GRACE: A Confidentiality-Preserving and High-Performance Permissioned Blockchain Framework for General Smart Contracts

Abstract

Data confidentiality among multiple parties is essential for safety-critical blockchain applications. A promising approach to achieving confidentiality is using cryptographic primitives like homomorphic encryption (HE) to encrypt user data, and enforcing the correct execution of contract logic (e.g., payment) through non-interactive zero-knowledge proofs (NIZKPs). However, existing solutions still face significant limitations in supporting general smart contracts: these solutions either cannot tolerate non-deterministic contracts whose correct execution cannot be proven by NIZKPs, or support cryptographic primitives with limited capabilities, such as partial HE, which allows only one type of arithmetic operation (either addition or multiplication) over encrypted data.

We present GRACE, a high-performance confidential permissioned blockchain framework that supports general non-deterministic smart contracts with any cryptographic primitives (e.g., fully HE). Inspired by the multi-party trusted execution mechanism of execute-order-validate (EOV) blockchains, GRACE leverages NIZKPs to prove only that quorum parties produce consistent execution results to ensure the correct execution of contracts, without involving the contract logic. Moreover, GRACE detects and merges multiple conflicting transactions into a single one, minimizing the number of conflict aborts in EOV and cryptographic primitive invocations for high performance. Theoretical analysis and extensive evaluation on notable contracts demonstrate that GRACE achieves the strongest confidentiality guarantee among existing systems and supports general contracts. Compared to three notable confidential permissioned blockhains, GRACE achieved up to 7.14× higher effective throughput and 13% lower end-to-end latency on average.

1 Introduction

Modern blockchains such as Hyperledger Fabric (HLF) [5] are widely deployed in both industry and academia. A driving force is their support for smart contracts that enable trusted execution over a global tamper-resistant shared ledger. However, the paradigm of processing and storing blockchain data in cleartext raises deep concerns about data confidentiality. Such concerns are especially problematic for applications involving highly sensitive information, such as medical data [29], as they are subject to stringent privacy laws and regulations, such as the General Data Protection Regulation of the Europe Union [49].

This problem has motivated previous works to achieve data confidentiality on smart contracts, and these works can be summarized into two main categories. One approach is designing new blockchain architectures to impose strict data access controls without resorting to cryptography technologies [3–5]. For example, HLF uses channels [6] and private data collections [23] to determine which parties are eligible to access certain transaction data. However, this approach still adheres to the paradigm of storing data in cleartext, making it highly vulnerable to corrupted nodes that can maliciously acquire sensitive on-chain data.

An alternative approach is using cryptographic primitives to protect confidentiality and enforce contract execution correctness [31, 44–46], thereby preventing the exposure of sensitive data to corrupted nodes. For example, ZeeStar [44] employs partial homomorphic encryption (PHE) and offloads the PHE computation to the client side. It ensures execution correctness by generating NIZKPs, which are then submitted to the global ledger for third-party verification. Another example is smartFHE [43], where clients are only required to provide NIZKPs to demonstrate the well-formedness of their encrypted inputs, and miners are responsible for performing fully homomorphic encryption (FHE [48]) operations, which support both addition and multiplication over ciphertext.

Unfortunately, existing solutions of the cryptography-based approach face a fundamental dilemma between supporting general smart contracts and achieving high performance. On the one hand, PHE schemes sacrifice smart contract generality as they do not support both on-ciphertext addition and multiplication operations, while FHE schemes are plagued by high computational overhead and long execution latency.

A recent survey [16] shows that under 128-bit security level, a proof-of-concept implementation of the ElGamal encryption scheme [33] (i.e., a PHE scheme) requires approximately 3 microseconds to perform a homomorphic multiplication, while a highly optimized FHE library Microsoft SEAL [41] takes a minimum of 3274 microseconds to complete the same arithmetic computation, resulting in a $10^3\times$ performance gap. On the other hand, these solutions prove execution correctness through re-executing smart contracts and verifying the result consistency, which not only lacks support for non-deterministic smart contracts, but also causes prohibitively high proof generation overhead for integrating NIZKP directly with HE schemes. For instance, using 2048-bit keys to generate a Groth16 [37] NIZKP for the Paillier encryption scheme [38] requires more than 256 GB of RAM, making it an impractical demand for commodity desktop machines available today.

Our key insight to tackle this dilemma is that we can leverage EOV's multi-party trusted execution mechanism to obviate the necessity of re-executing smart contracts. This mechanism enables concurrent transaction processing across multiple executor nodes, resulting in improved throughput and allowing for non-deterministic smart contracts. It also facilitates efficient proving of execution correctness through a straightforward consistency check of the independently executed results, leading to reduced end-to-end latency of transaction processing. These factors collectively contribute to high performance and support for general smart contracts.

This insight leads to GRACE[1], the first high-performance confidentiality-preserving permissioned blockchain framework that supports general blockchain applications. GRACE carries a novel Confidentiality-preserving Execute-Order-Validate (CEOV) workflow for provably correct execution of general smart contracts involving encrypted data, as shown in Figure 1c. For any transaction, it adheres to the following sequence: firstly, a client sends the cleartext input to a trusted executor, known as *lead executor*, which employs a designated cryptographic primitive (e.g., a FHE scheme) specified by the invoking contract to encrypt input data; next, multiple executor nodes concurrently execute the transaction over encrypted data; finally, the client generates an NIZKP to prove the consistency of the results. By leveraging the CEOV workflow, GRACE effectively preserves data confidentiality, while simultaneously providing guarantees of execution correctness, thus mitigating the risk of corrupted nodes revealing sensitive on-chain data.

However, GRACE still faces a non-trivial performance degradation challenge related to cascading transaction conflicting aborts caused by the concurrent execution of CEOV. In particular, when multiple transactions concurrently access the same key-value pair in the shared ledger and at least one of them modifies that pair, only one transaction can successfully commit, leading to the abort of other conflicting transactions. Such conflicting transactions are prevalent in typical blockchain applications [42]. For example, HLF achieved only 24.5% of its conflict-free peak throughput in the stock trading application, as confirmed in Figure 3. This challenge is further exacerbated for GRACE due to the involvement of resource-intensive cryptographic primitives, significantly wasting transaction execution resources.

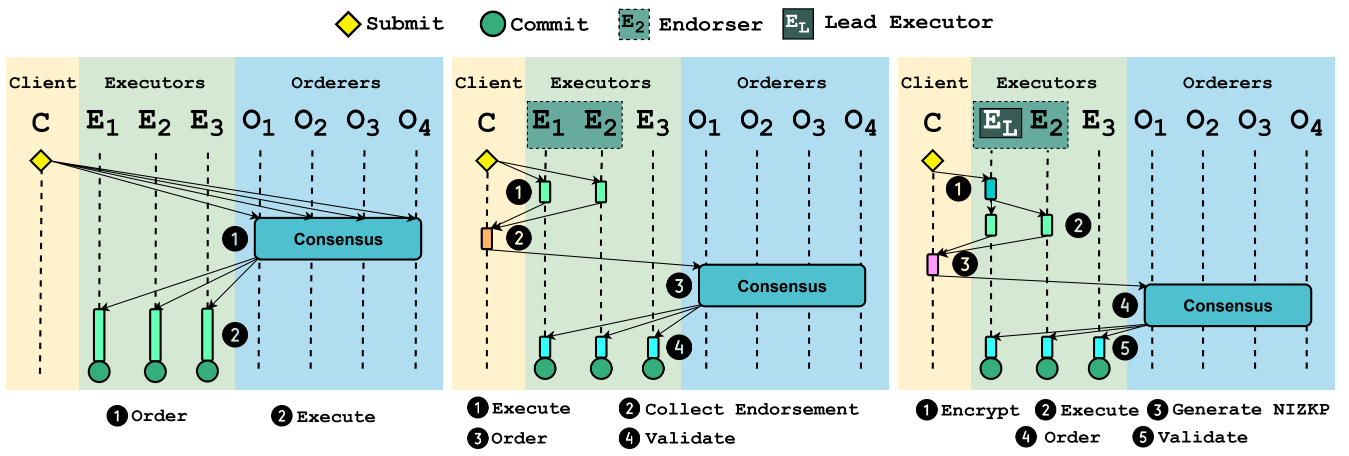
System	Data Encryption	Ciphertext Computability	General Contract	High Performance
✧ HLF [5]	✗	✗	✓	✓
✧ Qanaat [4]	✗	✗	✓	✓
✧ Caper [3]	✗	✗	✓	✓
♦ ZeeStar [44]	✓	✓	✗	✗
♦ smartFHE [43]	✓	✓	✗	✗
♦ Zapper [46]	✓	✗	✗	✗
♦ FabZK [31]	✓	✗	✗	✗
♦ **GRACE**	✓	✓	✓	✓

Table 1: Comparison of GRACE and related confidentiality-preserving blockchain systems. "✧ / ♦" means that the system either takes the non-cryptography-based approach (✧) or the cryptography-based approach (♦).

To tackle the conflicting aborts, we propose a Correlated Transaction Merging (CTM) protocol that utilizes both *offline analysis* for smart contract and *online scheduling* for transactions. Before deployment, a smart contract undergoes the CTM offline analysis. This analysis aims at determining whether multiple conflicting transactions that invoke this contract can be merged into a single equivalent transaction. During runtime, for smart contracts identified as "mergeable" by the offline analysis, the CTM online scheduling constructs a transaction dependency graph on a per-block basis. This graph captures the read-write dependencies among all transactions within the block, enabling the detection of conflicting transactions and facilitating their merging into a single transaction. For instance, if two transactions both perform an addition operation on the same key (e.g., two $X+1$ operations), the CTM protocol merges these two adding transactions into a single adding transaction on that key (e.g., one $X+2$ operation). As a result, the CTM protocol effectively resolves the challenge of cascading transaction conflicting aborts, which are common in EOV-based blockchains (e.g., HLF) where only one transaction commits while the remaining conflicting transactions are invalidated.

We implemented GRACE on Hyperledger Fabric, a mature EOV permissioned blockchain, with an efficient FHE library Lattigo [34] and a fast NIZKP library gnark [15]. We evaluated GRACE with SmallBank (the de facto blockchain benchmark workload) under different conflict ratios and a synthesized microbenchmark. We compared GRACE to three notable confidential blockchain systems, covering both the cryptography-based and non-cryptography-based approaches. Our evaluation shows that:

[1]GRACE stands for GeneRAl Confidentiality-prEserving blockchain



(a) Order-Execute (b) Execute-Order-Validate (c) Confidentiality-preserving Execute-Order-Validate

Figure 1: Three workflows for permissioned blockchains.

- GRACE is efficient (§7.1). GRACE achieved up to 7.14× higher effective throughput and 13% lower end-to-end latency than cryptography-based baselines. Although GRACE witness a 32% drop in throughput compared to the non-cryptography-based baseline, the baseline does not preserve confidentiality when corrupted nodes are present.
- GRACE is secure (§7.2). GRACE harmonizes various cryptographic primitives and is robust to attacks from malicious participants, thereby providing end-to-end confidentiality, liveness, and correctness.
- GRACE is general (§7.3). GRACE utilizes multi-party trusted execution mechanism of the CEOV workflow, enabling the execution of non-deterministic smart contracts with the integration of cryptographic primitives.

Our main contributions are CEOV, a new confidentiality-preserving blockchain workflow tailored for general smart contracts, and CTM, a new concurrency control protocol for transactions involving encrypted data. CEOV addresses the challenge of ensuring the provably correct execution of non-deterministic contracts incorporating general cryptographic primitives (e.g., FHE which supports both ciphertext addition and multiplication), rendering GRACE as secure as existing cryptography-based confidentiality-preserving blockchains. CTM enhances the effective throughput and reduces the average end-to-end latency of GRACE by minimizing the conflicting aborts through the merging of transactions with data races. Overall, GRACE will benefit blockchain applications that desire both data confidentiality and high performance, such as supply chain [20] and healthcare [32]. GRACE can also attract broad traditional non-deterministic applications, developed with general-purpose programming languages like Golang and Java, to be deployed upon. GRACE's code is available at TODO

In the rest of this paper, §2 discusses GRACE's related work. §3 shows an overview of GRACE, §4 GRACE's protocols, §5 the analysis of correctness, liveness, and confidentiality, §6 the implementation details. §7 presents our evaluation, and §8 concludes.

2 Background

2.1 EOV Permissioned Blockchain

A blockchain is a distributed ledger that records transactions across multiple nodes. It can be categorized into two main types: *permissionless* and *permissioned*. Permissionless blockchains (e.g., Bitcoin [35] and Ethereum [50]) are open to anyone, and their participants are mutually untrusted. In contrast, permissioned blockchains (e.g., HLF [5]) are maintained by a group of known organizations, and only grant access to explicitly authenticated participants which are trusted within their organizations. Thanks to explicit identity authentication, permissioned blockchains can utilize fast BFT consensus protocols like HotStuff [1] to tolerate malicious nodes. This enables GRACE to run multiple orderer nodes concurrently, forming a BFT ordering service that determines the transaction order within a few hundred milliseconds, demonstrating the performance advantage of GRACE.

Permissioned blockchains can be classified into two types according to their workflows: Order→Execute (OE) and Execute→Order→Validate (EOV). As depicted in Figure 1a, the OE workflow involves two main steps: Firstly, the orderer nodes establish a globally consistent transaction order (**O**); Subsequently, all executor nodes sequentially execute the transactions in the predetermined order (**E**). This workflow leads to significant performance issue since all executor nodes are required to perform the same execution steps, rendering the throughput cannot scale effectively as the number of executor node increases. In contrast, the EOV workflow (shown in Figure 1b) takes an optimistic approach. Initially, a group of executors called *endorsers* are selected to concurrently execute transactions (**E**). Then, the client collects the executed results, known as endorsement, from endorsers to validate their consistency; if the endorsements are consistent, the client sends the results to the orderers. Subsequently, the orderers determine the transaction order (**O**). Finally, all executors validate transaction results via multi-version concurrency control (MVCC) to prevent conflicting data access within a single block (**V**). GRACE extends EOV and proposes CEOV workflow (shown in Figure 1c) to leverage cryptographic primitives in order to preserve data confidentiality. Consequently, GRACE enjoys EOV's advantage of parallel execution, laying a solid foundation for high performance.

2.2 Homomorphic Encryption

Homomorphic encryption (HE) is a family of encryption schemes that allow direct computation over encrypted data without decryption. A HE scheme consists of four algorithms: *KeyGen* generates key pairs necessary for other operations; *Enc(m, pk, r)* uses the public key *pk* to encrypt a cleartext message *m* into a ciphertext *c* along with randomness *r*; *Dec(c, sk)* uses the private key *sk* to decrypt ciphertext *c* back into the original cleartext *m*; $\oplus$ is a binary operator taking two ciphertexts as input and produces a result ciphertext. For instance, in additive HE, the $\oplus$ operator satisfies the following property:

$$Enc(m_1, pk, r_1) \oplus Enc(m_2, pk, r_2) = Enc(m_1 + m_2, pk, r_3)$$

HE schemes can be categorized into two main types based on the supported operators: partial homomorphic encryption (PHE) and fully homomorphic encryption (FHE). PHE schemes are limited to supporting only one specific operation, such as addition or multiplication. In contrast, FHE schemes allow for arbitrary combinations of these operations. While PHE schemes have relatively low computation overhead, they are limited in their computational capabilities. This restriction sacrifices the generality of smart contracts, as discussed in §1. On the other hand, substantial improvements have been made since the proposal of the first plausible FHE scheme [24]. These improved FHE schemes are capable of preserving the generality of smart contracts, albeit with a higher but still practical computation cost. GRACE utilizes the BFV scheme [12, 21] implementation of Lattigo [34], which is an optimized FHE library. This implementation enables GRACE to perform FHE computations on 58-bit unsigned integers within tens of milliseconds, striking a balance between performance and contract generality.

2.3 Non-Interactive Zero-Knowledge Proofs

Non-interactive zero-knowledge proof (NIZKP) [22, 26] is a cryptographic primitive that enables a prover P to prove knowledge of a secret s to a verifier V without revealing s. Once P generates an NIZKP using a proving key, V can verify the proof using the corresponding verifying key without P being present. Formally, given a proof circuit ϕ, a private input s, and some public input x, prover P can use an NIZKP to prove knowledge of s satisfying a predicate $\phi(s;x)$. One popular type of NIZKP is called zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) [11, 27, 51], which allows any arithmetic circuit ϕ and guarantees constant-cost proof verification in the size of ϕ, making them suitable for blockchain applications [2, 7, 18, 37, 45]. GRACE employs the Groth16 [37] implementation of the gnark library [15] with advantages including constant proof size (several kilobytes) and short verification time (tens of milliseconds).

2.4 Related Work

We now discuss and compare prior work related to GRACE, as illustrated in Table 1.

Non-Cryptograhpy-Based Approach. Several prior works attempt to achieve data confidentiality by adopting specialized blockchain architectures without utilizing cryptography technology. HLF [5] enforces access control on inter-contract and intra-contract levels through channels [6] and private data collection [23], respectively. Caper [3] requires each party to maintain only a partial view of the global ledger and prohibits them from owning copies of other parties' data. Qanaat [4] adopts a hierarchical data model consisting of a set of data collections that store transaction data and are accessible only to authenticated parties. However, these systems do not utilize cryptography primitives and store all data in cleartext, exposing a significant attack surface for corrupted participants seeking to steal confidential data.

Cryptography-Based Approach. Unlike the aforementioned solutions, many work use cryptographic primitives to protect confidentiality and enforce contract execution correctness.

ZeeStar [44] uses exponential ElGamal encryption [33], which is an additive PHE scheme, for encrypting transaction data. It claims to support on-ciphertext multiplication by extending the addition operation. However, this extension is highly inefficient and only applicable to limited scenarios. Consequently, ZeeStar does not support general smart contracts due to the limitations imposed by PHE.

SmartFHE [43] is Ethereum-based and uses FHE schemes for enabling confidentiality-preserving contracts. However, two factors hinder its support for general smart contracts. Firstly, it cannot tolerate non-deterministic smart contracts due to its reliance on the OE workflow. Additionally, the single-key variant of SmartFHE necessitates that all encrypted inputs for a transaction be owned by the same party, while the multi-key variant is currently impractical according to the authors. Consequently, it is now infeasible to compute on foreign values (ciphertexts encrypted by different parties).

Zapper [46] uses an NIZK processor and an oblivious Merkle tree construction to provide confidentiality for both its users and the objects they interact with. Still, Zapper falls short by lacking support for computation on foreign values, thereby limiting its compatibility with general contracts.

Moreover, generating a new transaction in Zapper takes 22 seconds, presenting a notable performance challenge.

FabZK is a HLF-based blockchain that adopts a specialized tabular ledger data structure comprising $N+3$ columns, with N representing the participating organizations. This structure is designed to obfuscate the transaction-related organizations. However, this data structure significantly restricts the compatible smart contracts, thereby compromising its ability to support general smart contracts. Additionally, as the number of organizations increases, the system's performance is severely impacted since each transaction need to update all columns in the public ledgers.

3 Overview

3.1 System Model

GRACE consists of three types of participants: clients, executors, and orderers. The latter two are often referred to as nodes because they act as "servers" in an GRACE network with which clients interact. An GRACE network is typically maintained by multiple organizations, each consisting of clients and nodes. Members of the same organization are mutually trusted, but those from various organizations do not trust each other.

Clients. Clients are end-users who submit transaction proposals to executors for execution and orderers for ordering. Additionally, clients are responsible for generating NIZKP to prove the consistency of transaction results.

Executors. All executors in GRACE are responsible for evaluating transactions and maintaining a local copy of the blockchain, similar to HLF. GRACE executors utilize libHEOV (§6.1), which offers a set of APIs that support NIZKP and HE operations to perform the execution of transactions with homomorphically encrypted data. libHEOV is tailored for EOV contract execution and comes with pre-defined parameters for HE; therefore, smart contract developers do not need to be cryptography experts to efficiently incorporate smart contracts with HE operations.

Besides, a *lead executor* is elected for each organization to implement the CTM protocol, as shown in Figure 2. Lead executors are not different from other executors in transaction execution and blockchain maintenance, but they are assigned the following extra tasks.

- Receive transaction proposals from clients of its organization.
- Optimize transactions by merging multiple transactions targeting the same set of keys into a single transaction.
- Dispatch transaction proposals to other executors and orderers for execution and ordering, respectively.
- Pass transaction responses to clients.

Clients are not required to directly submit their transaction proposals to a lead executor as other executors will finally forward the proposals to it.

Orderers. Orderers in GRACE are the same as in HLF. Specifically, they determine the order of transactions via some consensus protocol (e.g., BFT) and pack them into individual blocks, which are ultimately validated and recorded by relevant executors.

GRACE identifies all participants through a membership mechanism similar to HLF, where each participant is issued a pair of public/secret keys that encode their identity information. This mechanism enables access control to GRACE resources and, more importantly, makes all operations on GRACE auditable and traceable since any transaction is signed by a specific participant with the corresponding privilege.

3.2 Threat model

GRACE adopts the Byzantine failure model [9, 13], where malicious orderers run BFT consensus protocols that tolerate up to f malicious orderers out of $3f+1$ total orderers.

GRACE groups participants (i.e., clients, executor nodes, and orderer nodes) into organizations. Participants are mutually trusted only when they belong to the same organization. This implies that clients and nodes are mutually untrusted, as an organization only contains clients or nodes. Any clients or nodes can be malicious, in which case they are called attackers.

3.3 Workflow Overview

The runtime workflow of GRACE can be summarized into several steps, as shown in Figure 2. A more comprehensive discussion of the runtime workflow can be found in §4.2.

Step 1: Correlated merging. The lead executor collects transaction proposals, merges consecutive *mergeable* proposals, and leaves non-mergeable proposals as is.

Step 2: encryption. The lead executor encrypts the merged proposals using relevant organizations' public keys and signs each transaction with its certificate.

Step 3: dispatch. The lead executor dispatches the signed transactions to other executors, gathering clients' signatures and notifying clients of the transaction ID.

Step 4: execution. Executors invoke smart contracts, sign proposals, and send them back to the lead executor along with the result.

Step 5: proof generation. The lead executor generates an NIZKP to prove the plaintext consistency of results without re-running the smart contract.

Step 6: consensus. GRACE orderer nodes use a BFT protocol to reach a consensus on the order of transactions in a block.

Step 7: commit. Executors commit a transaction after receiving a valid block, ensuring no overlap in read-write set with previous valid transactions.

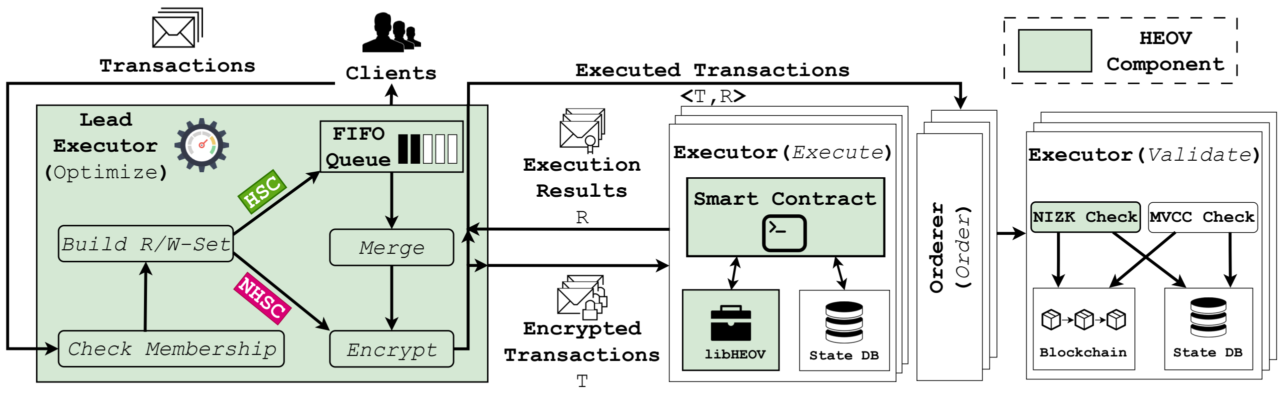


Figure 2: The GRACE Runtime Workflow. GRACE components are highlighted in green. An "HSC" transaction invokes a Homomorphic Smart Contract, while the "NHSC" one does not.

Overall, the highlight of the GRACE runtime workflow stems from achieving data privacy and high performance, which are ensured by several key elements. To guarantee the correctness of results, a lightweight NIZKP is generated. The workflow leverages the CTM protocol, which optimizes mergeable transactions, effectively reducing the occurrence of conflicting transactions and the need for HE operations. To maintain data confidentiality, lead executors are responsible for safeguarding the private keys of their respective organizations. Additionally, the libHEOV library plays a crucial role in facilitating HE computations and NIZKP operations. By combining these elements, the GRACE runtime workflow achieves both data privacy and high performance for permissioned blockchain smart contracts.

4 Protocol

4.1 Offline Homomorphic Smart Contract Analysis

GRACE's offline analysis protocol determines if a smart contract is *homomorphic*, meaning multiple transactions modifying the same set of states can be merged without affecting the execution results. As shown in Figure **??**, before deployment, a smart contract analyzer conducts the analysis and generates metadata indicating if the contract is additive homomorphic (**AH**), multiplicative homomorphic (**MH**), or non-homomorphic (**NH**). This metadata is essential for permitting lead executors to perform transaction merging.

GRACE analyzes smart contracts with random interpretation [2], which generates different input and initial states to capture all possible control flow paths. As a first step, two random input sets I_1 and I_2, along with a random initial memory state M, are generated for the candidate smart contract SC. The memory state contains the SC outputs and all keys written by SC. Then, SC is interpreted in two different ways. First, GRACE executes SC twice consecutively: first with I_1, producing M_{I_1}, and then with I_2, producing the final state M_{I_1,I_2}. Simultaneously, GRACE begins with the same initial state M and executes SC once with I_1+I_2, producing $M_{I_1+I_2}$. If $M_{I_1,I_2} = M_{I_1+I_2}$ for all control flow paths, SC is additive homomorphic. Similarly, if $M_{I_1,I_2} = M_{I_1\times I_2}$, it is multiplicative homomorphic.

4.2 Runtime Transaction Execution and Commitment

The runtime transaction execution and commitment protocol specifies the rules for GRACE's transaction execution, as visualized in Figure 2.

Phase 1: Submission. In the runtime protocol, the client first submits a transaction proposal to an executor within its organization. If the receiving leader is not a lead executor, it forwards the proposal to the designated lead executor. The submission process uses mutual TLS, which ensures the confidentiality of proposal arguments and requires both parties to authenticate themselves by presenting certificates to establish their identities.

Phase 2: Pre-execution preparation. Upon receiving a transaction from a client, the lead executor undertakes a series of sequential preparation tasks: (**Phase 2.1**) constructing the read-write set, (**Phase 2.2 and 2.3**) merging homomorphic transactions, and (**Phase 2.4**) encrypting transaction arguments.

Phase 2.1: Read-write set construction. The read-write set of a transaction includes the read set and the write set, which encompass the states being read and written, respectively. Constructing a read-write set helps identify the owner of the value being written, which is necessary for selecting the appropriate public key to encrypt the value into ciphertext.

Phase 2.2: Transaction diversion. During Phase 2.2 and 2.3, lead executors handle transactions differently based on the homomorphic property of the smart contracts they invoke. This property is determined through the offline homomorphic smart contract analysis protocol mentioned in §4.1. For non-homomorphic smart contracts, the CTM protocol is not applicable, and proposals for these contracts undergo direct encryption as specified in Phase 2.4. For homomorphic smart contracts, proposals are temporarily stored in a FIFO queue, which collects all the proposals that will be packaged into the same block, thereby facilitating the potential merging of these proposals.

Phase 2.3: Correlated merging. The lead executor merges consecutive correlated proposals from the FIFO queue, specifically those that invoke the same homomorphic smart contract and have overlapping read-write sets. This merging process reduces the number of transaction proposals by merging certain proposals into newly merged ones. For example, consider two correlated $transfer$ transactions, T_1 and T_2, with shared relevant parties and inputs I_1 and I_2, respectively. After merging, a new transaction T_{1+2} is created, incorporating the combined input $I_1 + I_2$, effectively replacing T_1 and T_2.

Phase 2.4: Homomorphic encryption. Before submission, the arguments of each proposal undergo encryption using public keys corresponding to the identities listed in the write set. This encryption step is crucial because each argument may be utilized on different ciphertexts owned by clients from other organizations. For example, if *Sam* transfers M tokens to *Rachel*, the plaintext value M must be encrypted into two ciphertext, $Ct_{M,1}$ and $Ct_{M,2}$, using the public keys of *Sam*'s and *Rachel*'s organizations, respectively.

Phase 3: Dispatch. The lead executor signs and dispatches the proposal to other executors based on the endorsement policy. The client transaction ID and proposal signatures are retained as metadata, enabling clients to verify the correct execution of their proposals.

Phase 4: Execution. An GRACE executor executes a smart contract and sends the execution response to the lead executor. Unlike HLF, GRACE smart contracts leverage the capabilities of libHEOV for HE and NIZKP operations. Additionally, GRACE smart contracts use NIZKP as an alternative to the *if* statements found in most programming languages. In GRACE, *if* statements are neither recommended nor allowed when ciphertext is involved in the condition expression because HE ciphertext does not support comparison operations such as $<$, $>$, or $=$, and executors are not authorized to access private keys, making it impossible to decrypt the ciphertext and compare the underlying plaintext. To address this, GRACE smart contracts generate NIZKP that validate the execution results and are stored on the blockchain for future reference.

Phase 5: Correctness NIZKP generation. Before submitting a transaction for ordering, the lead executor collects results from the executing executors. Each result comprises ciphertext and endorsement information (i.e., the acknowledgment of the transaction). The lead executor decrypts and compares the ciphertexts to ensure *plaintext consistency*. If a ciphertext belongs to another organization, the lead executor requests verification from that organization. Valid results require a quorum, typically a majority, of ciphertexts referencing the same plaintext. Only transactions with valid results are eligible for ordering. To ensure the integrity and prevent forgery of results, the lead executor generates an NIZKP to prove the plaintext consistency, which is also stored on the blockchain and is publicly verifiable. By employing this approach, the lead executor cannot create a fake result or tamper with the on-chain data, as the verification process ensures the accuracy and integrity of the results.

Phase 6: Ordering. GRACE orderers run a BFT consensus protocol (e.g., BFT-SMaRT [10]) to reach a consensus on the order of transactions in one block. The block is then distributed to all executors for validation once finalized by orderers.

Phase 7: Validation. Upon receiving a block from orderers, an executor sequentially validates each transaction, including verifying the consistency NIZKP associated with the transactions. If validation is successful, the executor applies the transaction result to its local world state database, ensuring accurate reflection of transaction changes. After validating and processing all transactions in the block, the block is permanently appended to the local blockchain. Since all executors take deterministic and consistent actions for the same block, all local copies of the blockchain maintain consistency, ensuring the overall consistency of GRACE.

Phase 8: Response. As a final step, the lead executor responds to the client who proposed the transaction. This response informs the client whether their transaction has been accepted or rejected.

4.3 Defense Against No-Resubmission Attack

No-resubmission attack is a performance degradation attack that exploits the EOV workflow by maliciously skipping the transaction resubmission process. The attack is simple but detrimental because it wastes a significant amount of the computation power of executors, especially when smart contracts involve heavy HE computations. However, distinguishing no-resubmission attacks from legitimate actions can be challenging. A benign transaction can produce different results when executed by different executors, particularly for non-deterministic smart contracts. In such cases, the results of the current execution are discarded, and no-resubmission becomes a necessary action.

To mitigate the damages caused by no-resubmission attacks, GRACE adopts a per-executor counter-based approach. To be specific, each executor maintains a counter for each lead executor in the GRACE network, tracking the number of no-resubmission instances within a timeframe. If the count exceeds a configurable upper bound, the executor rejects transactions from that lead executor for a specified duration, mitigating the damages caused by no-resubmission attacks.

5 Analysis

This section defines and analyzes the three guarantees of

GRACE: correctness, liveness, and confidentiality.

5.1 Correctness

Definition 1 (Correctness). *For any two benign nodes $Node^1$ and $Node^2$, they hold the same world state with the n-th block as the latest block. The $(n+1)$-th blocks received by $Node^1$ and $Node^2$, denoted as $Block^1_{n+1}$ and $Block^2_{n+1}$ respectively, are consistent and deterministic.*

The correctness of smart contract execution in GRACE is ensured by satisfying ACID and BFT safety properties at the transaction and block levels, respectively.

GRACE guarantees ACID properties for transaction execution, which are the foundation of all reliable failure-tolerant transaction processing systems and are indispensable as concurrent transaction execution is possible and common in EOV workflow.

- *Atomicity*. GRACE executes and commits (or aborts) each transaction as a whole.
- *Consistency*. GRACE only commits valid transactions with correct execution results (conform to the smart contract's logic) that satisfy the application-level endorsement policy 4.2. Specifically, the execution results of a transaction are considered valid as long as enough executors digitally sign the result.
- *Isolation*. GRACE inherits the isolation property from the EOV workflow. In other words, each GRACE transaction operates as if it is the only transaction executing, without interference or conflicts with other transactions.
- *Durability*. Same as existing EOV permissioned blockchains [5], GRACE inherits the durability guarantee from EOV's consensus protocol. Executors can retrieve the blockchain's transactions from their local copies of the blockchain.

GRACE achieves equivalent BFT safety as existing EOV blockchains [5] through the presence of two crucial properties:

- *Block consistency*. GRACE treats the consensus protocol as a blackbox, thereby enjoying the mature consistency (safety) guarantee of existing BFT consensus protocols [10] and their implementations. This ensures that all executors process and append the same set of blocks in an identical order determined by orderers.
- *State consistency*. Every executor follows the same deterministic protocol (§4.2) to validate each transaction in a given block. A transaction's result is committed to the state database *iff* all of its results produced by various executors are consistent and do not modify any key that has already been modified by a previous transaction in the same block. In this way, all executors maintain a consistent state and a local copy of the blockchain after validation of a newly appended block.

5.2 Liveness

Definition 2 (Liveness). *Let C be a benign client and T be a well-formed transaction submitted by C and accepted by the lead executor LE. T will eventually be committed to* GRACE*, even in the event of a crash of C or a reboot of LE.*

Two liveness requirements must be met to commit T to GRACE successfully.

Firstly, the orderers must include T in a publicly agreed-upon clock to guarantee its proper ordering and inclusion in the blockchain. This requirement is satisfied as GRACE inherits the BFT liveness guarantee from its BFT protocol [10] running in EOV's ordering phase.

Secondly, T needs to be submitted to and executed by executors during the execution phase. This is addressed by the combined liveness guarantees of GRACE lead executor (LE) and the inherited EOV workflow. After T is accepted by LE, it is stored in a persistent FIFO queue that withstands crashes. In the event of a crash, T is not lost and will be finally dispatched to executors after LE reboots. The endorsement policy [5], which is an application-level agreement protocol among participating organizations in the EOV workflow, ensures that a sufficient number of benign organizations execute the transactions. This prevents a certain number of malicious executors from tampering with the correct result. For example, in a typical token transfer application with three organizations (*orgA*, *orgB*, *orgC*), the endorsement policy is *majority(orgA, orgB, orgC)*, which requires that every T must be endorsed by at least two of them and ensures that at most one organization can act maliciously.

5.3 Confidentiality

Definition 3 (Confidentiality). *Suppose C is a smart contract following the guideline of §6.2. With this setup, data owners can securely store and process the ciphertext CT without compromising the confidentiality of the corresponding plaintext PT. An active attacker A cannot deduce PT from CT (storage confidentiality). Moreover, even if A observes the execution of C and knows its type, A cannot infer specific details about the changes made to PT (computation confidentiality).*

GRACE is instantiated with two specific cryptographic primitives: (1) the BFV [12,21] scheme, which achieves IND-CPA security based on the Decisional Ring Learning With Errors (D-RLWE) problem, and (2) Groth16 [26], which is a computationally sound and perfectly NIZKP system. When a probabilistic polynomial time attacker A corrupts a participating organization and observes all transactions, GRACE ensures *storage confidentiality*. This is achieved because A is computationally unable to distinguish the plaintext of a given ciphertext without the corresponding private key (decryption key), thanks to the IND-CPA security of the BFV scheme. Furthermore, throughout the transaction execution, transaction arguments are always concealed in ciphertext form. This ensures that any two transactions referring to the same set of keys are probabilistically indistinguishable. In other words, A cannot gain any information beyond the identities of relevant keys during transaction execution, thereby achieving *computation confidentiality*.

6 Implementation

We implement GRACE on the codebase of HLF v2.5 LTS [5]. The smart contract syntax of GRACE remains the same as HLF, but smart contracts need to be rewritten to use the libHEOV API for leveraging HE and NIZKP support, as discussed in §6.1.

6.1 libHEOV

libHEOV is a developer-friendly toolkit that minimizes the need for cryptography expertise and simplifies the use of HE and NIZKP functionalities. It harmonizes two mature libraries, lattigo [34] and gnark [15], which handle the arithmetic computations for HE and NIZKP operations, respectively.

Cryptography Arguments. libHEOV provides pre-defined cryptography arguments out-of-the-box (which will be discussed shortly) that balance security, speed, and memory consumption. libHEOV also offers high configurability, enabling developers to customize these arguments to achieve their desired trade-off between security and performance.

Homomorphic Encryption. libHEOV employs the BFV scheme [12, 21] as its FHE solution, ensuring 128-bit security in the default setting and supporting 58-bit unsigned integer arithmetic. HE multiplications in this setting take only tens of milliseconds on modern commodity servers [34]. Furthermore, the 58-bit unsigned integer is sufficient for many general applications without concerns about overflow.

Non-Interactive Zero-Knowledge Proof. libHEOV uses Groth16 [26] on the BN254 elliptic curve as its NIZKP solution. Like other systems relying on Groth16 zk-SNARKs [7, 44], our implementation requires a circuit-specific trusted setup, which can be performed using secure multi-party computation [8]. The overhead of the trusted setup process is considered trivial, as it is a one-shot procedure and therefore not included in the later evaluation section.

6.2 Rewrite Smart Contract With libHEOV

To enable HE and NIZKP support, developers need to manually rewrite a HLF smart contract according to the specifications stated below. Automated transpiling is not applicable here as it is challenging due to the potential ambiguity of smart contract semantics.

Use encrypted data type for private data. An encrypted variable is the ciphertext encrypted from its underlying plaintext. Throughout the execution of HE computation, the plaintext of it is never revealed without access to the relevant private key, ensuring the confidentiality of private data.

Use libHEOV functions to express HE and NIZKP operations. Replace standard addition and multiplication logic with the provided *Add* and *Mul* functions for HE computation. For NIZKP operations, use the $ProofGen$ and $ProofVerify$ functions to generate new proof instances and verify proofs with specific public input.

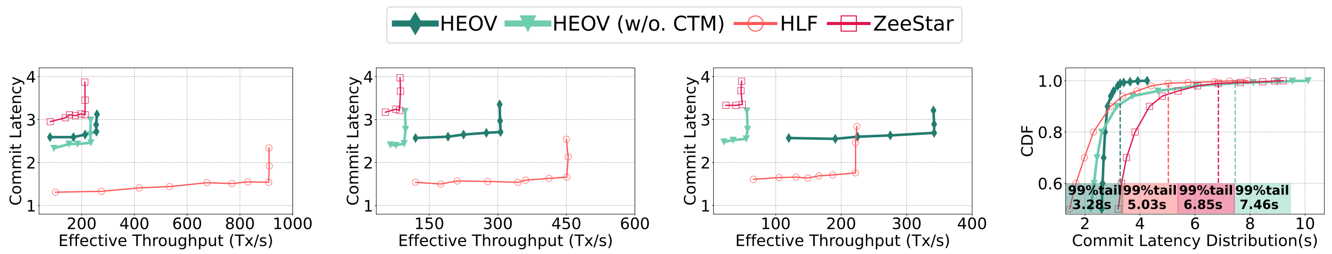
Use NIZKP instead of if-statements. Although most modern programming languages support if-statements or similar structures to control program flow based on conditional expressions, these statements cannot handle ciphertext directly due to the lack of support for direct comparison. As an alternative, GRACE employs NIZKP to simulate if-statements on the ciphertext. However, note that NIZKP cannot perfectly replace if-statements because, if a proof is verified as false, the program flow will exit without executing the remaining instructions.

7 Evaluation

Testbed. We ran all experiments in a cluster with 20 machines, each equipped with a 2.60GHz E5-2690 CPU, 64GB memory, and a 40Gbps NIC. Each node and client was run in a separate docker container. A multi-host Docker Swarm network [17] orchestrated all participant containers. The average node-to-node RTT was about 0.2 ms.

Baselines. To evaluate the performance of GRACE, we compared it with three notable confidentiality-preserving blockchain systems: HLF [5], ZeeStar [44], and GRACE (w/o. CTM). HLF adopts a non-cryptography approach to ensure data confidentiality and is one of the most popular blockchain frameworks in both industry and academia [25, 39, 40]. ZeeStar is a notable cryptography-based blockchain, which encrypts sensitive data by using exponential ElGamal encryption [33] (i.e., a PHE scheme) and generates Groth16 NIZKPs to prove correct contract execution. During our evaluation, we utilized the open-source implementations of HLF and ZeeStar. Additionally, We developed GRACE (w/o. CTM), which represents GRACE without the integration of the CTM protocol, in order to assess the performance gains achieved by incorporating the CTM protocol.

Workloads. We evaluated GRACE and three baselines with two workloads. The first workload is *SmallBank* [28], a widely used benchmark [39, 42] for evaluating blockchain systems that simulates bank business logic. The SmallBank workload was used to measure end-to-end performance (§7.1) and performance under no-resubmission attacks (§7.2). Due to the disparity in capabilities, we created three semantically equivalent implementations of SmallBank for each blockchain platform: an unencrypted version for HLF, an implementation



(a) App1: voting (CR:10%) (b) App2: supply chain (CR:50%) (c) App3: stock trading (CR:90%) (d) Commit latency under App2

Figure 3: End-to-end performance of GRACE and HLF under applications with different conflict ratios.

using additive homomorphic encryption for ZeeStar, and an implementation using FHE with the libHEOV APIs (§6.1) for GRACE and GRACE (w/o. CTM). Each experiment started with ten organizations, each with 100 accounts initialized with the same balance. Subsequently, we generated a set of *transfer* transactions, where a randomly selected account $Acct_1$ transferred a specific number of tokens to another randomly chosen account $Acct_2$. Note that $Acct_1$ and $Acct_2$ may or may not belong to the same organization.

Our second workload is a microbenchmark designed to showcase the performance improvement introduced by the CTM protocol. It comprises two smart contracts, SC_A and SC_M, which execute homomorphic additive and multiplicative transactions, respectively. The ratio between these two transaction types is configurable to simulate various scenarios. We assessed this workload exclusively on GRACE and GRACE (w/o. CTM) (§7.3), as HLF does not support on-ciphertext computation and ZeeStar does not utilize the CTM protocol, therefore not applicable to this evaluation.

Metrics. We evaluated two metrics: *effective throughput*, which represents the average number of valid client transactions committed to the blockchain per second, excluding conflicting aborted transactions, and *commit latency* (also known as *end-to-end latency*), which measures the duration from client submission to transaction commitment. Note that if a transaction fails to commit, GRACE's lead executor will automatically resubmit the transaction until it is eventually committed. Additionally, we reported the 99th percentile commit latency and the cumulative distribution function (CDF) of the transactions' commit latency.

Evaluation methodology. We developed a distributed benchmark using Tape [30], an efficient benchmark tool for HLF. The benchmark spawned 128 clients across multiple servers to prevent the benchmarking tool from becoming a bottleneck.

Each system was configured with ten executors and four orderers, utilizing BFT-SMaRT [10] as the consensus protocol. The default block size was set to 100 transactions, a commonly used setting in relevant studies [40, 42]. To mimic real-world scenarios, we designated 1% of the accounts as *Hot Accounts*, and the *Conflict Ratio* represented the probability of each transaction accessing these hot accounts, with a default value of 50%. For each evaluation, we performed ten runs and reported the average values for each metric. Our evaluation focused on three primary questions.

§7.1 How efficient is GRACE in a benign environment?

§7.2 How robust is GRACE against no-resubmission attacks?

§7.3 How efficient is GRACE across diverse applications?

7.1 End-to-End Performance

We first conducted experiments in benign environments where the network was stable and all participants behaved correctly. All systems are evaluated with different conflict ratios of 10%, 50%, and 90%, respectively.

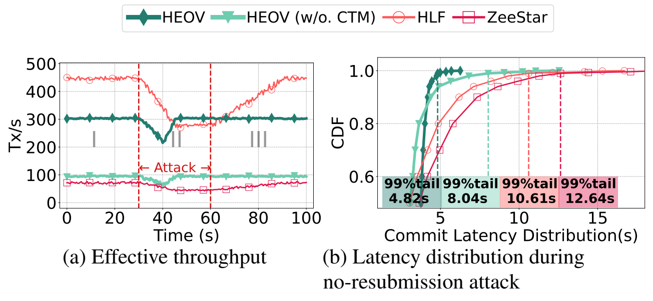
GRACE outperformed both ZeeStar and GRACE (w/o. CTM) in terms of effective throughput, achieving increases up to 7.14×. In Figure 3, GRACE achieved throughput values of 255 TPS, 307 TPS, and 343 TPS at conflict ratios of 10%, 50%, and 90%, respectively. In contrast, GRACE (w/o. CTM) only achieved 235 TPS, 95 TPS, and 55 TPS, demonstrating a significant performance gap between GRACE and GRACE (w/o. CTM), especially with higher conflict ratios. ZeeStar performed even worse, achieving only 213 TPS, 85 TPS, and 48 TPS.

The average end-to-end latency for GRACE is 2.7s, which is only 87% of the baseline ZeeStar (3.1s) and slightly higher than GRACE (w/o. CTM) (2.4s). The additional latency in GRACE is due to the pending time required for merging transactions, as introduced by the CTM protocol (§4.2). However, the latency overhead in GRACE was merely 10%, while the throughput gain could reach up to 5.2×. Even when using FHE, which incurs higher costs than the PHE solution of ZeeStar, GRACE demonstrated shorter average end-to-end latency compared to ZeeStar. This advantage is due to GRACE's lightweight NIZKP (§4.2), eliminating the need to prove the correctness of individual data updates in each transaction, a requirement in ZeeStar.

GRACE is particularly suitable for applications with conflicting transactions thanks to its CTM protocol (§4.2). Surprisingly, in Figure 5, GRACE significantly outperformed HLF with 1.54x higher throughput at a 90% conflict ratio, de-

System	Average Latency (s)					99% tail
	P	E	O	V	E2E	latency (s)
GRACE	0.75	0.85	0.89	0.23	2.72	3.28
GRACE (w/o. CTM)	0.47	0.87	0.88	0.24	2.46	7.46
ZeeStar	n/a	n/a	n/a	n/a	3.11	6.85
HLF	0.03	0.41	0.91	0.19	1.54	5.03

Table 2: Latency of systems under test in Figure 3b and 3d. The "P" phase represents the preparation period before execution.



(a) Effective throughput (b) Latency distribution during no-resubmission attack

Figure 4: Performance under no-resubmission attack

spite the extra overhead from HE and NIZKP generation. In less conflicting scenarios, GRACE was not as performant as HLF, but it still achieved higher throughput than ZeeStar and GRACE (w/o. CTM) in all three settings, albeit at the cost of ensuring data confidentiality.

As shown in Figure 3, GRACE exhibited an increasing trend in throughput as the conflict ratio increased, performing even better in less conflicting scenarios. In contrast, ZeeStar, GRACE (w/o. CTM), and HLF suffered from a significant drop in throughput. This is because GRACE's CTM protocol effectively reduces a substantial portion of conflicts, resulting in fewer cryptographic operations and conflicting aborts. Unlike the other systems, GRACE efficiently handles conflicts, allowing for sustained high throughput.

Figure 3d shows that GRACE's average latency (2.7s) was higher than that of HLF (1.6s), due to the extra overhead introduced by the HE and NIZKP operations. However, the increase in latency is fully justified by GRACE's confidentiality guarantee for general smart contracts. Furthermore, it is noteworthy that GRACE achieved a shorter 99%-tail latency compared to HLF, and all transaction latencies were concentrated within a narrow range. This indicates that the CTM protocol employed by GRACE generally reduces the need for transaction re-submission.

Table 2 provides a more detailed insight into the latency for each phase: preparation (**P**), execution (**E**), ordering (**O**), and validation (**V**). GRACE incurred extra overhead primarily in the preparation and execution phases due to encryption, merging, expensive operations like HE evaluation, NIZKP generation, and verification. The validation phase also experienced a slight latency increase due to NIZKP verification. The ordering phase, on the other hand, remains unaffected as expected. These results highlight the trade-off between the confidentiality guarantee provided by GRACE and the extra overhead from FHE and NIZKP operations.

Overall, GRACE offers a compelling combination of high performance and strong security guarantees, making it well-suited for applications that prioritize data privacy and involve conflicting transactions.

7.2 Robustness to Malicious Participants

We conducted no-resubmission attacks on four systems (Figure 4) involving four benign organizations (O_B) and one malicious organization (O_M). A victim organization O_V was selected from O_B. O_M created transactions involving O_V and intentionally did not resubmit them in order to attack O_V. The experiments were divided into three consecutive periods: *Period I: pre-attack*, *Period II: attack*, and *Period III: post-attack*. Metrics are shown in Figure 4.

Figure 4a shows that initially, all four systems experienced varying levels of performance degradation during the no-resubmission attack. However, GRACE and GRACE (w/o. CTM)'s throughput quickly resumed to the *Period I* level, while ZeeStar and HLF remained at a low level. This showcases the effectiveness of the countermeasure, where executors of benign organizations block malicious lead executors, preventing monopolization and ensuring fair access.

In Figure 4b, both GRACE and GRACE (w/o. CTM) exhibited much better average latency than ZeeStar and HLF during the no-resubmission attack. They efficiently detected and rejected malicious attackers, preserving computation power for benign clients and mitigating latency impact.

Overall, the results in Figure 4 emphasize that GRACE is particularly suitable for privacy-sensitive applications involving malicious participants. It incorporates countermeasures against no-resubmission attacks, ensuring fair access, and reducing latency.

7.3 Performance under Diverse Applications

The *SmallBank* workload represents real-world applications like token transfers, focusing on homomorphic addition rather than multiplication. To fully benchmark the performance improvement brought by CTM on both addition and multiplication, we developed the second workload introduced in §7, and conducted three experiments to evaluate the performance of GRACE and GRACE (w/o. CTM): (1) (**Add**) all transactions only invoked SC_A; (2) (**Mul**) all transactions only invoked SC_M; and (3) (**Add-Mul**) 50% of transactions invoked SC_A, while the other 50% invoked SC_M.

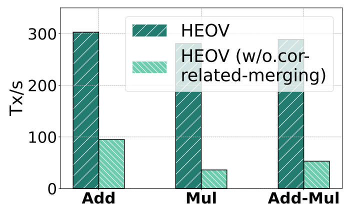


Figure 5: GRACE's optimization improvement under different combinations of HE computation.

Results in Figure 5 show that GRACE's CTM protocol supports both addition and multiplication operations, making it ideal for applications that require one or both of these types of operations, such as cryptocurrency [19] and finance [47]. Figure 5 also demonstrates that GRACE significantly outperformed GRACE (w/o. CTM) in all three experiments, with throughput enhancements of 3.2x, 5.4x, and 7.8x for addition-only, multiplication-only, and hybrid applications, respectively. These results highlight the effectiveness of GRACE's CTM protocol, particularly when both addition and multiplication are involved. Additionally, the results suggest that the CTM protocol excels when multiplication is the major HE computation of the business logic.

In summary, GRACE's CTM protocol has delivered substantial performance advantages to GRACE without altering the transaction semantics. This breakthrough enables GRACE to safeguard the privacy of sensitive user data with cryptographic guarantees while maintaining high performance.

7.4 Lessons Learned

GRACE has two limitations. First, GRACE is designed specifically for blockchain applications involving conflicting transactions, such as token transfers [19]. It employs the CTM protocol that effectively reduces the rate of conflicting aborts and minimizes the computational overhead of expensive HE operations. However, for conflict-free applications, GRACE performs similarly to the baseline. Note that conflicts are common in typical real-world blockchain applications, and achieving conflict-free applications would require significant modifications to the application logic, such as rewriting the smart contract using CRDT [36]. This may not be feasible for widely used blockchain applications, such as financial trading [36].

Second, the current implementation of GRACE only supports HE computations on bounded unsigned integers due to its adoption of the BFV scheme [12, 21]. However, incorporating more capable FHE schemes like CKKS [14]) would enable GRACE to easily support HE computations on floating-point numbers. This would broaden the range of applications that GRACE can accommodate, enhance its flexibility, and improve its usability for real-world blockchain applications.

8 Conclusion

We present GRACE, the first high-performance confidential permissioned blockchain. GRACE integrates fully homomorphic encryption with a novel CTM protocol to significantly lower the cost of ensuring data confidentiality. GRACE also tailors non-interactive zero-knowledge proofs by leveraging the endorsement mechanism of permissioned blockchains, resulting in lightweight transaction result verification. Extensive evaluation demonstrates that GRACE achieves superior performance compared to baselines while maintaining data confidentiality, making it perfect for privacy-sensitive blockchain applications that require both high throughput and low latency.

References

[1] Ittai Abraham, Guy Gueta, and Dahlia Malkhi. Hot-stuff the linear, optimal-resilience, one-message BFT devil, 2018.

[2] Farhana Aleen and Nathan Clark. Commutativity analysis for software parallelization: letting program transformations see the big picture. *ACM Sigplan Notices*, 44(3):241–252, 2009.

[3] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. Caper: a cross-application permissioned blockchain. *Proceedings of the VLDB Endowment*, 12(11):1385–1398, 2019.

[4] Mohammad Javad Amiri, Boon Thau Loo, Divyakant Agrawal, and Amr El Abbadi. Qanaat: A scalable multi-enterprise permissioned blockchain system with confidentiality guarantees. *arXiv preprint arXiv:2107.10836*, 2021.

[5] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, New York, NY, USA, 2018. ACM.

[6] Elli Androulaki, Christian Cachin, Angelo De Caro, and Eleftherios Kokoris-Kogias. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In *Computer Security: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I 23*, pages 111–131. Springer, 2018.

[7] Nick Baumann, Samuel Steffen, Benjamin Bichsel, Petar Tsankov, and Martin T. Vechev. zkay v0.2: Practical data privacy for smart contracts, 2020.

[8] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304, 2015.

[9] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. State machine replication for the masses with bft-smart. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362, 1730 Massachusetts Ave., NW Washington, DCUnited States, 2014. IEEE, IEEE Computer Society.

[10] Alysson Bessani, João Sousa, and Eduardo E.P. Alchieri. State machine replication for the masses with bft-smart. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362, 2014.

[11] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349, New York, NY, USA, 2012. ACM.

[12] Zvika Brakerski. 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*, pages 868–886, Heidelberg, 2012. Springer, Springer Berlin.

[13] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, New York, NY, USA, 1999. ACM.

[14] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 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*, pages 409–437, Hong Kong, China, 2017. Springer, Springer.

[15] consensys. Gnark, 2023.

[16] Thi Van Thao Doan, Mohamed-Lamine Messai, Gérald Gavin, and Jérôme Darmont. A survey on implementations of homomorphic encryption schemes. *The Journal of Supercomputing*, pages 1–42, 2023.

[17] Docker. Swarm mode overview.

[18] Jacob Eberhardt and Stefan Tai. Zokrates-scalable privacy-preserving off-chain computations. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1084–1091, Halifax, Nova Scotia, Canada, 2018. IEEE, IEEE.

[19] Ethreum. Erc-20 token standard, 2023.

[20] HyperLedger Fabric. 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.

[21] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144.

[22] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on computing*, 29(1):1–28, 1999.

[23] The Hyperledger Foundation. Private data collections: A high-level overview – hyperledger foundation, Oct 2018.

[24] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[25] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. Fastfabric: Scaling hyperledger fabric to 20 000 transactions per second. *International Journal of Network Management*, 30(5):e2099, 2020.

[26] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[27] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 581–612, Cham, 2017. Springer International Publishing.

[28] H-Store. H-store: Smallbank benchmark, 2013.

[29] Change Healthcare. Change healthcare case study, 2023.

[30] Hyperledger-TWGC. Hyperledger-twgc/tape: A simple traffic generator for hyperledger fabric, 2023.

[31] Hui Kang, Ting Dai, Nerla Jean-Louis, Shu Tao, and Xiaohui Gu. Fabzk: Supporting privacy-preserving, auditable smart contracts in hyperledger fabric. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 543–555, Portland, Oregon, USA, 2019. IEEE, IEEE.

[32] Medicalchain. Medicalchain. https://medicalchain.com, 2022.

[33] Andreas V Meier. The elgamal cryptosystem. In *Joint Advanced Students Seminar*, 2005.

[34] Christian Vincent Mouchet, Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Lattigo: A multiparty homomorphic encryption library in go. In *Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, pages 6. 64–70, ., 2020. HomomorphicEncryption.org.

[35] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[36] Pezhman Nasirifard, Ruben Mayer, and Hans-Arno Jacobsen. Fabriccrdt: A conflict-free replicated datatypes approach to permissioned blockchains. In *Proceedings of the 20th International Middleware Conference*, pages 110–122, 2019.

[37] NCCGroup, 2016.

[38] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[39] Ji Qi, Xusheng Chen, Yunpeng Jiang, Jianyu Jiang, Tianxiang Shen, Shixiong Zhao, Sen Wang, Gong Zhang, Li Chen, Man Ho Au, and Heming Cui. Bidl: A high-throughput, low-latency permissioned blockchain framework for datacenter networks. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 18–34, New York, NY, USA, 2021. Association for Computing Machinery.

[40] Pingcheng Ruan, Dumitrel Loghin, Quang-Trung Ta, Meihui Zhang, Gang Chen, and Beng Chin Ooi. A transactional perspective on execute-order-validate blockchains. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 543–557, New York, NY, USA, 2020. Association for Computing Machinery.

[41] Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL, March 2022. Microsoft Research, Redmond, WA.

[42] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. Blurring the lines between blockchains and database systems: The case of hyperledger fabric. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 105–122, New York, NY, USA, 2019. Association for Computing Machinery.

[43] Ravital Solomon, Rick Weber, and Ghada Almashaqbeh. smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption. *Cryptology ePrint Archive*, 2021.

[44] Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin Vechev. Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 179–197, San Francisco, California, USA, 2022. IEEE.

[45] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. Zkay: Specifying and enforcing data privacy in smart contracts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1759–1776, New York, NY, USA, 2019. Association for Computing Machinery.

[46] Samuel Steffen, Benjamin Bichsel, and Martin Vechev. Zapper: Smart contracts with data and identity privacy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2735–2749, 2022.

[47] taXchain. taxchain provides a faster, better, cheaper way to complete eu tax forms using hyperledger fabric, 2023.

[48] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. Sok: Fully homomorphic encryption compilers. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1092–1108, San Francisco, CA, USA, 2021. IEEE.

[49] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

[50] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[51] zcash. What are zk-snarks?, Jun 2022.