Fides: Scalable Censorship-Resistant DAG Consensus via Trusted Components

Shaokang Xie[1], Dakai Kang[1], Hanzheng Lyu[2], Jianyu Niu[3], Mohammad Sadoghi[1]
[1]Exploratory Systems Lab, Department of Computer Science, University of California, Davis
[2]School of Engineering, University of British Columbia (Okanagan campus)
[3]Department of Computer Science Engineering, Southern University of Science and Technology

Abstract

Recently, consensus protocols based on Directed Acyclic Graph (DAG) have gained significant attention due to their potential to build robust blockchain systems, particularly in asynchronous networks. In this paper, we propose Fides, an asynchronous DAG-based BFT consensus protocol that leverages Trusted Execution Environments (TEEs) to tackle three major scalability and security challenges faced by existing protocols: (i) the need for a larger quorum size (*i.e.*, at least 3x larger) to tolerate Byzantine replicas, (ii) high communication costs and reliance on expensive cryptographic primitives (*i.e.*, global common coin) to reach agreement in asynchronous networks, and (iii) poor censorship resilience undermining the *liveness* guarantee. Specifically, Fides adopts four trusted components—*Reliable Broadcast*, *Vertex Validation*, *Common Coin*, and *Transaction Disclosure*—within TEEs. Incorporating these components enables Fides to achieve linear message complexity, guaranteed censorship resilience, 2x larger quorum size, and lightweight common coin usage. Besides, abstracting these essential components rather than porting the entire protocol into TEE can significantly reduce the Trusted Computing Base (TCB). Experimental evaluations of Fides in local and geo-distributed networks demonstrate its superior performance compared to established state-of-the-art protocols such as Tusk, RCC, HotStuff, and PBFT. The results indicate that Fides achieves a throughput of 400k transactions per second in a geo-distributed network and 810k transactions per second in a local network. Our analysis further explores the protocol's overhead, highlighting its suitability and effectiveness for practical deployment in real-world blockchain systems.

1 Introduction

Recently, the surging popularity of blockchain technologies and Web3 have highlighted the importance of Byzantine Fault Tolerant (BFT) consensus, which enables replicas to agree on an ever-growing sequence of transactions even in the presence of some Byzantine replicas. Traditional BFT consensus protocols, such as PBFT [8], are predominantly designed for partial synchronous networks [9], where messages between honest replicas are guaranteed to be delivered within a fixed upper bound Δ after an unknown Global Stabilization Time (GST). However, there is uncertainty about when GST occurs and the variability in network conditions before GST. As a result, optimism in networks causes significant inefficiencies in designs, causing performance to degrade considerably, especially in blockchain environments with geographically dispersed replicas and unstable network conditions [10–14]. For instance, studies have shown that partially synchronous BFT protocols can struggle to reach consensus on transactions (*i.e.*, *liveness* violation) in asynchronous networks [15–19].

To enable BFT consensus to thrive in unstable networks, a series of fully asynchronous protocols [16–21] are proposed. These protocols can work (*i.e.*, ensuring *safety* and *liveness*) without any reliance on network synchrony, making them robust and suitable for mission-critical applications like Decentralized Finance (DeFi). Among them, a new class of asynchronous Directed Acyclic Graph (DAG)-based protocols [19–21] gained significant popularity due to their efficiency, robustness, and simplicity. Notable examples include DAG-Rider [19] and Bullshark [20]. These protocols divide operations into two phases: data dissemination and consensus. In the dissemination phase, replicas use reliable broadcast (RB) to distribute proposals (i.e., DAG nodes), which encapsulate pending transactions and references to preceding proposals (i.e., DAG edges), forming the DAG structure. In the consensus phase, replicas agree on specific prefixes of the DAG, producing a globally ordered sequence of transactions. DAG-based BFT protocols are resource-efficient since they can fully utilize each replica's bandwidth by producing proposals.

Despite their promising features, DAG-based BFT protocols still fall short in scalability and censorship resilience, limiting their adoption in large-scale geo-distributed Web3 applications. First, to tolerate Byzantine behaviors from f replicas, DAG-based BFT protocols require a larger quorum size of $n = 3f + 1$ replicas. Analytical and empirical results demonstrate that the system performance (*i.e.*, throughput and latency) will degrade when the number of replicas reaches hundreds (Sec. 7). Second, they suffer from high communication costs and the reliance on expensive cryptographic primitives to reach agreement in asynchronous networks. Specifically, they use n RB in parallel to produce proposals, with each involving multiple rounds of message exchanges, resulting in quadratic communication overhead $O(n^2)$. Meanwhile, the common coin realized by threshold signatures may lead to potential safety risks and significantly degrade performance [22, 23]. Third, existing protocols fail to address censorship attacks. Specifically, attackers can delay the broadcasts of undesired transactions in the dissemination phase, or manipulate the ordering of proposals in the consensus phase to prevent certain transactions from being output.

In this paper, we introduce Fides, an asynchronous DAG-based BFT consensus protocol that leverages Trusted Execution Environments (TEEs) to address the aforementioned scalability and security challenges. TEEs provide secure hardware enclaves for executing critical operations in isolation, ensuring the integrity and confidentiality of running applications. By abstracting and tailoring essential trusted components within TEEs for DAG-based protocols,

Fides can effectively mitigate malicious behavior from Byzantine replicas. As a result, this approach can reduce the quorum size, minimize message exchanges for lower message complexity, eliminate reliance on expensive cryptographic primitives, and enhance resilience against censorship attacks.

To this end, we propose four trusted components—*Reliable Broadcast*, *Vertex Validation*, *Common Coin*, and *Transaction Disclosure*—within TEEs. These components are essential for maintaining system consistency and achieving consensus even in asynchronous environments. Additionally, limiting the reliance to these components helps reduce the Trusted Computing Base (TCB), improving both security and efficiency.

We built end-to-end prototypes of Fides using the open-source Apache ResilientDB (Incubating) platform [6, 24]. We use Open Enclave SDK [7] to develop trusted components on *Intel SGX* [25]. We conducted extensive experiments in the public cloud to evaluate and compare Fides with several representative counterparts including Tusk [26], RCC [27], HotStuff [28], and PBFT [8]. Experimental results of Fides in local and geo-distributed networks demonstrate its superior performance. The results indicate that Fides achieves a throughput of 400k transactions per second in a geo-distributed network and 810k transactions per second in a local network. Our analysis further explores the protocol's overhead, highlighting its suitability and effectiveness for practical deployment in real-world blockchain systems.

Contributions. We propose Fides, an asynchronous DAG-based consensus protocol that integrates TEEs:

- We improve *scalability* by reducing the replication factor from $n = 3f+1$ to $n = 2f+1$, while introducing novel commitment rules specifically designed for asynchronous DAG-based protocols.
- We enhance *compute-efficiency* by minimizing reliance on crypto-heavy computations such as threshold signatures and improve *network-efficiency* by minimizing the communication phases required in reliable broadcast.
- We ensure *censorship-resistance* through a novel method of not revealing transaction content a priori.
- We provide a *robust design* that employs the causal structure of DAG protocols without the need for intricate recovery mechanisms even in asynchronous networks.

2 Background

This section provides essential background on asynchronous DAG-based protocols and TEE with mitigation methods.

2.1 Asynchronous DAG Protocols

Asynchronous DAG-based protocols achieve consensus among replicas even in asynchronous networks, and their leaderless design significantly boosts system throughput. DAG-based consensus protocols [19, 20, 26] operate in a round-by-round manner. In each round, every replica proposes a block containing a batch of transactions and each block references blocks from the previous round. These blocks act as *vertices*, and the reference links between them serve as *edges*, collectively forming a DAG. Reliable broadcast algorithms disseminate these vertices, ensuring that all replicas eventually achieve a consistent view of the DAG (Sec. 3.3).

Rounds are grouped into consecutive waves, with each wave comprising a specific number of rounds. During each wave, a leader vertex is elected using a common coin (Sec. 3.3), which produces the same value across all replicas. By ordering all vertices based on the elected leader vertices, the common coin ensures that replicas commit transactions in a consistent order. This approach eliminates the need for additional communication overhead to achieve transaction ordering.

However, asynchronous DAG-based consensus protocols face scalability and deployment challenges due to quadratic communication overhead from reliable broadcasts, performance bottlenecks [23] and safety risks[22] from threshold-based common coin mechanisms, and vulnerability to censorship attacks like front-running [29]. Existing DAG-based protocols [19–21, 26, 30–32] fail to address these issues effectively, resulting in notable vulnerabilities in security and fairness.

2.2 Trusted Execution Environment

Trusted Execution Environments (TEEs) provide an isolated and secure enclave within replicas to execute critical operations while protecting sensitive data. By ensuring confidentiality and integrity, TEEs effectively prevent Byzantine replicas from tampering with messages, forging data, or accessing confidential information, thereby enhancing the security and reliability of consensus protocols.

TEEs enhance the security and efficiency of consensus protocols by enabling isolated and tamper-resistant execution of sensitive computations. TEEs can generate cryptographic certificates that attest to the integrity and authenticity of the computation results, allowing external parties to verify the correctness and authenticity of the result. This ensures secure, trustworthy processing in environments with limited trust, enhancing both reliability and confidentiality.

Ensuring censorship resistance is essential in blockchain consensus protocols to prevent malicious replicas from selectively including, excluding, or reordering transactions for unfair advantages. One effective approach is encrypting transaction content, which keeps transaction details hidden from replicas until they are finalized. TEEs provide a secure and isolated environment for reliable management of transaction encryption and decryption. This ensures that transaction content remains confidential and tamper-proof throughout the consensus process. Compared to other mechanisms, TEE-based encryption offers a lightweight and efficient solution, minimizing computational and communication overhead while maintaining the integrity and fairness of the protocol.

3 Model and Problem Formulation

This section defines the system and threat models, formulates the BFT consensus problem, and introduces some building components, such as reliable broadcast and the common coin.

3.1 System Model

Replicas with TEEs. We consider a network Π consisting of $n = 2f + 1$ replicas, denoted by $\Pi = \{p_1, p_2, ..., p_n\}$. Each replica p_i ($1 \leq i \leq n$) has a unique identity known to all others within the network. Every replica runs delicately on an SGX-enabled machine. Among these replicas, up to f may be Byzantine, which may exhibit

arbitrary behaviors. The remaining replicas are honest and strictly follow the protocol.

Threat Model. Fides adopts the threat model commonly used in prior TEE-assisted BFT protocols [33–37]. TEEs are assumed to guarantee integrity, confidentiality and secure remote attestation, ensuring unforgeable cryptographic proof that specific programs are executed within the enclave [2, 3]. At Byzantine replicas, all components outside the TEE are considered vulnerable to compromise. An adversary is assumed to have root-level access to the host machine, enabling control over the network. However, the TEE itself is assumed to remain secure and untamper, maintaining its core trusted functionalities. Additionally, any attempts to tamper with the TEE's output are detectable because all outputs are cryptographically signed by the TEE.

Consistent with prior works [33–37], Fides adopts established mitigation solutions to address the prevalent TEE-related attacks (*e.g.*, rollback attacks and side-channel attacks), which have been extensively explored in the literature [4, 38–44].

Authentication. Each replica has a public/private key pair (pk_i, sk_i) established through a Public-Key Infrastructure (PKI). The private keys of replicas are securely stored within their respective TEEs, while the corresponding public keys are across the network. This PKI framework enables secure message signing, verification, encryption, and the generation of trusted operation certificates.

For enhanced confidentiality and efficiency, replicas also maintain a shared RSA key pair, termed the *main key*. The private key resides securely within the TEE, and the public key is disseminated across the network. Clients may generate symmetric AES *session keys* to encrypt their transaction data, sharing these keys with replicas as detailed in Sec. 6.3.

We assume that replicas are computationally bounded and cannot break public-key cryptography. Further, we assume the existence of collision-resistant hash function $H(x)$, where for a computationally bounded replica, it is impossible to find a value x', such that $H(x) = H(x')$ [45].

Asynchronous Network. We assume the existence of an asynchronous network [46]: Messages sent between replicas experience arbitrary but finite delays. In other words, messages are never lost, but there are no guarantees of delivery time or order.

3.2 Problem Formulations

We consider a DAG-based *BFT Consensus*, which allows replicas to agree on a sequence of client transactions. Each replica continuously proposes transactions, with each transaction tx assigned a consecutive round number r. The round number r is used to distinguish between transactions proposed by the same replica. Each transaction corresponds to a vertex in the DAG, and each vertex references $f + 1$ vertices from the previous round number, thus creating a DAG structure. Each correct replica commits a transaction tx with a unique sequence number sn. A transaction is irreversible upon commitment, and the results are sent back to the clients. A BFT Consensus protocol satisfies the following properties:

- **Safety:** If two correct replicas p_i and p_j commit two transactions tx_1 and tx_2 with sequence number sn, then $tx_1 = tx_2$.
- **Liveness:** If a transaction tx is sent to and proposed by at least one correct replica, then every correct replica will eventually commit tx.

In addition to ensuring correctness, we also aim to mitigate censorship attacks by encrypting transactions. This approach prevents leaders from manipulating transaction ordering for personal gain. Specifically, we introduce the following property:

- **Censorship-Resistance:** A transaction remains concealed from all replicas until it is delivered.

3.3 Building Blocks

Reliable Broadcast. Reliable broadcast is a fundamental primitive in distributed systems to ensure consistent delivery of messages to all replicas, even in the presence of Byzantine faults. Formally, a sender replica p_i broadcasts a message m in a round $r \in \mathbb{N}$ by invoking $R\_Bcast_i(m, r)$. Upon receiving and handling the broadcast, every replica p_k outputs $R\_Deliver_k(m, r, p_i)$, where m is the broadcast message, r is the round number and p_i is the replica that initiated $R\_Bcast_i(m, r)$. Reliable broadcast guarantees the following properties:

- **Validity:** If a correct replica p_j invokes $R\_Bcast_j(m, r)$, then every correct replica p_i eventually invokes $R\_Deliver_i(m, r, p_j)$.
- **Integrity:** A replica delivers a message m at most once, and only if m was actually broadcast by replica p_j through $R\_Bcast_j(m, r)$.
- **Agreement:** If any correct replica p_i invokes $R\_Deliver_i(m, r, p_j)$, then all correct replicas eventually invoke $R\_Deliver_k(m, r, p_j)$.

Fides adopts a TEE-assisted reliable broadcast, which can ensure the above three properties with the minority of Byzantine replicas. Except for higher tolerance, the TEE-assisted Reliable Broadcast (Sec. 5.1) only has one phase of message exchanges, significantly reducing latency and complexity.

Common Coin. The common coin (also known as a global random coin) provides a mechanism for random decision-making among a set of participants, which is the essential component of asynchronous consensus protocols [15, 18, 19]. In Fides, the common coin is utilized in every wave $w \in \mathbb{N}$. A replica $p_i \in \Pi$ invokes the coin by calling $\textsc{ChooseLeader}_i(w)$. This function eventually returns a randomly chosen replica p_j, which is designated as the leader of wave w. The common coin provides the following guarantees:

- **Agreement:** If two correct replicas invoke $\textsc{ChooseLeader}_i(w)$ and $\textsc{ChooseLeader}_j(w)$ with respective return values p_1 and p_2, then $p_1 = p_2$.
- **Termination:** If at least $f+1$ replicas invoke $\textsc{ChooseLeader}(w)$, then every correct replica invoking the function will eventually receive a result.
- **Unpredictability:** As long as fewer than $f + 1$ replicas have invoked $\textsc{ChooseLeader}(w)$, the outcome is computationally indistinguishable from a random value, except with negligible probability ϵ.
- **Fairness:** The coin introduces unbiased randomness into the leader selection process, ensuring fairness. For any wave $w \in \mathbb{N}$ and for any replica $p_k \in \Pi$, the probability of selecting p_k as the leader is uniformly distributed, specifically $Pr[Leader = p_k] = \frac{1}{n}$, where $n = |\Pi|$. The cryptographic randomness used by the

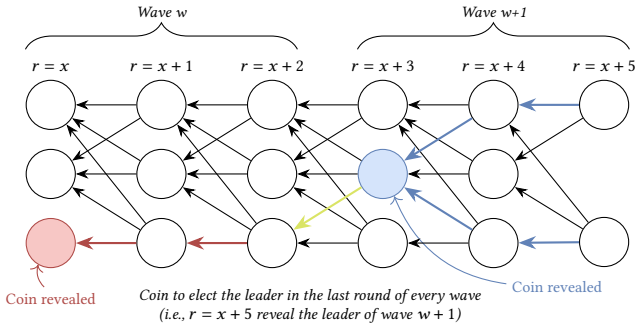


Figure 1: Example of commit rule in Fides. If the leader has less than $f + 1$ references from its third round (red in the figure), they are ignored; otherwise (blue in the figure), the algorithm searches the causal DAG to commit all preceding leaders (dark yellow and red in the figure) and totally orders the rest of the DAG by the predefined rule afterward.

coin ensures robustness against bias or manipulation, thereby maintaining the integrity of the leader selection process.

Unlike traditional common coin mechanisms [23] that use resource-intensive cryptographic techniques, Fides realizes a TEE-assisted common coin (Sec. 5.3). By leveraging TEEs, Fides eliminates the computational overhead of traditional cryptographic techniques while maintaining the guarantees of all the properties.

4 Fides Protocol Overview

We provide a high-level overview for Fides, an asynchronous DAG-based protocol that tolerates f Byzantine fault replicas out of $n = 2f + 1$ replicas, as illustrated in Fig. 1. Unlike previous DAG-based BFT protocols [19–21, 26, 47], Fides reduces the replication factor while introducing a three-round wave structure instead of the traditional two-round design (e.g., Bullshark). This structure enhances fault tolerance, supports leader election, and ensures a stable and well-ordered DAG. This three-round design is carefully chosen to optimize both causal consistency and commitment efficiency.

Generally, client transactions in Fides are organized causally within a DAG, where vertices represent batches of transactions, and edges encode causal dependencies. The DAG grows causally round-by-round, ensuring dependencies are preserved across rounds. Fides operates in two distinct layers:

- **Dissemination layer:** This layer is responsible for constructing the DAG. Replicas utilize the *TEE-Assisted Reliable Broadcast* mechanism (Sec. 5.1) to disseminate vertices reliably. To ensure consistency of the DAG, the *TEE-Assisted Vertex Validation* mechanism (Sec. 5.2) is employed to validate the references between vertices. Following specific rules (Sec. 6.1), replicas construct their local DAG accordingly.
- **Ordering layer:** This layer is responsible for determining a consistent order of the transactions (vertices). Each replica independently interprets its causally ordered local DAGs, leveraging the *TEE-Assisted Common Coin* mechanism (Sec. 5.3) to establish a definitive and consistent order of vertices across replicas (Sec. 6.2) without requiring additional communication.

Additionally, to address potential censorship attacks, clients can utilize the *TEE-Assisted Transaction Disclosure* mechanism (Sec. 5.4)

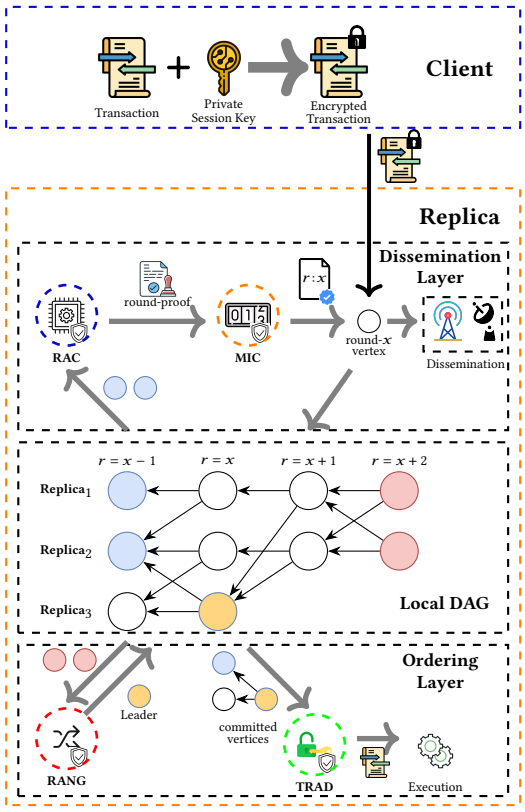


Figure 2: Overview Architecture of Fides

to keep transaction content encrypted throughout the entire consensus process (Sec. 6.3).

4.1 Consensus Flow

We present a high-level flow to process client transactions (Fig. 2).

❶ **Transaction submission.** Clients submit transactions to any known replica in Fides. To mitigate the risk of censorship attacks, clients can optionally encrypt transaction content by utilizing the *TEE-Assisted Transaction Disclosure* service. The receiving replica verifies the transaction and forwards it for further processing.

❷ **Vertex construction.** Once a replica enters a new round, it constructs a vertex containing pending transactions, a validity proof, and associated metadata. The validity proof is generated by the *TEE-Assisted Vertex Validation* service based on $f + 1$ references to vertices from the previous round.

❸ **Vertex dissemination.** After constructing the vertex, the replica disseminates the vertex to all other replicas using the *TEE-Assisted Reliable Broadcast* service, which efficiently ensures consistency.

❹ **DAG construction.** Upon receiving a vertex from others, the replica verifies the presence of all referenced vertices in its local DAG and validates the counter value. If any referenced vertices are missing, the replica waits until they are received and successfully validated. Once all dependencies are resolved and the vertex passes verification, it is then added to the local DAG.

❺ **Wave leader election.** Each replica locally interprets its view of the DAG to determine the order of vertices. Fides operates the

DAG wave-by-wave. At the third round in a wave, replicas utilize the shared randomness provided by the *TEE-Assisted Common Coin* service to collaboratively elect a leader for that wave, facilitating consistent ordering.

❻ **Vertices ordering and commitment.** Once the leader is elected and gains sufficient support—defined as having at least $f+1$ vertices from the third round of the same wave with a path to the leader—the leader vertex, along with all vertices in its causal history, is then committed in some predefined order (*e.g.*, depth-first search).

❼ **Transaction reveal and execution.** Immediately after the vertices are committed, the *TEE-Assisted Transaction Disclosure* service reveals the encrypted content of all transactions within the committed vertices, if necessary. The transactions are then executed, and the results are replied to the client in response.

5 TEE-Assisted Components

Fides leverages TEEs to enhance efficiency, mitigate censorship attacks, and optimize the system threshold of asynchronous DAG-based BFT consensus. Fides adopts four trusted components realized by TEEs including *Reliable Broadcast*, *Vertex Validation*, *Common Coin*, and *Transaction Disclosure*, implemented through trusted components: ***M**onotonically **I**ncreasing **C**ounter (**MIC**)*, ***R**ound **A**dvancement **C**ertifier (**RAC**)*, ***RA**ndom **N**umber **G**enerator (**RANG**)*, and ***T**ransaction **RSA D**ecryptor (**TRAD**)*.

5.1 TEE-Assisted Reliable Broadcast

Trusted Component - MIC. This component is designed to effectively address equivocation in the reliable broadcast. Operating within the TEE, it facilitates the implementation of an efficient reliable broadcast in Fides. In each round of the DAG, *MIC* assigns a unique, verifiable identifier to every message using a monotonically increasing counter. The integrity of these counter values is guaranteed by the TEE, protecting them from tampering by malicious hosts. This unique identification prevents equivocation, a critical vulnerability that can lead to double spending, state inconsistencies, or even disruptions in the consensus process. The *MIC* provides the following interface to support *TEE-Assisted Reliable Broadcast*:

- $\langle counter, \phi \rangle \leftarrow$ GetCounter*(round-cert, v)*: This function accepts a round certificate *round-cert* (discussed in Sec. 5.2) and a vertex v that the counter value is attached to. The *round-cert* represents the replica's eligibility to advance to the next round and obtain a counter. Upon validation, the function generates a unique counter value and attaches it to the vertex v, along with a corresponding attestation ϕ. The attestation ϕ includes a cryptographic proof ensuring the authenticity of the counter value. Afterwards, the local counter value is incremented to prepare for the next request. Since the counter value synchronizes with the replica's progress in the DAG's rounds, the binding is inherently clear and can be easily verified by other replicas.

Reliable Broadcast using *MIC*. Reliable broadcast, as introduced in Sec. 3.3, is a fundamental primitive that ensures consistent delivery of messages to all replicas, even in the presence of Byzantine faults. It guarantees several properties: Validity, Integrity and Agreement. Traditional reliable broadcast protocols, such as Bracha's Reliable Broadcast [48], require three rounds of all-to-all communication and a replication factor of $n = 3f + 1$ to tolerate f Byzantine faults. While reliable, these protocols impose substantial communication overhead. By integrating *MIC*, Fides optimizes the reliable broadcast process, effectively reducing both communication overhead and the quorum threshold to $n = 2f + 1$. Each replica generates and broadcasts a single valid proposal per round, authenticated by *MIC* using its synchronized monotonic counter.

TEE-Assisted Reliable Broadcast:

1: **function** $R\_Bcast(v)$ **do**
2: **Step 0 (Preparation):**
3: $v.counter = MIC.getCounter(round\text{-}cert, v)$
▷ Generate a unique counter value and corresponding certificate
4: **Step 1 (Proposal):**
5: **If** Replica$_i$ is the Initiator **then**
6: Replica$_i$ broadcast $\langle proposal, v \rangle$
7: **Step 2 (Preservation):**
8: **Upon receiving** a valid $\langle proposal, v \rangle$, broadcast $\langle echo, v \rangle$
9: **Step 3 (Reply):**
10: Upon receiving at least 1 $\langle echo, v \rangle$, $R\_Deliver(v)$ if v is valid.

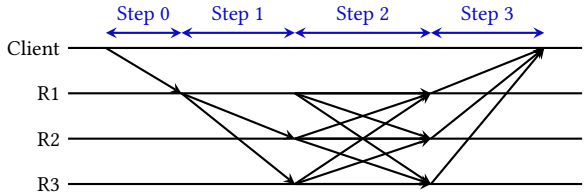


Figure 3: TEE-Assisted Reliable Broadcast with *MIC*

Specifically, the detailed process of our *TEE-Assisted Reliable Broadcast* is shown in Fig. 3, and we also prove its correctness in Sec. A.1. The incorporation of *MIC* introduces several key optimizations to the reliable broadcast process:

- Optimized Quorum Threshold: Fides achieves improved fault tolerance with reducing the replication factor from $n = 3f + 1$ to $n = 2f + 1$, while still tolerating f Byzantine faults.
- Reduced Communication Overhead: The confidentiality of TEE simplifies the equivocation validation, thus eliminating the need for multiple all-to-all communication rounds traditionally required to prevent equivocation.

Initialization. The round counter of each replica is set to zero within the TEE, ensuring a secure and tamper-proof starting state.

5.2 TEE-Assisted Vertex Validation

Trusted Component - RAC. This component plays a pivotal role in *TEE-Assisted Vertex Validation* during DAG construction. Leveraging TEE provides a secure mechanism to validate referenced vertices and determine a replica's eligibility for round advancement. After successful validation, *RAC* issues a round certificate that guarantees all vertices are consistent with the causal history of the DAG. Specifically, when a replica generates a new vertex, it must reference a quorum of at least $f + 1$ vertices from the previous round (see Sec. 6.1). The *RAC* is responsible for systematically validating these references. Specifically, it confirms that the quorum includes at least $f + 1$ vertices from unique replicas that have

been delivered via *TEE-Assisted Reliable Broadcast*. Additionally, it verifies the validity of each referenced vertex (*e.g.*, ensuring that the associated metadata meets the expected criteria) and excludes any stale or invalid references. Upon successful validation, the *RAC* issues a round certificate that certifies the replica's eligibility to advance to the next round. This certificate acts as cryptographic proof of the correctness and consistency of the referenced vertices.

This *TEE-Assisted Vertex Validation* process, enforced by the *RAC*, strengthens the system's resilience against Byzantine faults. It ensures that all newly created vertices are consistent with the DAG's causal structure and effectively prevents malicious activities, such as referencing invalid or insufficient vertices.

Functional Interface. The *RAC* provides the following core functions to support vertex validation and round advancement:

- $\langle$*round-cert*$\rangle \leftarrow$ ValidateVertices(*vertexList*): This function accepts a list of vertices that the replica intends to reference. It verifies if they are all reliably delivered, if they meet quorum requirements, and if the associated metadata meets the required criteria. It returns a round certificate referred to as *round-cert* if all the validations are deemed valid.

Initialization. At the beginning of the protocol, each replica starts with a genesis vertex. These genesis vertices enable the system to generate a valid *round-cert* for the first round.

5.3 TEE-Assisted Common Coin

Trusted Component - RANG. This component is central to the *TEE-Assisted Common Coin* mechanism, which ensures efficient and secure generation of global random numbers essential for leader election and progress in asynchronous consensus protocols. By utilizing a shared random seed across replicas, *RANG* ensures a uniform sequence of random numbers, with its internal state securely isolated within the TEE to prevent unauthorized access.

As noted in Sec. 1, conventional implementations often rely on threshold cryptography, which incurs significant computational overhead [23]. The *TEE-Assisted Common Coin*, implemented via *RANG*, addresses this challenge by providing a lightweight, efficient alternative that minimizes the computational overhead. However, Byzantine replicas may attempt to exploit *RANG* by generating multiple random numbers and predicting future values, potentially causing liveness issues. To mitigate this, access to *RANG* is gated by strict eligibility checks. Specifically in Fides, any replica requesting a new random number from *RANG* must provide a quorum proof, which must include the *round-cert* generated by *RAC* for round r. This *round-cert* value verifies the valid participation of $f + 1$ replicas in certain rounds, ensuring that the requested random number is generated only when the quorum conditions are satisfied. This mechanism guarantees that *RANG* operates securely and only under necessary circumstances.

In Fides's commit rule(Sec. 6.2), the common coin is typically invoked every three rounds for each replica. To maintain alignment, *RANG* maps the sequence number to the round number and generates a valid random number only when this mapping satisfies predefined conditions. This mechanism ensures that replicas only access random values at the appropriate stage, preventing Byzantine replicas from influencing future leader elections. Moreover, as introduced in Sec. 3.3, this component functions as a common coin, and we have formally proven that it satisfies the properties of a common coin in Sec. A.2.

Functional Interface. The *RANG* has the following interfaces:

- SetSeed(*seed*): This function accepts a newly generated seed, collaboratively established among all replicas, to reset the internal random seed of the Pseudo-Random Number Generator (PRNG) within the enclave.
- $\langle r, \phi \rangle \leftarrow$ Rand(*round-cert*): This function accepts two parameters: a *round-cert* for round r which is the final round of the wave. *round-cert* is a certificate issued by *RAC* that proves the participation of at least $f+1$ replicas in. The function validates the *round-cert* to ensure the replica's eligibility to access the random sequence. Upon successful validation, it generates and returns a random number r, computed as the output of the PRNG modulo, the total number of replicas n within the enclave, along with a signed certificate ϕ.

Initialization. At system startup, a Distributed Key Generation (DKG) protocol [49] is employed to produce a shared random seed for *RANG*, enabling the common coin functionality.

5.4 TEE-Assisted Transaction Disclosure

Trusted Component - TRAD. This component is central to Fides's *TEE-Assisted Transaction Disclosure* mechanism, ensuring transaction security by concealing transaction content until commitment, protecting sensitive information from premature exposure. This approach mitigates censorship attacks, where adversaries exploit transaction ordering for financial gain.

To address this challenge efficiently, we introduce two types of keys: the main key and the session key. Initially, Fides employs a shared RSA (asymmetric) key, referred to as the main key, across all replicas to manage the session keys securely. The private key of the main key is securely stored within the TEE, while the corresponding public key is published for client use. Clients requiring censorship protection should generate their own AES session key, which serves as a cryptographic key for encrypting and decrypting transaction data. Before utilizing this session key, the client needs to encrypt it using the public key of the main key and then submit it to the replicas as part of the transaction payload. Upon receiving and confirming the transaction, replicas decrypt the session key inside the TEE with a securely stored private key. Subsequently, the replicas respond to the client with the *sessionId* corresponding to the key. In this way, the session key is securely shared between the client and the TEEs of the replicas, maintaining confidentiality and protecting it from exposure to untrusted environments.

Within this framework, clients encrypt transaction content with the session key before submission. Replicas store the encrypted data until the corresponding vertex in the DAG is confirmed. At this stage, replicas verify decryption conditions, including the status of the *RANG* component and confirmation of the next-wave leader. If conditions are met, *TRAD* decrypts the transaction content using the session key, preventing premature disclosure. This delayed decryption mechanism, referred to as *TEE-assisted Transaction Disclosure*, prevents any premature knowledge of transaction details,

thereby reducing the risk of censorship attacks by keeping data invisible during the consensus process.

Functional Interface. The *TRAD* component provides the following core interfaces:

- $\langle pubkey, \phi \rangle \leftarrow$ GetPubKey(): This function returns the published public key used for encrypting client transactions.
- $\langle sessionId, \phi \rangle \leftarrow$ AddSessionKey($encryptedSessionKey$): This function accepts an encrypted session key generated by the client, securely stores it within the TEE, and returns a unique session ID associated with the stored key.
- $\langle$decrypted-block, $\phi \rangle \leftarrow$ Decrypt(*encrypted-block*): This function accepts a block of encrypted transactions. Prior to decryption, the function verifies the state of the *RANG* component within the TEE, specifically checking the call count and ensuring the vertex is within the causal history of the next-wave leader. Upon satisfying these conditions, the function decrypts the ciphertext with the corresponding session keys stored inside *TRAD* and returns the block with decrypted transactions.

Initialization. Replicas initialize a shared key pair for *TRAD*, ensuring secure management of the session keys. Once this setup completes, *TRAD* releases the corresponding public key, which clients use to encrypt their session keys.

6 Fides Design: Putting it all Together

In this section, we present the detailed design of Fides, a TEE-assisted BFT asynchronous consensus protocol based on a DAG structure. We detail the DAG construction, its commit rules, and the built-in mechanism for censorship resistance.

6.1 Dissemination Layer

Fides employs a DAG to capture and structure the results of participant communication. Transactions are organized into this DAG, where each vertex represents a batch of transactions and points to vertices from preceding rounds. The rounds are arranged sequentially, and the DAG grows incrementally as each round progresses.

DAG structure. To provide a comprehensive understanding of how is our DAG structure, we first provide pseudocode for our DAG structure in Fig. 4. Formally, each vertex v in the DAG is associated with: (i) a batch of transactions, (ii) some associated metadata (*e.g.*, round number, source, *MIC* value *etc.*), and (iii) references to vertices from earlier rounds. The references are indicated as edges in the DAG, and Fides distinguishes between two types of edges:

- **Strong Edges:** These connect a vertex v to at least $f + 1$ vertices from the immediately preceding round. Such references guarantee that v extends a sufficiently robust subset of the prior round's vertices, effectively preventing forks and equivocation while providing a foundation for the liveness guarantees (as detailed in Sec. A.4.2).
- **Weak Edges:** These optionally connect v to vertices from earlier rounds when no direct path exists through strong edges. Although not essential for immediate round progression, they ensure eventual inclusion and ordering of all vertices.

To ensure both safety and liveness, every round in Fides requires at least $f+1$ vertices to ensure proper resilience with a threshold of $n = 2f + 1$ (with f representing the number of Byzantine replicas and n the total number of replicas). These $f + 1$ vertices guarantee a sufficient number (at least 1) of honest replicas' participation in each round. Once a replica p_i has successfully received and verified at least $f + 1$ valid vertices from the current round r, it can advance to round $r+1$, generate a new vertex referencing these $f+1$ vertices in round r, and broadcast it, incrementally constructing the DAG. Additionally, to ensure the safe and efficient removal of outdated vertex records, we adopted the garbage collection mechanism from [26].

DAG construction. The systematic process of Fides's DAG construction is detailed in Fig. 5, offering a comprehensive overview of how vertices are created, verified, and integrated into the DAG.

The DAG construction begins with a genesis vertex at each replica and progresses through sequential rounds once at least $f + 1$ vertices have been observed in the current round (line 15). To guarantee the correctness of this progression, Fides employs the *TEE-Assisted Vertex Validation* with the trusted component *RAC*. Specifically, before p_i can move to the next round, it is required to invoke *RAC* to validate that the required quorum criteria are met (line 19), which includes verifying if the $f+1$ vertices originate from distinct replicas and have been reliably delivered. Upon successful validation, *RAC* issues a round certificate (*round-cert*), authorizing p_i to advance to the next round. Once advancing to the next round, p_i generates a new vertex (line 21) and disseminates it to all replicas utilizing the *TEE-Assisted Reliable Broadcast* mechanism (line 22). Together, these two TEE-Assisted mechanisms ensure that all newly created vertices are verifiably honest and reliably delivered with the DAG's causal structure, thus effectively enhancing DAG construction performance.

Since all the replicas progress independently, each non-faulty replica maintains its own local copy of the DAG. Upon reliably delivering a vertex, the replica should wait until all its referenced vertices have been added to the DAG before integrating the newly delivered vertex (line 9). Although local DAGs may temporarily diverge due to variations in message delivery times across replicas, the *TEE-Assisted Reliable Broadcast* mechanism guarantees that all non-faulty replicas will eventually receive the same sets of vertices (Lemma 1). The potential differences in the order of delivery do not affect the correctness of the DAG because the topological order of vertices is inherently defined by their references. As a result, the system eventually converges on a consistent, unified DAG view across all replicas. This convergence not only maintains the integrity of the DAG but also supports the deterministic progression of the protocol, even in the presence of Byzantine faults or network inconsistencies.

An example of the DAG construction is illustrated in Fig. 6. Each completed round contains between $f + 1$ and $n = 2f + 1$ vertices. Each vertex establishes connections to at least $f + 1$ vertices from the preceding round through strong edges (solid black lines). Additionally, vertices may connect to earlier vertices via weak edges if no alternative paths exist within the DAG (dash black lines). For instance, replica$_j$ may receive v_1 before v' due to network inconsistencies. However, according to the DAG construction rules, a vertex can be confirmed only if all its predecessors have already

Data Structure:
struct vertex v:
$v.round$ - the round of v in the DAG
$v.source$ - the replica that broadcast v
$v.block$ - a block of transactions
$v.strongEdges$ - a set of hashes of the vertices in $v.round - 1$ that represent *strong* edges
$v.weakEdges$ - a set of hashed of the vertices in rounds $< v.round - 1$ that represent *strong* edges
$v.counter$ - the *MIC* value and its certificate for the vertex
$DAG[]$ - An array of sets of vertices, initially:
$DAG_i[0]$:= predefined set of $f+1$ genesis vertices
$\forall j \geq 1 : DAG_i[j] := \{\}$
$blocksToPropose$ - A queue, initially empty, p_i enqueues valid blocks of transactions from clients

1: **function** PATH(v,u) **do**
2: **return** exists a sequence of $k \in \mathbb{N}$, vertices v_1,v_2,...,v_k s.t. $v_1 = v$,$v_k = u$,and $\forall i \in [2..k] : v_i \in \bigcup_{r \geq 1} DAG_i[r] \wedge (v_i \in v_{i-1}.weakEdges \cup v_{i-1}.strongEdges)$

3: **function** STRONGPATH(v,u) **do**
4: **return** exists a sequence of $k \in \mathbb{N}$, vertices v_1,v_2,...,v_k s.t. $v_1 = v$,$v_k = u$,and $\forall i \in [2..k] : v_i \in \bigcup_{r \geq 1} DAG_i[r] \wedge v_i \in v_{i-1}.strongEdges$

5: **function** VERIFYVERTEX(v) **do**
6: **return** the validity of the certificate in $v.counter$

Figure 4: Data Structures and basic utilities for replica$_i$

been added to the DAG. Thus, v_1 cannot be confirmed until v' is fully received and processed.

Conclusively, FIDES's DAG construction ensures that all replicas converge to a unified DAG view while maintaining integrity and consistency, establishing a foundation for the ordering layer to interpret and commit transactions based on this DAG.

Each replica then initializes the genesis vertex of its local DAG and proceeds with protocol execution.

6.2 Ordering Layer

Overview. Building on the DAG structure, FIDES is an innovative asynchronous Byzantine consensus algorithm that ensures safety and liveness under conditions of asynchrony. Upon interpreting the local DAG, FIDES requires no additional communication for commitment while significantly improving latency in typical operational scenarios.

FIDES organizes vertices into waves, ensuring leader commitment through quorum-based validation and recursive checks for causal consistency. By leveraging lightweight deterministic randomization (*TEE-Assisted Common Coin*) and requiring zero additional communication overhead, FIDES enables replicas to independently derive a global transaction order from their local DAG views, ensuring both safety and liveness under asynchronous conditions.

TEE-Assisted Commit Rule. The commit rule process in FIDES, as outlined in Fig. 7, begins with a DKG protocol [49], which initializes the *TEE-Assisted Common Coin* service with a shared random seed. This adaptively secure seed, stored confidentially within the trusted enclave, serves as the foundation for deterministic randomness.

DAG Construction (running at each replica$_i$) **:**
Local variables:
$r := 0$
$buffer := \{\}$
Suppose each replica has a genesis vertex v_0

7: **event** $R\_Deliver_i(v, round, p_k)$ **do**
▷ Receive a vertex from p_k through reliable broadcast
8: **if** VERIFYVERTEX(v) **then**
▷ Verify the vertex before trying to add to the DAG
9: **if** $\forall v' \in v.strongEdges \cup v.weakEdges : v' \in \bigcup_{k \geq 1} DAG[k]$ **then**
10: $DAG[v.round] := DAG[v.round] \cup \{v\}$
11: **else**
12: DEFER dealing with v until any new vertex committed
13: **end if**
14: **end if**

15: **event** $|DAG[r]| \geq f + 1$ **do**
▷ The quorum for the system is adjusted to $f+1$ out of $n = 2f+1$
16: **if** $r > 1 \wedge (r-1)\ mod\ 3 = 0$ **then**
▷ Each wave contains 3 rounds
17: WAVEREADY ($\frac{(r-1)}{3}$)
18: **end if**
19: *round-cert* := RAC.VALIDATEVERTICE($DAG[r]$)
20: $r := r + 1$
21: v := CREATENEWVERTEX(r, *round-cert*)
22: $R\_Bcast_i(v, r)$

23: **function** CREATENEWVERTEX(r, *round-cert*) **do**
24: **wait until** $\neg blocksToPropose$.EMPTY()
25: $v.block := blocksToPropose$.DEQUEUE()
26: $v.strongEdge := DAG[r-1]$
27: SETWEAKEDGES(v,r)
28: **return** v

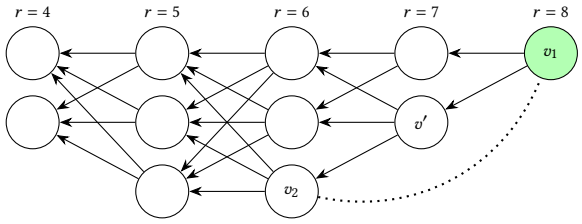
29: **function** SETWEAKEDGES(v,r) **do**
30: $v.weakEdges := \{\}$
31: **for** $r = round - 2$ down to 1 **do**
32: **for** $u \in DAG_i[r]$ s.t. $\neg$PATH(v, u) **do**
33: $v.weakEdges := v.weakEdges \cup \{u\}$
34: **end for**
35: **end for**

Figure 5: Pseudocode for DAG construction

A key improvement of FIDES is the utilization of this lightweight common coin to generate globally consistent random values for leader election. This coin introduces negligible computational and communication overhead, ensuring efficient leader election.

Specifically, the DAG is organized into *waves*, with each comprising three consecutive rounds (we provide the reason for using three rounds per wave in A.4.2). Conceptually, during the first round, replicas propose vertices that encapsulate their entire causal history. The second and third rounds involve replicas voting on these proposals by referencing them in subsequent vertices. The third round uses the *TEE-Assisted Common Coin* mechanism to retrospectively elect a leader vertex from the first round of the wave (line 61). A leader vertex is committed if it is referenced by at least $f+1$ vertices in the third round (line 41). Once a leader vertex is committed, all

The DAG view of replica$_i$:



The DAG view of replica$_j$:

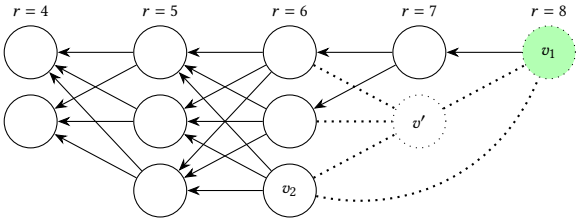


Figure 6: DAGs in Fides

of its preceding causal history is transitively committed in a predefined order (line 67, *e.g.*, Deep-First Search, Breadth-First Search, *etc.*).

However, due to potentially divergent views of the DAG among replicas or situations where a vertex fails to gain sufficient support, not all replicas may commit a leader vertex in each wave. To address this, Fides employs a recursive committing mechanism to maintain consistency (line 45):

1. Once a leader is committed in wave w, it becomes the candidate for transaction ordering.
2. The system recursively checks preceding waves to identify the most recent wave w' with a committed leader.
3. For each wave between w' and w, paths between the current candidate leader and the leader vertices of earlier waves are transitively committed in ascending order ($w'+1, w'+2, ..., w$).

This recursive committing mechanism ensures that leader commitments across replicas are eventually consistent and causally ordered, even in the presence of Byzantine faults or network inconsistencies. Fig. 1 illustrates a recursively committing process in Fides. In consequence, an elected wave leader vertex b of wave v is committed by a non-faulty replica r through either of the following mechanisms:

(1) **Directly Commit:** In the local view of p, b is referenced by at least $f+1$ vertices from the third round of wave w.
(2) **Indirectly Commit:** A path exists from b_s to b, where b_s is the directly committed leader vertex of a subsequent wave $w_s > w$.

Conclusively, by leveraging all these TEE-assisted services, Fides achieves a better fault tolerance (*i.e.*, tolerating up to f Byzantine faults in an $n = 2f+1$ setting) and enables more frequent commitments, which is especially advantageous in large scale system. As demonstrated in Lemma 7 and Lemma 8 in Sec. A, Fides is expected to commit a vertex leader every three rounds in the DAG with random message delays and every six rounds under an asynchronous adversary.

Fides (running at each replica p_i) :

Local variables:
$decidedWave := 0$
$deliveredVertices := \{\}$
$leadersStack :=$ {initialize empty stack with IsEmpty(), Push(), and Pop() functions}

37: **event** On receiving a block of transactions b from clients **do**
38: $blocksToPropose$.Enqueue(b)

39: **function** WaveReady((w)) **do**
40: $v :=$ GetWaveVertexLeader(w)
41: **if** $v = \perp \vee |\{v' \in DAG_i[round(w,3)] : \text{StrongPath}(v', v)\}| < f+1$ **then**
42: **return**
43: **end if**
44: $leadersStack$.Push(v)
45: **for** wave w' from w - 1 down to $decidedWave + 1$ **do**
46: $v' :=$ GetWaveVertexLeader(w')
47: **if** $v' \neq \perp \wedge$ StrongPath(v, v') **then**
48: $leadersStack$.Push(v')
49: $v := v'$
50: **end if**
51: **end for**
52: $decidedWave := w$
53: OrderVertives($leadersStack$)

54: **function** GetWaveVertexLeader(w) **do**
55: $p_j :=$ ChooseLeader$_i(w)$
56: **if** $\exists v \in DAG[round(w,1)]$ $s.t.$ $v.source = p_j$ **then**
57: **return** v
58: **end if**
59: **return** $\perp$

60: **function** ChooseLeader$_i(w)$ **do**
61: **return** $RANG$.Rand(w, $DAG_i[round(w,3)]$)
▷ Use RANG as a common coin to elect the leader

62: **function** OrderVertives($leadersStack$) **do**
63: **while** $\neg leadersStack$.IsEmpty() **do**
64: $v := leadersStack$.Pop()
65: $verticesToDeliver := \{v' \in \bigcup_{r>0} DAG_i[r] \mid$ Path$(v, v') \wedge v' \notin deliveredVertices\}$
66:
67: **for every** $v' \in verticesToDeliver$ in some predefined order **do**
68: $v'.block :=$ $TRAD$.Decrypt($v'.block$)
▷ Reveal transactions before global commit
69: **output** CommitBlock$_i(v'.block, v'.round, v'.source)$
▷ The deliver result of the Byzantine consensus process
70: $deliveredVertices := deliveredVertices \cup \{v'\}$
71: **end for**
72: **end while**

73: **event** On confirming a transactions tx with new $sessionKey$ **do**
74: $\langle sessionId, \phi \rangle :=$ $TRAD$.AddSessionKey($tx.transactionData$)
75: Respond to the client with the $sessionId$

Figure 7: Pseudocode of Fides's Byzantine Consensus

6.3 Censorship Resistance

Ensuring censorship resistance is a critical challenge in blockchain systems, where transaction visibility can be exploited by malicious

```
Client-Side Workflow:
Local Variables:
    transactionData: The transaction content to be submitted
    CensorshipProtection: Boolean flag indicating whether MEV protection is required
    mainPK: The public key of the main key published by Fides
    sessionKey: AES key generated for encrypting transactions
    sessionId: Identifier for the securely exchanged session key

76: function InitializeSessionKey() do
77:     if CensorshipProtection then
78:         sessionKey := GenerateAESKey()
79:         encryptedKey := TRAD.Encrypt(sessionKey, mainPK)
                                    ▷ Encrypt using main key's public key
80:         if sessionKey is new then
81:             transactionData := encryptedKey + transactionData
82:         end if
83:     end if

84: function SubmitTransaction(transactionData) do
85:     if CensorshipProtection then
86:         transactionData := Encrypt(transactionData, sessionKey)
87:     end if
88:     SendTransaction(transactionData, sessionId)
                                    ▷ Send transaction to replicas

89: event OnReceivingRespond(resp) do
90:     if sessionId =⊥ then
91:         sessionId = resp.sessionId
92:     end if
```

Figure 8: Pseudocode for Client-Side Workflow

replicas to selectively include, exclude, or reorder transactions for unfair advantages. To address this, we propose a *TEE-assisted Transaction Disclosure* mechanism supported by TEE and dual-key encryption mechanisms. This mechanism ensures that transaction details remain concealed during the consensus process, only being disclosed at the final commitment stage. By protecting transaction data from premature access or manipulation, it strengthens the integrity of transaction ordering and mitigates censorship-related vulnerabilities at some points.

Instead of always relying solely on the computationally expensive asymmetric cryptography, we leverage a balanced approach by reasonably leveraging the symmetric cryptography. To enable secure transaction handling, Fides employs a two-tier encryption mechanism consisting of:

- **Main Key:** A shared RSA asymmetric key pair is used for managing session keys. The private component is securely stored within the *TRAD* component of the TEE, while the public component is distributed to all clients.
- **Session Keys:** Symmetric AES keys, generated by individual clients, to encrypt transaction content. These session keys are securely shared with replicas by encrypting them using the main key's public key.

The detailed mitigation process is shown in Fig. 8, involving the following steps:

1. **Main Key Setup:** A shared RSA key pair is generated during the system initialization by a DKG protocol. The private key is securely stored within the trusted component *TRAD*, while the corresponding public key is published to all clients for secure submission of their session keys.
2. **Secure Session Key Exchange:** (Optional for clients requiring censorship protection) Each client generates its unique session key (AES) (line 78), encrypts it using the public key of the main key (line 79), and includes it in the transaction payload. Once this transaction is confirmed, the replicas respond with a *sessionId* (line 89), which the client can use this session key for subsequent operations.
3. **Client Submission:** Clients can choose to submit transaction data either encrypted with a session key or in plaintext (line 84), depending on their preference. If encrypted, the confidentiality of sensitive content is assured.
4. **Consensus and Deferred Decryption:** Encrypted transaction data remains protected throughout the consensus process, preventing inspection or manipulation by replicas. Once the consensus is reached, replicas validate the necessary conditions, including the status of the *RANG* component, the causal history of the vertex in the DAG, and confirmation of the next-wave leader. Upon validation, the *TRAD* component will decrypt the transaction content using the associated session key within the TEE (line 68).

By utilizing this *TEE-assisted Transaction Disclosure* mechanism, several architectural improvements have been implemented:

Dedicated Decryption Enclave: The decryption process is isolated within a dedicated enclave, separate from other TEE-assisted services such as reliable broadcast and vertex validation. This separation effectively reduces interference, enhances concurrency, and minimizes latency.

Parallel Execution To maximize resource utilization and optimize system performance, parallel execution is enabled, which allows the decryption enclave to operate simultaneously with other system processes, such as DAG construction and validation. Moreover, A multi-threaded approach is used for decryption, enabling the parallel processing of multiple tasks. This approach supports higher transaction throughput, making the protocol suitable for high-demand environments.

Minimized Client Reliance By securely managing the session keys within the TEE, Fides eliminates dependence on clients for decryption. All sensitive and cost-intensive operations, including transaction decryption and disclosure, occur exclusively within the trusted TEE environment, reinforcing the integrity of the protocol. Furthermore, in a blockchain system, it is reasonable to assume the existence of client proxies that are trusted by clients. These proxies are responsible for establishing a secure connection, which can then be efficiently maintained and reused over an extended period.

These enhancements provide notable advantages. Isolating the decryption process into a dedicated enclave improves concurrency and avoids conflicts with other TEE-assisted services. By encrypting transactions throughout the consensus process and deferring decryption to the final commitment stage, the protocol significantly reduces censorship risks and thus ensures robust confidentiality.

Furthermore, the optimized design enhances performance by reducing latency and supporting higher transaction throughput. Collectively, these refinements mitigate censorship attacks while introducing minimal computation overhead, maintaining both scalability and efficiency.

7 Evaluation

We compare the performance of Fides with several state-of-the-art BFT consensus protocols: RCC [27], Tusk [26], PBFT [8], and HotStuff [28]. We measure throughput and latency under varying conditions (*e.g.*, faulty nodes, network latency, batching). Our experiments aim to answer the following questions:

- **Q1:** How does Fides scale compare to existing approaches across various scenarios, including fault tolerance and batching? (Sec. 7.2)
- **Q2:** What is the resiliency of Fides under varying network latencies and leader failure scenarios? (Sec. 7.3)
- **Q3:** How does the latency breakdown of Fides compare to Tusk? (Sec. 7.4)

7.1 Implementation and Evaluation Setup

Implementation. We implemented Fides atop the open-source Apache ResilientDB (Incubating) platform [6, 24], a scalable and global blockchain infrastructure that supports high-throughput distributed ledger applications. Our implementation, developed in C++, utilizes *std::thread* for managing asynchronous operations. We use Apache ResilientDB for durable data structure storage. For reliable point-to-point communication, we use TCP, which is critical for maintaining the correctness of distributed system abstractions. We use *Intel SGX*[25] to realize the core trusted components (*i.e.*, *MIC*, *RAC*, *RANG*, and *TRAD*) within SGX enclaves at each replica. We utilize the Open Enclave SDK[7] to integrate these components into the enclaves.

Evaluation Setup. We conducted experiments on Alibaba Cloud Elastic Compute Service (ECS) machines to evaluate Fides's performance. Trusted instances, available exclusively in the Hong Kong region, utilized the SGX-enabled (`g7t.xlarge`) instance type, while the untrusted instances used the non-SGX (`g7.xlarge`) instance type. Each process ran on dedicated virtual machines configured with 4 vCPUs and 16GB of RAM, operating on Ubuntu Linux 22.04.

In the LAN environment, all machines were located within the same data center in Hong Kong, which supports secure Intel SGX mode. WAN environments spanned four data centers across the United States, UAE, Singapore, and Germany, with inter-datacenter latencies ranging from 80ms to 270ms.

However, as SGX secure mode is not available in these regions, we conducted WAN experiments in a simulated SGX [5] mode. To further evaluate Fides's performance under real SGX-enabled environments under WAN-like conditions, we introduced a *Delay-Injected Network (DIN)* environment within the Hong Kong region. This configuration manually injected specific end-to-end delays between machine pairs to emulate WAN conditions, with a default delay of 100ms representing typical inter-datacenter latencies.

We conduct experiments in three network environments: LAN, WAN, and DIN. Specifically, in all experiments, each node exclusively ran a single Fides process, with the number of clients matching the number of consensus processes. Clients continuously sent requests until system saturation was detected internally. The benchmark used was a key-value service, where each request involved a randomly generated key-value insertion. Transactions were transmitted in plain text, bypassing decryption operations. To guarantee statistical robustness, we performed three experimental runs for each condition and plotted the median value.

7.2 Scalability Evaluation

We evaluate the scalability of Fides by comparing its performance with other protocols—Tusk, RCC, HotStuff, and PBFT—under different fault tolerance levels and batch sizes.

7.2.1 Varying Fault Tolerance. We evaluate the performance of different protocols with varying fault tolerance under WAN, DIN, and LAN environment.

Performance under WAN. In the WAN environment, where network latencies are substantially higher, scalability is a critical challenge for consensus protocols. Fig. 9(a) and Fig. 9(b) depicts the throughput and latency of Fides (SGX-SIM), Tusk, RCC, HotStuff and PBFT as the system's fault tolerance f increases from 2 to 20.

The figures show Fides (SGX-SIM) demonstrates superior scalability, consistently achieving higher throughput compared to other protocols across all fault tolerance levels. Notably, Fides (SGX-SIM) outperforms Tusk by more than fivefold and RCC by nearly double for all f values. At $f = 2$, Fides (SGX-SIM) achieves approximately 0.57s latency, which is five times lower than Tusk and half that of RCC. While latency increases moderately with f, Fides (SGX-SIM) remains the most efficient protocol.

Conclusively, the significant performance gains of Fides in WAN environments are attributed to its optimized Reliable Broadcast mechanism, enabled by the Trusted Component, *MIC*, which reduces message exchanges and mitigates high network latencies.

Performance under DIN To evaluate potential overhead introduced by the TEE, particularly in hardware mode (*SGX-HW*), we conducted tests under a *Delay-Injected Network* environment. This setup introduces an artificial 100ms end-to-end latency between replicas in the same region. Fig. 9(c) and Fig. 9(d) illustrate the throughput and latency results in this scenario.

In DIN environments, both Fides-HW and Fides (SGX-SIM) outperform all other protocols significantly. At $f = 2$, both Fides modes achieve approximately 72k txn/s, and as f increases to 20, Fides scales effectively, reaching around 560k txn/s for both modes. In contrast, RCC and Tusk achieve significantly lower throughput, with 228k txn/s and 160k txn/s, respectively.

Additionally, Fides consistently maintains low latency across all fault tolerance levels, demonstrating the effectiveness of Fides in scalability.

Performance under LAN. We also evaluate Fides in a LAN environment, as shown in Fig. 9 (e) and Fig. 9 (f). While all protocols benefit from negligible network latencies in a LAN setup, Fides (in both SGX hardware mode and simulate mode) continues to outperform in most conditions.

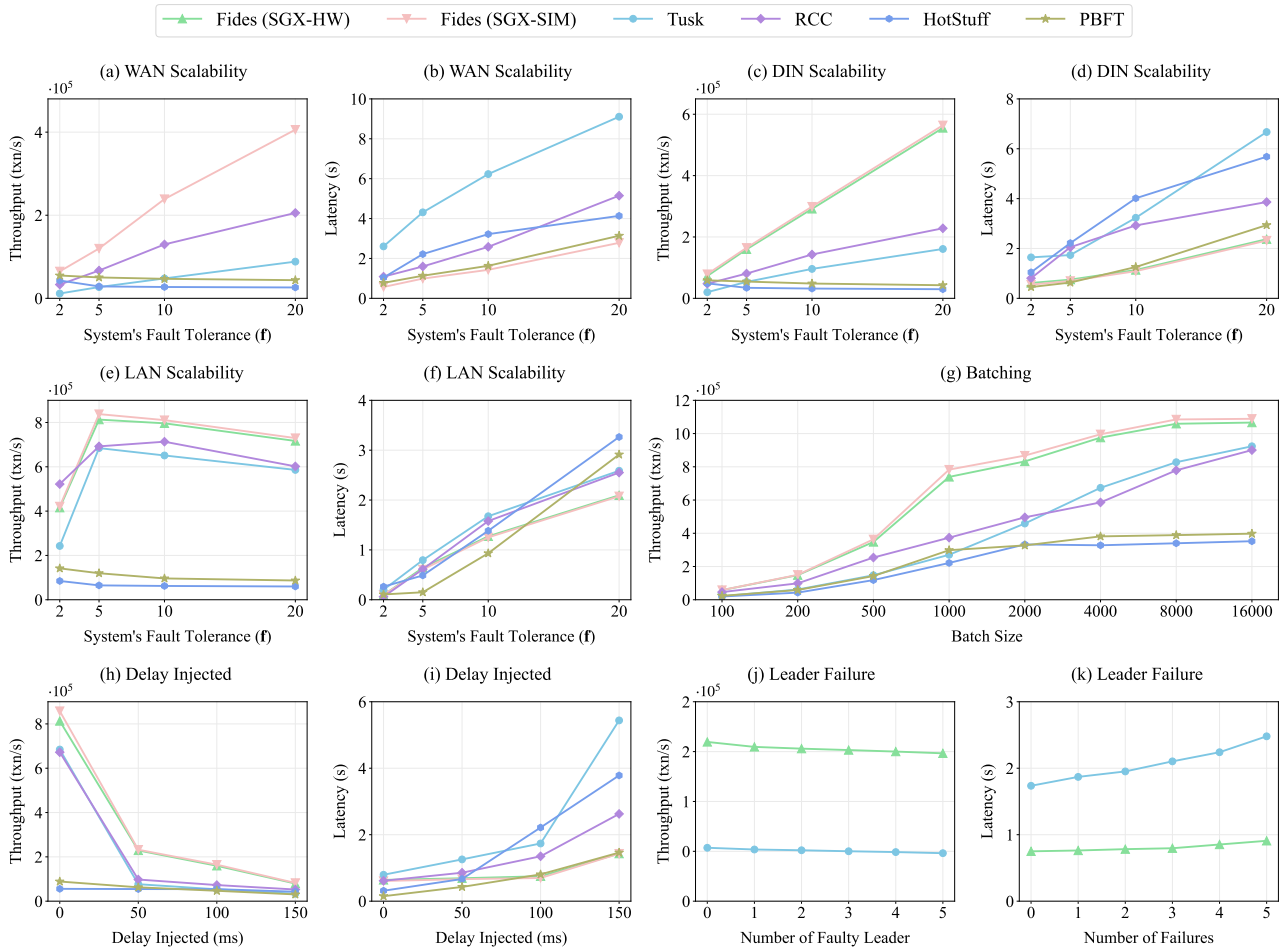


Figure 9: Impact of varying degrees of fault tolerance, batch sizes, and failures

At $f = 2$, Fides achieves approximately 415k txn/s (SGX mode), surpassing Tusk and PBFT. RCC shows higher throughput at lower f values due to its advantage in low round-trip time environments. However, as f increases, Fides scales more effectively, eventually overtaking all other protocols, including RCC.

Latency remains low across all protocols in the LAN environment. Fides maintains latency comparable to RCC and PBFT while consistently outperforming Tusk and HotStuff in terms of latency efficiency.

Although the benefits of Fides's RBC optimization are less pronounced in LAN environments, it still demonstrates strong scalability, especially at higher fault tolerance levels.

7.2.2 Varying Batch Size. Fig. 9(g) illustrates the impact of batch size on performance in systems with a fault tolerance of 5. All protocols show increased throughput as the batch size grows, though the gains plateau beyond a certain point. Notably, Fides consistently achieves the highest throughput, highlighting its efficiency in handling large batches of transactions and maintaining superior performance across varying batch sizes.

7.3 Resiliency Evaluation

Next, we assess the resiliency of Fides by examining the impact of varying network latencies and the number of leader failures on streamlined protocols.

7.3.1 Injecting Network Delay. We evaluate the performance of all the protocols under network delays injected in the range of 0ms to 150ms. As shown in Fig. 9(h) and Fig. 9(i), Fides (in both SGX-HW and SGX-SIM modes) maintains higher throughput and lower latency compared to other protocols across all delay settings. This demonstrates Fides's resilience to network-induced latencies, ensuring stable and efficient performance even in challenging network conditions.

7.3.2 Varying Number of Leader Failure Nodes. We evaluate the performance of Fides (SGX-HW) and Tusk under a fault tolerance of 5 in a DIN environment. A failure node in these protocols is one that permanently fails to receive enough votes, preventing it from being committed as a leader. As shown in Fig. 9(j), Fides (SGX-HW) consistently outperforms Tusk in throughput, with both gradual declines as the number of failures increases from 0 to 5. For latency shown in Fig. 9(k), Fides (SGX-HW) maintains lower and more moderate growth compared to Tusk, showcasing Fides's robustness and efficiency in handling node failures.

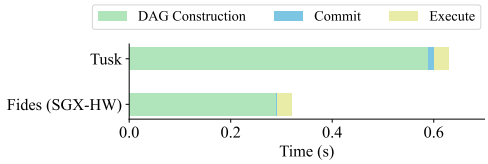


Figure 10: Protocol Time Breakdown by Stage

7.4 System Breakdown

To analyze protocol latency, we provide a breakdown of Fides (SGX mode) and Tusk in a DIN environment with a fault tolerance of 10. For fairness, transaction queuing latency is excluded. The breakdown is divided into three stages, and Fig. 10 shows the average time spent in each stage:

DAG Construction. On average, Fides constructs a new vertex in the DAG in 0.082s, compared to 0.231s for Tusk. While Fides requires 3.54 rounds to commit per vertex, whereas Tusk requires 2.56 rounds, Fides achieves a total DAG construction time of 0.29s-a 50% improvement over Tusk's 0.59s. This efficiency stems from Fides's optimized Reliable Broadcast mechanism, which minimizes communication overhead and mitigates high network latency for faster DAG construction.

Commit. Fides achieves a commit time of 0.0016s, outperforming Tusk's 0.011s by 85%. Although both times are relatively small, this reduction demonstrates Fides's efficiency in handling transaction finalization.

Execute. Both protocols have identical execution times of 0.029s, as execution-related optimizations are not a distinguishing factor.

The breakdown result indicates that Fides achieves a total latency of 0.32s, compared to 0.63s for Tusk, representing a 49% improvement. The key advantage lies in Fides's faster DAG construction, making it particularly efficient in WAN environments where low latency is critical.

8 Related Work

DAG-Based BFT protocols. Recently, DAG-based BFT protocols [19–21, 26, 30–32, 47, 50–56] link transactions in a non-linear, directed manner without forming cycles, enabling higher scalability and faster transaction throughput compared to linear-chain approaches. HashGraph [50] is among the first protocols to incorporate an asynchronous consensus mechanism into a DAG, which separates the network communication layer from the ordering logic. Aleph [51] further introduced a round-based DAG structure, enhancing security and efficiency. DAG-Rider [19] advanced asynchronous BFT consensus by combining reliable broadcast and a round-by-round DAG structure, which optimizes resilience, round complexity, and amortized communication complexity. Narwhal and Tusk [26] introduced a modular approach, decoupling data availability from transaction ordering, while Bullshark [20] adapts Narwhal/Tusk to also optimize partially synchronous environments. In contrast, partially synchronous DAG-based protocols, like [21, 30, 31, 47, 52], achieve low latency in well-behaved networks, but do not fully address challenges in fully asynchronous conditions. Fides builds on these asynchronous DAG protocols and leverages trusted execution environments (TEEs) to further improve scalability in adversarial or unpredictable network settings one step further.

Hardware-assisted BFT protocols. Hardware-assisted BFT protocols [33, 35, 37, 57–60] leverage trusted hardware components—such as TEEs—to improve the performance and security of BFT consensus. TrInc [57] introduces a non-decreasing counter to prevent equivocation in distributed systems, reducing the fault tolerance requirement from $n = 3f + 1$ to $n = 2f + 1$, and simplifying the consensus process from three phases to two. Hybster [33] combines traditional Byzantine fault tolerance with a crash-only trusted subsystem, achieving over 1 million operations per second on an Intel SGX-based prototype. TBFT [37] further reduces message complexity while preventing equivocation, improving both efficiency and security. DAMYSUS[35] employs the *Checker* and *Accumulator* trusted components to simplify consensus while enhancing fault tolerance with fewer communication rounds. In cases where the trustworthiness of replicas and their TEEs is uncertain, FlexiTrust [58] further refines this by dynamically adjusting the fault tolerance threshold between $2f+1$ and $3f+1$, based on the reliability of trusted components. OneShot [59] presents a view-adapting BFT protocol that utilizes TEEs for efficient fault tolerance, making it particularly suitable for systems requiring high reliability. Recent advancements include leaderless or view-adapting BFT protocols that utilize TEEs for enhanced fault tolerance. Trusted Hardware-Assisted Leaderless BFT [60] eliminates the need for a leader, reducing vulnerabilities in the consensus process. In contrast, Fides goes a step further by leveraging TEE's guarantees to offload expensive tasks—such as global randomness generation and equivocation prevention—into the enclave, thus achieving significant performance improvements.

9 Conclusion

In this paper, we present Fides, an asynchronous DAG-based BFT consensus protocol leveraging TEEs to address scalability, efficiency, and security challenges. By leveraging four TEE-assisted components—Reliable Broadcast, Vertex Validation, Common Coin, and Transaction Disclosure—Fides reduces communication complexity, minimizes reliance on computationally expensive cryptographic primitives, and enhances resilience against censorship attacks. The scalable and robust consensus mechanism designed in Fides, validated by comprehensive experimental evaluations, demonstrates its suitability for real-world deployment in decentralized systems.

References

[1] Cybernews: Front Runners Draining $280 Million per Month from Crypto Transactions. https://cybernews.com/crypto/flash-boys-2-0-front-runners-draining-280-million-per-month-from-crypto-transactions. (Accessed: 2024-11-28).
[2] Intel. SGX Remote Attestation. https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html. (Accessed: 2024-11-28).
[3] AMD. SEV Remote Attestation. https://enarx.dev/docs/technical/amd-sev-attestation. (Accessed: 2024-11-28).
[4] Intel. Software Security Guidance. https://www.intel.com/content/www/us/en/developer/topic-technology/software-security-guidance/overview.html. (Accessed: 2024-11-28).
[5] How to Run Intel® Software Guard Extensions' Simulation Mode. https://www.intel.com/content/www/us/en/developer/articles/training/usage-of-simulation-mode-in-sgx-enhanced-application.html. (Accessed: 2024-11-28).
[6] Apache ResilientDB (Incubating). https://resilientdb.incubator.apache.org/. (Accessed: 2024-11-28).
[7] Open Enclave SDK. https://openenclave.io. (Accessed: 2024-11-28).

[8] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *OSDI*, 1999.

[9] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the Presence of Partial Synchrony. In *JACM*, 1988.

[10] Tushar Deepak Chandra and Sam Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. In *JACM*, 1996.

[11] Chunbo Chu and Monica Brockmeyer. Predicate Detection Modality and Semantics in Three Partially Synchronous Models. In *ICIS*, 2008.

[12] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. In *NSDI*, 2009.

[13] Zeta Avarikioti, Lioba Heimbach, Roland Schmid, Laurent Vanbever, Roger Wattenhofer, and Patrick Wintermeyer. FnF-BFT: Exploring Performance Limits of BFT Protocols. In *arXiv preprint arXiv:2009.02235*, 2020.

[14] Andrew Lewis-Pye and Ittai Abraham. Fever: Optimal Responsive View Synchronisation. In *arXiv preprint arXiv:2301.09881*, 2023.

[15] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The Honey Badger of BFT Protocols. In *CCS*, 2016.

[16] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *PODC*, 2019.

[17] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-NG: Fast Asynchronous BFT Consensus with Throughput-Oblivious Latency. In *CCS*, 2022.

[18] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-MVBA: Optimal Multi-valued Validated Asynchronous Byzantine Agreement, Revisited. In *PODC*, 2020.

[19] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All You Need is DAG. In *PODC*, 2021.

[20] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT Protocols Made Practical. In *CCS*, 2022.

[21] Alexander Spiegelman, Balaji Aurn, Rati Gelashvili, and Zekun Li. Shoal: Improving DAG-BFT Latency and Robustness. In *FC*, 2024.

[22] Victor Shoup. Blue fish, red fish, live fish, dead fish. In *Cryptology ePrint Archive, Paper 2024/1235*, 2024.

[23] Zhuolun Xiang, Sourav Das, Zekun Li, Zhoujun Ma, and Alexander Spiegelman. The latency price of threshold cryptosystem in blockchains. In *arXiv preprint arXiv:2407.12172*, 2024.

[24] Sajjad Rahnama, Suyash Gupta, Thamir M. Qadah, Jelle Hellings, and Mohammad Sadoghi. Scalable, Resilient, and Configurable Permissioned Blockchain Fabric. In *PVLDB*, 2020.

[25] Victor Costan and Srinivas Devadas. Intel SGX Explained. In *Cryptology ePrint Archive, Paper 2016/086*, 2016.

[26] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus. In *EuroSys*, 2022.

[27] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing. In *ICDE*, 2021.

[28] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT Consensus with Linearity and Responsiveness. In *PODC*, 2019.

[29] Jianting Zhang and Aniket Kate. No Fish Is Too Big for Flash Boys! Frontrunning on DAG-based Blockchains. In *Cryptology ePrint Archive*, 2024.

[30] Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin. BBCA-CHAIN: One-Message, Low Latency BFT Consensus on a DAG. In *FC*, 2024.

[31] Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the Limits of Latency with Uncertified DAGs. In *arXiv preprint arXiv:2310.14821*, 2024.

[32] Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. Sailfish: Towards improving the latency of DAG-based BFT. In *S&P*, 2025.

[33] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. Hybrids on Steroids: SGX-Based High Performance BFT. In *EuroSys*, 2017.

[34] Giuliana Veronese, Miguel Correia, Alysson Bessani, Lau Lung, and Paulo Veríssimo. Efficient Byzantine Fault-Tolerance. In *TC*, 2013.

[35] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. DAMYSUS: Streamlined BFT Consensus Leveraging Trusted Components. In *EuroSys*, 2022.

[36] Balaji Arun and Binoy Ravindran. Scalable Byzantine Fault Tolerance via Partial Decentralization. In *PVLDB*, 2022.

[37] Jiashuo Zhang, Jianbo Gao, Ke Wang, Zhenhao Wu, Yue Li, Zhi Guan, and Zhong Chen. TBFT: Efficient Byzantine Fault Tolerance Using Trusted Execution Environment. In *ICC*, 2022.

[38] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. ROTE: Rollback Protection for Trusted Execution. In *Security*, 2017.

[39] Sebastian Angel, Aditya Basu, Weidong Cui, Trent Jaeger, Stella Lau, Srinath Setty, and Sudheesh Singanamalla. Nimble: Rollback Protection for Confidential Cloud Services. In *OSDI*, 2023.

[40] Marcus Brandenburger, Christian Cachin, Matthias Lorenz, and Rüdiger Kapitza. Rollback and Forking Detection for Trusted Execution Environments using Lightweight Collective Memory. In *DSN*, 2017.

[41] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In *S&P*, 2019.

[42] Stephan Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. RIDL: Rogue In-Flight Data Load. In *S&P*, 2019.

[43] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. ZombieLoad: Cross-Privilege-Boundary Data Sampling. In *CCS*, 2019.

[44] Yuval Yarom, Daniel Genkin, and Nadia Heninger. CacheBleed: A Timing Attack on OpenSSL Constant Time RSA. In *CHES*, 2016.

[45] Jonathan Katz and Yehuda Lindell. Introduction to Modern Cryptography (2nd edition). In *Chapman and Hall/CRC*, 2014.

[46] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. On the diversity of asynchronous communication. In *FAC*, 2016.

[47] Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High Throughput DAG BFT Can Be Fast! In *arXiv preprint arXiv:2405.20488*, 2024.

[48] Gabriel Bracha. Asynchronous Byzantine Agreement Protocols. In *Inf. Comput.*, 1987.

[49] Adi Shamir. How to Share a Secret. In *CACM*, 1979.

[50] Leemon Baird and Atul Luykx. The Hashgraph Protocol: Efficient Asynchronous BFT for High-Throughput Distributed Ledgers. In *COINS*, 2016.

[51] Adam Gagol, Damian Leśniak, Damian Straszak, and Michał undefinedwiundefinedtek. Aleph: Efficient Atomic Broadcast in Asynchronous Networks with Byzantine Nodes. In *AFT*, 2019.

[52] Philipp Jovanovic, Lefteris Kokoris Kogias, Bryan Kumara, Alberto Sonnino, Pasindu Tennage, and Igor Zablotchi. Mahi-mahi: Low-latency asynchronous BFT DAG-based consensus. In *arXiv preprint arXiv:2410.08670*, 2024.

[53] Dahlia Malkhi and Pawel Szalachowski. Maximal Extractable Value (MEV) Protection on a DAG. In *SBC*, 2022.

[54] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. Autobahn: Seamless high speed bft. In *SOSP*, 2024.

[55] Xiaohai Dai, Zhaonan Zhang, Jiang Xiao, Jingtao Yue, Xia Xie, and Hai Jin. GradedDAG: An Asynchronous DAG-based BFT Consensus with Lower Latency. In *SRDS*, 2024.

[56] Xiaohai Dai, Guanxiong Wang, Jiang Xiao, Zhengxuan Guo, Rui Hao, Xia Xie, and Hai Jin. LightDAG: A Low-latency DAG-based BFT Consensus through Lightweight Broadcast. In *IPDPS*, 2024.

[57] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. TrInc: Small Trusted Hardware for Large Distributed Systems. In *NSDI*, 2009.

[58] Suyash Gupta, Sajjad Rahnama, Shubham Pandey, Natacha Crooks, and Mohammad Sadoghi. Dissecting BFT Consensus: In Trusted Components we Trust! In *EuroSys*, 2022.

[59] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. OneShot: View-Adapting Streamlined BFT Protocols with Trusted Execution Environments. In *IPDPS*, 2024.

[60] Liangrong Zhao, Jérémie Decouchant, Joseph Liu, Qinghua Lu, and Jiangshan Yu. Trusted Hardware-Assisted Leaderless Byzantine Fault Tolerance Consensus. In *TDSC*, 2024.

A Correctness Analysis

A.1 TEE-Assisted Reliable Broadcast

Theorem A.1. *The TEE-Assisted Reliable Broadcast in Fides satisfies* ***Validity****.*

Proof. Suppose a correct initiator p_j create a vertex v with $v.source = p_j$ and invokes $R\_Bcast_j(v)$ at round $v.round$. By using the TEE's monotonic counter, p_j obtains a unique certificate ϕ attached into v ensuring that it can produce only one valid message for round $v.round$. Even if the network is asynchronous or some correct replicas receive the message late, at least $f + 1$ correct replicas will eventually receive v. Each of these replicas, upon validating ϕ, performs an echo broadcast to the entire network. Since there are $n = 2f + 1$ replicas and at least $f + 1$ of them are correct, every correct replica will eventually receive an echo of v. After successfully verifying ϕ, each correct replica will invoke $R\_Deliver_i(v)$. Hence, if a correct initiator $R\_Bcast(v)$, all correct replicas eventually $R\_Deliver(v)$, satisfying *Validity*. □

Theorem A.2. *The TEE-Assisted Reliable Broadcast in Fides satisfies* ***Integrity****.*

Proof. Consider any vertex v. By construction, the TEE ensures that only the authorized initiator $v.source$ can produce the unique valid certificate ϕ for round $v.round$. If an adversary attempts to forge the vertex v or create multiple conflicting messages for the same round, it will fail the TEE validation, since the counter value is synchronized with the DAG's round, and correct replicas will discard such messages. Consequently, replicas only deliver a vertex v once they confirm it carries a valid ϕ associated with $v.source$ for round $v.round$. As no other message with the same ϕ can exist, and no correct replica will accept a message lacking a valid TEE certificate from $v.source$, a message is delivered at most once and only if it was truly broadcast by $v.source$. This ensures *Integrity*. □

Theorem A.3. *The TEE-Assisted Reliable Broadcast in Fides satisfies* ***Agreement****.*

Proof. Assume some correct replica p_i has delivered $R\_Deliver_i(v)$. For p_i to do so, it must have received v containing its unique certificate ϕ and observed that the vertex is correct. Since p_i is correct, it must have received at least one valid $\langle echo, v\rangle$ from a correct replica who validated v. Because there are $f + 1$ correct replicas who have seen v and echoed it, any other correct replica p_k will eventually receive at least one valid $\langle echo, v\rangle$. On receiving this echo and verifying the ϕ inside v, p_k will also $R\_Deliver_k(v)$. Thus, if any correct replica delivers v, all correct replicas eventually do the same, ensuring *Agreement*. □

A.2 TEE-Assisted Common Coin

Theorem A.4. *The TEE-Assisted Common Coin in Fides satisfies* ***Agreement****.*

Proof. When at least $f + 1$ replicas invoke ChooseLeader(w) for the same wave w, they rely on the same shared seed and input parameters within their *RANG* components. Since the *RANG* operates deterministically based on these inputs and the shared cryptographic seed, it produces the same output at all correct replicas. Thus, the selected leader is identical across all correct replicas, satisfying the **Agreement** property. □

Theorem A.5. *The TEE-Assisted Common Coin in Fides satisfies* ***Termination****.*

Proof. If at least $f + 1$ replicas invoke ChooseLeader(w), sufficient valid *MIC* values are generated to construct the required quorum proof. The reliable broadcast mechanism ensures that these *MIC* values, embedded in the corresponding vertices, are eventually delivered to all correct replicas. Consequently, any correct replica invoking ChooseLeader(w) can collect the necessary quorum proof and obtain a response from the *RANG* component, thus ensuring **Termination**. □

Theorem A.6. *The TEE-Assisted Common Coin in Fides satisfies* ***Unpredictability****.*

Proof. As long as fewer than f+1 replicas have invoked ChooseLeader(w), the quorum proof required by the *RANG* component cannot be formed. Without this proof, the output of the *RANG* remains inaccessible and unpredictable to any adversary. Additionally, the internal state of the *RANG* is securely protected within the TEE, preventing external entities from predicting its output. Therefore, the outcome remains computationally indistinguishable from random, satisfying the **Unpredictability** property. □

Theorem A.7. *The TEE-Assisted Common Coin in Fides satisfies* ***Fairness****.*

Proof. The *RANG* component uses a cryptographically secure PRNG initialized with a shared seed, and when selecting a leader, it computes a random number modulo n, where n is the number of replicas. This process ensures that each replica has an equal probability $\frac{1}{n}$ of being selected as the leader. Since the PRNG outputs are uniformly distributed and cannot be biased by any replica, the **Fairness** property is satisfied. □

A.3 DAG

Claim 1. *If a correct replica p_i adds a vertex v to its DAG, then eventually, all correct replicas add v to their DAG.*

Proof. We prove the claim by induction on the rounds.

Base Case: In the initial round, all correct replicas share the same initial state, so the claim holds trivially.

Inductive Step: Assume that for all rounds up to $r - 1$, any vertex added by a correct replica is eventually added to the DAGs of all other correct replicas.

Now consider a vertex v added by a correct replica p_i in round r. When v is reliably delivered to p_i (line 7), it conducts validation, including verifying v's validity (line 8) and ensuring all referenced vertices by v are already in the *DAG* (line 9).

By the *Agreement* property of reliable broadcast (Sec. 3.3), v will eventually be delivered to all correct replicas. Since v passed these checks at p_i, and all correct replicas have consistent DAGs for earlier rounds by the induction hypothesis, v will pass validation at all correct replicas. Thus, all correct replicas will eventually add v to their DAGs. □

Lemma 1. *All correct replicas will eventually have an identical DAG view.*

Proof. From Claim 1, any vertex added by a correct replica will eventually be added to the DAGs of all other correct replicas. Despite network asynchrony and variations in message delivery times, Claim 1 guarantees that all correct replicas receive the same set of messages. Since all correct replicas apply the same deterministic rules to incorporate these vertices into their DAGs, the DAGs of all correct replicas will converge to the same state over time. Thus, all correct replicas will eventually have identical DAG views. □

A.4 Asynchronous Consensus

A.4.1 Safety Analysis

Lemma 2. *When electing the leader vertex of wave w, for any two correct replicas p_i and p_j, if p_i elects v_i and p_j elects v_j, then $v_i = v_j$.*

Proof. To prove that $v_i = v_j$, we need to ensure that all correct replicas use the same inputs when calling the random function *RANG* (e.g., the vertices from corresponding round in line 61) to elect the leader of the wave w. From Lemma 1, we know that all correct replicas will eventually have identical DAGs. When a correct replica attempts to elect the leader for wave w, it does so after it has advanced to the necessary round and has incorporated all required vertices from previous rounds. Since the DAGs are identical and the random seed in *RANG* is the same across replicas, the inputs to the *Rand* function will be identical. Therefore, the output p_j selected as leader will be the same at all correct replicas. Consequently, the vertex v proposed by p_j in round w will be the same at all correct replicas, so $v_i = v_j$.

□

Lemma 3. *If the leader vertex v of wave w is committed by a correct replica p_i, then for any leader vertex v' committed by any correct replica p_j in a future wave $w' > w$, there exists a path from v' to v.*

Proof. A leader vertex v in wave w is committed only if at least $f + 1$ third-round vertices in wave w reference v. Since each vertex in the DAG must reference at least $f + 1$ vertices from the previous round, quorum intersection ensures that all vertices in the first round of wave $w + 1$ have paths to v. By induction, this property extends to all vertices in all rounds of waves $w' > w$. Thus, every leader vertex v' in waves $w' > w$ has a path to v, proving the lemma.

□

Lemma 4. *A correct replica p_i commits leader vertices in an ascending order of waves.*

Proof. In the algorithm, when a leader vertex v of wave w is committed (either directly or indirectly), the replica pushes v onto a stack (line 44 or line 48). The algorithm then recursively checks for earlier uncommitted leader vertices in waves w' from $w - 1$ down to the last decided wave (line 45). If such a leader v' exists and there is a path from v to v', v' is also pushed onto the stack. Since the stack is a LIFO (Last-In-First-Out) structure when popping vertices to commit (Line 64), the leaders are committed in order from the earliest wave to the current wave w. This ensures that leader vertices are committed in ascending order of waves.

□

Lemma 5. *All correct replicas commit leader vertices in the same ascending order of waves.*

Proof. To show a contradiction, we assume that a correct replica p_i commits leader vertex v of wave w and another correct replica p_j commits leader vertex v' of wave w', where $w < w'$, but v is not committed by p_j.

By Lemma 3, since p_i committed v in wave w, and p_j committed v' in wave $w' > w$, there must exist a path from v' to v in the DAG. This path implies that v should have been indirectly committed by p_j, which contradicts the assumption that v is not committed by p_j. Thus, if a leader vertex is committed by one correct replica, then no other replica can skip committing it.

Combining this observation with Lemma 4, all correct replicas commit leader vertices in the same ascending order of waves. □

Theorem 1. *Fides guarantees **Safety** property.*

Proof. The commit procedure, as defined in the function OrderVertices (lines 62–72), is responsible for committing vertices.

By Lemma 5, all correct replicas commit leader vertices in the same ascending order of waves. Furthermore, each committed leader's causal history is delivered in a predefined, deterministic order (line 67). This ensures that the sequence of vertices—and, by extension, the sequence of transactions contained within these vertices—is identical for all correct replicas.

Additionally, Lemma 1 establishes that all correct replicas maintain identical DAGs. Since all correct replicas apply the same deterministic commit rules, the assignment of sequence numbers *sn* to transactions is consistent across all replicas. Consequently, if two correct replicas p_i and p_j commit transactions tx_1 and tx_2 with sequence number *sn*, it follows that $tx_1 = tx_2$. Thus, Fides guarantees the **Safety** property. □

A.4.2 Liveness Analysis

Lemma 6. *For every wave w there are at least $f + 1$ vertices in the first round of w that satisfy the commit rule.*

Proof. Since each third-round vertex connects to at least $f + 1$ vertices in the first round, we consider two cases:

Case 1: All third-round vertices reach exactly $f+1$ first-round vertices

In this case, the second-round vertices must point to the same set of $f + 1$ first-round vertices due to intersection.

To prove by contradiction, assume that two third-round vertices p_{k1} and p_{k2} reference disjoint sets S_1 and S_2, each containing $f + 1$ first-round vertices. This would require at least $f + 1$ second-round vertices to connect exclusively to S_1 and another $f + 1$ to S_2. Such a configuration is impossible in a system with only $2f + 1$ vertices.

Thus in this case, $f + 1$ first-round vertices satisfy the commit rule.

Case 2: At least one third-round vertex reaches more than $f + 1$ first-round vertices

Let V be the set of third-round vertices, where $f+1 \leq |V| \leq 2f+1$. Define $R(v_k)$ as the set of reachable first-round vertices from each

v_k, and $I(V) = \bigcap_{k=1}^{|V|} R(v_k)$ as the intersection of these sets. By set theory, we have:

$$|U(V)| = |I(V)| + \sum_{k=1}^{|V|} (R(v_k) - |I(V)|)$$

In the DAG, there are at most $2f + 1$ and at least $f + 1$ vertices in each round, and we assume $f > 0$. Moreover, given that at least one third-round vertex reaches more than $f + 1$ first-round vertices, indicating $\sum_{k=1}^{|V|} R(v_k) > |V| \times (f + 1)$.

After substituting the known values and simplifying, we find $|I(V)| > f$, indicating that at least $f + 1$ first-round vertices are reachable.

Conclusion: In both cases, there are at least $f + 1$ first-round vertices that satisfy the commit rule. Consequently, the proof is complete. □

Lemma 7. *In expectation, Fides commits a vertex leader every six rounds in the DAG under an asynchronous adversary*

Proof. Consider any wave w. By Lemma 6, at least $f + 1$ first-round vertices in wave w satisfy the commit rule. The common coin (implemented by *RANG*) ensures that the adversary cannot predict the coin's outcome before the leader is revealed. Consequently, the probability of electing a leader that satisfies the commit rule in wave w is at least $\frac{f+1}{2f+1} \geq \frac{1}{2}$. Therefore, in expectation, a vertex leader is committed every two waves. Since every wave consists of three rounds, we get that, in expectation, Fides commits a vertex leader every six rounds in the DAG. □

Lemma 8. *In networks with random message delays, in expectation, Fides commits each vertex in the DAG in 3 rounds.*

Proof. Consider a wave w. Let v be the leader of wave w, and let U be sets of $f + 1$ vertices in the third round of wave w. Given that message, delays are uniformly distributed at random, and each vertex independently references vertices from its previous round. For the worst-case scenario, assume each vertex references exactly $f + 1$ vertices from the prior round.

Under these conditions, any second-round vertex has a strong path to v with a probability of $p = \frac{f+1}{2f+1}$. The number of vertices X in the second round, out of $n = 2f + 1$ total vertices, that point to the leader v follows a binomial distribution: $X \sim \text{Binomial}(n = 2f + 1, p = \frac{f+1}{2f+1})$, with the probability mass function given by:

$$P(X = k) = \binom{2f+1}{k} \cdot (\frac{f+1}{2f+1})^k \cdot (\frac{f}{2f+1})^{2f+1-k}.$$

Each third-round vertex randomly selects $f + 1$ second-round vertices. The probability that at least one of the $f + 1$ selected vertices points to the leader v is:

$$P(\textit{at least one selected} \mid X = k) = 1 - \frac{\binom{2f+1-k}{f+1}}{\binom{2f+1}{f+1}}$$

Thus, the probability that a third-round vertex has a path to the leader v is:

$$p_u = \sum_{k=0}^{2f+1} P(X = k) \cdot P(\textit{at least one selected} \mid X = k).$$

By introducing an intermediate round in each wave, vertex connectivity is significantly improved, increasing the likelihood of satisfying the commit rule. For $f > 0$, the p_u reaches a minimum of $p_u = 0.889$ when $f = 1$ and continues to increase with larger f, eventually converging to 1.

Next, we compute the probability that at least $f + 1$ out of the $2f + 1$ vertices in U include v. This follows a binomial distribution: $X \sim \text{Binomial}(n = 2f + 1, p = p_u)$, and and is given by:

$$P = \sum_{i=f+1}^{2f+1} \left(\binom{2f+1}{i} \cdot p^i \cdot (1-p)^{(2f+1-i)} \right)$$

This probability function is monotonically increasing for all f at $f > 0$, which means the lowest probability is 0.965 when $f = 1$. For larger f, the probability continues to increase and eventually converges to 1.

Conclusively, for every wave, the probability of electing a leader that satisfies the commit rule is at least 0.965. Consequently, Fides commits a vertex leader in expectation every three rounds in the DAG. □

Claim 2. *Within a system with $n = 2f + 1$ replicas, liveness cannot be guaranteed when using a two-round commit strategy.*

Proof. Assume the asynchronous adversary exists. In each round, consider any set S of $f + 1$ vertices in the second round of wave w. The total number of links from S to the first-round vertices is $(f + 1) \cdot (f + 1) = f^2 + 2f + 1$. The number of first-round vertices ranges from $f + 1$ to $2f + 1$.

Now, suppose each first-round vertex has f links from S. Calculating the difference gives: $f^2 + 2f + 1 - (2f + 1) \cdot f = 1 - f^2 \leq 0$. This indicates that, in the worst-case scenario, progress can stall in each wave, as there is no guarantee that all vertices in the second round can achieve sufficient connectivity to advance.

Consequently, this adversary can effectively halt the protocol's progress within a bounded number of rounds. □

Lemma 9. *Every vertex v in the DAG will eventually be committed by every correct replica.*

Proof. When a correct replica is created, a new vertex establishes strong edges pointing to vertices from the previous round (line 26) and weak edges pointing to vertices with no alternative path in the DAG (line 33). This construction ensures that all vertices are reachable and eventually included in the total order.

By Theorem 1, all correct replicas commit the leader vertices, along with their corresponding causal history, in the same order. Furthermore, Lemma 7 and Lemma 8 demonstrate that Fides guarantees vertices are committed within a constant number of rounds, ensuring a probability of 1 for their eventual commitment.

Thus, every correct replica will eventually commit v within a constant number of rounds. □

Theorem 2. *Fides guarantees **Liveness** property.*

Proof. In Fides, assume a correct replica p_i receives transaction tx. Upon receiving tx, p_i creates a vertex v encapsulating tx along with potentially other transactions. The vertex v is then reliably broadcast to all other replicas.

By Lemma 9, every vertex v in the DAG is eventually committed by every correct replica. Since transaction tx is contained in v, it follows that tx will also be committed by all correct replicas.

Thus, Fides ensures that any transaction received by a correct replica will eventually be committed by all correct replicas, satisfying the **Liveness** property. □

A.4.3 Censorship-Resistance Analysis

Lemma 10. *Fides can defend against* ***censorship attack****.*

Proof. In Fides, all replicas share a common key pair managed within the trusted component *TRAD*, where the private key is securely stored within the TEE, and the public key is made publicly accessible. Clients encrypt their transaction data with this public key before submission, ensuring that transactions remain encrypted throughout the consensus process.

Decryption occurs within the TEE only when the vertex is about to be committed (Line 68).

Since replicas cannot access the transaction content before consensus is reached and the ordering is decided, they are prevented from manipulating transactions for MEV attacks. The TEE ensures that even Byzantine replicas cannot bypass this mechanism.

Thus, Fides effectively defends against MEV attacks. □