition due to the use of PHE schemes.

2.2 GPU-Accelerated Fully Homomorphic Encryption

Much prior work leverages GPU acceleration for FHE computations to unlock the potential of FHE in real-world applications. HE-Booster [26] accelerates polynomial arithmetic computation by mapping five common phases of typical FHE schemes to the GPU parallel architecture. It also introduces thread-level and data-level parallelism to enable acceleration on single-GPU and multi-GPU setups, respectively. Ozcan et al. [21] presents a library that makes efficient use of the GPU memory hierarchy and minimizes the number of GPU kernel function calls. Yang et al. [28] provides GPU implementations of three notable FHE schemes (BGV [6], BFV [5,12], and CKKS [10]) along with various optimizations, including a hybrid key-switching technique and several kernel fusing strategies. Note that Gafe is orthogonal to the implementation of GPU-accelerated FHE schemes, allowing Gafe to leverage the latest advancements in this field.

3 Overview

3.1 System Model

Gafe comprises three types of participants: *client*, *executor*, and *orderer*. Executors and orderers are referred to as *nodes*. Gafe uses permissioned settings, where all participants are organized into distinct *organizations* and explicitly identified. Each organization runs multiple executors and orderers, as well as possesses a set of clients. We built a prototype system of Gafe on top of HLF. Specifically, each type of participant is described as follows:

Client. Clients encrypt transaction input data, submit the encrypted transactions to the nodes for execution and commitment, and prove the execution correctness. Each client is identified by a unique string *id* and owns a public-private key set for performing FHE computations. In addition, each client is associated with FHE ciphertexts, such as the client's balance *bal*.

Executor. Each executor is responsible for three main tasks: (1) executing encrypted transactions, (2) validating transaction results, and (3) maintaining the

[6] We refer to the system proposed in [27] as RFPB for convenience.

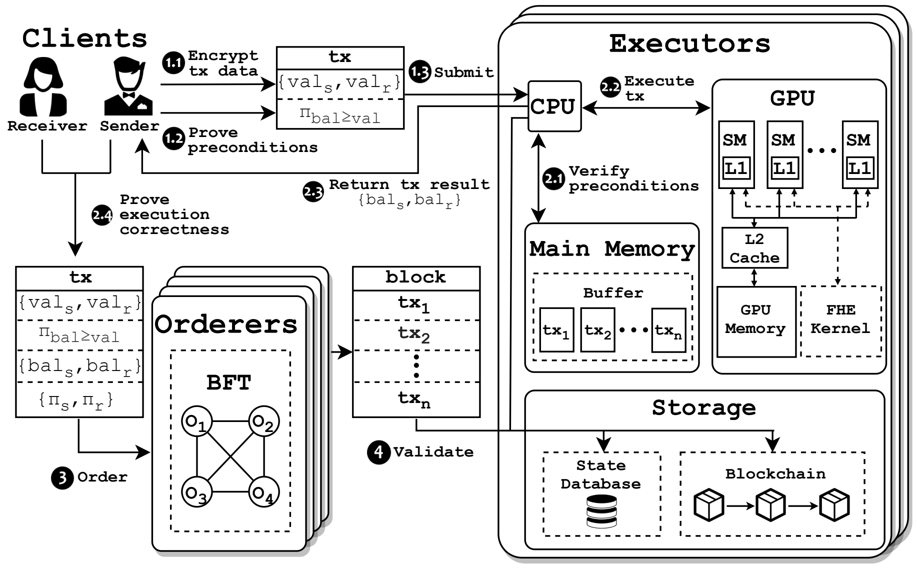


Fig. 1: Gafe's GPU-accelerated transaction execution workflow. Each executor utilizes multiple Streaming Multiprocessors (SM) of a GPU to concurrently execute FHE computations for multiple transactions.

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 determine the order of transactions within each block via a Byzantine-Fault-Tolerant consensus protocol (e.g., PBFT [8]).

3.2 Threat Model

Gafe adopts the Byzantine failure model [4,8], which tolerates up to N malicious orderers out of $3N + 1$ orderers. Each client is limited to accessing only the plaintext of its own data and is unable to access the plaintext of other clients' data. Nodes are incapable of revealing the plaintext of any client's data. We make standard assumptions on FHE and NIZKP.

3.3 Gafe's Workflow Overview

Gafe carries a *GPU-accelerated transaction execution workflow* (§4.2) that consists of four phases, as shown in Figure 1. We take digital payment as an example to illustrate our workflow.

Phase 1: Construction. The client constructs a transaction tx by encrypting the transaction input data (e.g., payment amount val) and 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 tx and $\pi_{bal \geq val}$ to all executors for execution.

Phase 2: Execution. The executor verifies $\pi_{bal \geq val}$ and continues execution iff $\pi_{bal \geq val}$ 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 then the execution results are returned to the submitting clients. Upon receiving the results from all executors, the client generates NIZKPs to prove the execution correctness. 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 iff the transaction has valid correctness NIZKPs and has no write conflict with previously committed transactions within the same block. Otherwise, the executor aborts the transaction.

4 Workflow Description

4.1 Client Key Generation

Each client must generate a unique public-private key set for conducting 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 open to all participants, sk must remain private to the client. Specifically, Gafe runs the key generation algorithm associated with the chosen FHE scheme (e.g., CKKS [10]), which initially generates sk, and then derives pk and ek from sk. In addition, Gafe derives 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 [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 before submitting it for execution.

Phase 1.1: Encrypting transaction data. The client encrypts transaction input data using the encryption keys. In digital payment, a transaction involves a sender client c_s and a receiver client c_r. Thus, 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 that Gafe employs single-key FHE, which restricts computation to be performed on ciphertexts that are encrypted with the same encryption key. Gafe does not use multi-key FHE, which enables computation on ciphertexts encrypted with

Algorithm 1: Transaction execution workflow of the client.

```
   input: Plaintext transaction input 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  tx $\leftarrow \{\mathtt{val_s}, \mathtt{val_r}, \pi_{\mathtt{bal} \geq \mathtt{val}}\}$;
   // Phase 2: Execution
6  results $\leftarrow \varnothing$;
7  For every executor e; do in parallel
8      $\mathtt{bal'_s}, \mathtt{bal'_r} \leftarrow$ SendForExecution(e,tx);
9      results $\leftarrow$ 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 tx $\leftarrow$ tx $\cup \{\mathtt{bal'_s}, \mathtt{bal'_r}, \pi_\mathtt{s}, \pi_\mathtt{r}\}$;
13 SendForOrderingAndValidation(tx);
```

different encryption keys. This is because multi-key FHE incurs an extremely high computational overhead [29], impractical for real-world applications.

Phase 1.2: Proving preconditions. In certain applications, the client generates NIZKPs to prove the satisfaction of preconditions that are compulsory for the correct execution of transactions. For instance, digital payment demands that the sender client c_s's balance bal_s must be not less than val. To prove this precondition, 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 property of NIZKPs, GAFE preserves the confidentiality for c_s's decryption key and the two resulting plaintexts.

Phase 1.3: Submitting transactions. Lastly, the client submits the transaction to all executors, including the encrypted input and precondition NIZKPs.

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, the executor first verifies the precondition NIZKPs associated with the transaction (e.g., $\pi_{bal \geq val}$). If $\pi_{bal \geq val}$ is invalid, the executor terminates the transaction execution.

Phase 2.2: Executing transactions with GPU-accelerated FHE. Each executor independently executes transactions. Internally, the executor maintains a buffer that stores the transactions received within a predefined time frame (e.g., 500 milliseconds). When a timeout occurs or the number of buffered transactions reaches a specific threshold, the executor (1) moves the encrypted data of all buffered transactions from main memory to GPU memory, (2) launches the corresponding GPU kernels of FHE computation, and (3) copies 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 ciphertexts of the payment amount $\{val_s, val_r\}$, and the sender's and receiver's balances $\{bal_s, bal_r\}$.

Algorithm 2: Execution phase of the executor.

```
buffer ← $\varnothing$;
// Phase 2.1: Verifying preconditions
Upon reception of transaction tx from client; do
    $\{\mathtt{val_s}, \mathtt{val_r}, \pi_{\mathtt{bal}\geq\mathtt{val}}\}$ ← tx;
    $\mathtt{bal_s}$ ← GetBalance($\mathtt{id_s}$); $\mathtt{ek_s}$ ← GetEvaluationKey($\mathtt{id_s}$);
    $\mathtt{bal_r}$ ← GetBalance($\mathtt{id_r}$); $\mathtt{ek_r}$ ← GetEvaluationKey($\mathtt{id_r}$);
    if VerifyNIZKP($\pi_{\mathtt{bal}\geq\mathtt{val}}$,$\mathtt{bal_s}$,$\mathtt{val_s}$) = $true$ then
        buffer ← buffer $\cup$ $\{\mathtt{val_s}, \mathtt{val_r}, \mathtt{bal_s}, \mathtt{bal_r}, \mathtt{ek_s}, \mathtt{ek_r}\}$;
    else
        Terminate();
// Phase 2.2: Executing transactions with GPU-accelerated FHE
Upon timeout or |buffer| $\geq$ threshold
    MoveFromMainMeoryToGPUMemory(buffer);
    For every transaction tx in buffer; do in parallel on GPU
        $\mathtt{bal'_s}$ ← FHESub($\mathtt{bal_s}$,$\mathtt{val_s}$,$\mathtt{ek_s}$); $\mathtt{bal'_r}$ ← FHEAdd($\mathtt{bal_r}$,$\mathtt{val_r}$,$\mathtt{ek_r}$);
        buffer ← buffer $\cup$ $\{\mathtt{tx}, \mathtt{bal'_s}, \mathtt{bal'_r}\}$;
    MoveFromGPUMeoryToMainMemory(buffer);
    buffer ← $\varnothing$;
// Phase 2.3: Returning transaction results
For every transaction tx in buffer; do in parallel
    ReturnResultToClient(tx,$\mathtt{id_s}$)
```

Next, the executor launches the GPU kernels for FHE addition and subtraction to compute the updated balances: $bal'_s = bal_s - val_s$ for the sender, and $bal'_r = bal_r + val_r$ for the receiver. Inside each kernel, we concurrently perform polynomial arithmetic computations that are highly parallelizable and pervasive in FHE schemes, such as Number-Theoretic Transform [26]. Note that GAFE is independent of FHE implementation and open to various GPU acceleration techniques [21,26,28]. Once the FHE computation is completed, the executor copies the updated balances $\{bal'_s, bal'_r\}$ back to main memory.

Phase 2.3: Returning transaction results. After execution, the executor returns the 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.

Phase 2.4: Proving execution correctness. Upon receiving the results of a transaction from all executors, the clients of the transaction must prove the execution correctness. To achieve this, the clients must be online and generate NIZKPs to prove the consistency of the majority of the results. In 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. The receiver c_r follows a similar procedure and generates an NIZKP π_r using c_r's decryption key to ensure the consistency of c_r's updated balance. Lastly, c_s sends the following materials to the orderers for ordering: the input 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 a BFT consensus protocol (e.g., PBFT [8]) 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 sequentially validates all transactions according to their order in 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 associated with valid correctness NIZKPs (e.g., $\{\pi_s, \pi_r\}$), 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 HLF v2.5 [2] and simulated the business logic of digital payment. We implemented the CKKS scheme [10] for Gafe based on the state-of-the-art studies on GPU-accelerated FHE [21,26,28]. Gafe adopted the gnark [11] library's implementation for the Groth16 NIZKP system [13] and employed our Golang implementation of the PBFT [8] consensus protocol. We also developed a baseline system called Gafe (w/o. GPU) that 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.

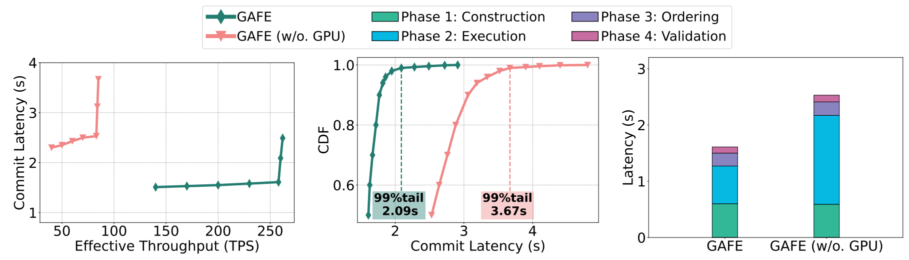
Metrics. We evaluated two metrics: (1) *effective throughput*, which indicates the average number of transactions per second (TPS) committed to the blockchain; and (2) *commit latency*, which measures the time duration from transaction construction to commitment. Additionally, we also reported the cumulative 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 of 4 machines, each 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 created three executors, four orderers, and one thousand clients. Next, we constructed and submitted 100,000 digital payment transactions. To prevent transaction aborts caused by write conflicts, we explicitly ensured that no two transactions in the same block shared identical 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



(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.

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 FHE computations on GPU. Figure 2c compares the latency of each phase between GAFE and the baseline. Note that GAFE achieved notably lower latency in the execution phase. The reduced latency is enabled by GPUs' optimized parallel processing capability, facilitating concurrent execution of a significant portion of arithmetic computations in typical FHE schemes. As a result, GAFE avoids performing FHE computation on the CPU, which has significantly fewer cores and is less efficient in executing a large number of compute-intensive computations in parallel.

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 by encrypting transaction data using FHE, ensures execution correctness by generating lightweight NIZKPs, and achieves high performance by leveraging GPUs to execute transactions concurrently. We implemented GAFE on the codebase of HLF. 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).

Acknowledgements The work is supported in part by National Key R&D Program of China (2022ZD0160200), HK RIF (R7030-22), HK ITF (GHP/169/20SZ), the Huawei Flagship Research Grants in 2021 and 2023, and HK RGC GRF

(Ref: HKU 17208223), the HKU-SCF FinTech AcademyR&D Funding Schemes in 2021 and 2022, the HKU-CAS Joint Laboratory for Intelligent System Software, and the Shanghai Artificial Intelligence Laboratory.

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. 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. 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)
16. 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)
17. 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)
18. Landerreche, E., Stevens, M.: On immutability of blockchains. In: Proceedings of 1st ERCIM Blockchain Workshop 2018. European Society for Socially Embedded Technologies (EUSSET) (2018)
19. Meier, A.V.: The elgamal cryptosystem. In: Joint Advanced Students Seminar (2005)
20. Monero: The monero project, https://www.getmonero.org/
21. Özcan, A.Ş., Ayduman, C., Türkoğlu, E.R., Savaş, E.: Homomorphic encryption on gpu. IEEE Access (2023)
22. 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/
23. 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)
24. 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)
25. 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
26. 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)
27. 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)
28. Yang, H., Shen, S., Dai, W., Zhou, L., Liu, Z., Zhao, Y.: Phantom: A cuda-accelerated word-wise homomorphic encryption library. IEEE Transactions on Dependable and Secure Computing pp. 1–12 (2024). https://doi.org/10.1109/TDSC.2024.3363900
29. 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)
30. Zhang, T., Huang, Z.: Blockchain and central bank digital currency. ICT Express **8**(2), 264–270 (2022)