Efficient and DoS-resistant Consensus for Permissioned Blockchains


Abstract—Existing permissioned blockchain systems designate a fixed and explicit group of committee nodes to run a consensus protocol that confirms the same sequence of blocks among all nodes. Unfortunately, when such a permissioned blockchain runs in a large scale on the Internet, these explicit committee nodes can be easily turned down by denial-of-service (DoS) or network partition attacks. Although work proposes scalable BFT protocols that run on a larger number of committee nodes, their efficiency drops dramatically when only a small number of nodes are attacked.

In this paper, our EGES protocol leverages Intel SGX to develop a new abstraction called “stealth committee”, which effectively hides the committee nodes into a large pool of fake committee nodes. EGES selects a distinct group of stealth committee for each block and confirms the same sequence of blocks among all nodes with overwhelming probability. Evaluation on typical geo-distributed settings shows that: (1) EGES is the first permissioned blockchain’s consensus protocol that can tolerate tough DoS and network partition attacks; and (2) EGES achieves comparable throughput and latency as existing permissioned blockchains’ protocols.

1 Introduction

A blockchain is a distributed ledger recording transactions maintained by nodes running on a peer-to-peer (P2P) network. These nodes run a consensus protocol to ensure consistency: nodes confirm the same sequence of blocks (no forks). Each block contains the hash of its previous block, forming an immutable hash chain. A blockchain can be permissioned or permissionless. A typical permissionless blockchain does not manage membership for nodes and is usually attached with a cryptocurrency mechanism (e.g., Bitcoin [1]) to incite nodes to follow the blockchain’s protocol.

In contrast, a permissioned blockchain runs on a set of identified member nodes and can leverage the mature Byzantine Fault-Tolerant (BFT) protocols [2], [3], [4], [5] to achieve better efficiency (i.e., throughput and latency on confirming blocks). This paper focuses on permissioned blockchains because their decoupling from cryptocurrencies has facilitated the deployment of many real-world general data-sharing applications, including a medical chain among UK hospitals [6], IBM supply chains [7], and the Libra payment system [8].

For performance and regulation reasons (e.g., meeting the honesty threshold of BFT protocols [9]), a permissioned blockchain (e.g., Hyperledger Fabric [10]) typically runs its consensus protocol on a static and explicit committee. This static committee approach is already robust for a permissioned blockchain among a small scale of enterprises [7].

Unfortunately, as permissioned blockchains become popular and are deployed in large scales on the Internet, this static committee approach is vulnerable to targeted Denial-of-Service (DoS) and network partition [11], [12], [13] attacks. For instance, Libra [8] aims to build a global payment system (there are more than 5,000 banks in the US [14]) and identifies DoS attacks as a significant threat, but it only provides partial mitigation [12, 22].

Indeed, great progress has been made in designing scalable BFT protocols (e.g., SBFT [4]) running on a larger group of committee nodes and tolerating more nodes being attacked. However, these protocols designate a small number of committee nodes to finish critical tasks (e.g., combing ACKs), making these protocols’ efficiency drop dramatically if these nodes are under DoS attacks [12, 22].

With recent DoS attacks lasting for days [15], [16], tolerating such attacks is crucial, yet challenging, for applications deployed on permissioned blockchains.

To address such vulnerabilities of static committees, a promising direction is to adopt the dynamic committee merit from permissionless blockchain systems [11], [17]. These systems select a distinct committee for each block to handle the frequent leaving of nodes and to provide fairness and can ensure liveness even if a committee fails to confirm a block.

Simply applying the dynamic committee approach, however, cannot ensure DoS-resistance. Another crucial requirement is unpredictability: the identifies of nodes in a committee must be unpredictable to the attacker before the committee tries to achieve consensus on a block. Otherwise, the attacker can adaptively attack the ready-to-be committee and cause the system stuck. For instance, ByzCoin [18] lets the proof-of-work winners of recent blocks be the committee, but these explicit nodes are easily targeted by a DoS attacker, causing ByzCoin to lose liveness permanently [19].

To the best of our knowledge, Algorand is the only work that can tolerate targeted DoS attacks. However, as Algorand is designed for permissionless blockchains, it confirms a block with up to 15 rounds and minute-level latency (discussed in [22]), making it unsuitable for general data-sharing applications on permissioned blockchains (e.g., Libra).

This paper aims to explore the new design point of building a permissioned blockchain’s consensus protocol that adopts the unpredictable dynamic committee merit to defend against targeted DoS or partition attacks, and at the same time, achieves comparable efficiency as existing BFT protocols (e.g., SBFT [4] has second-level latency).

A main obstacle is to ensure that any selected committee
meets the honesty requirement for byzantine problems: for consistency, each committee must have at most one-third of nodes being malicious [9]. Permissionless blockchains meet this requirement by selecting committees based on nodes’ wealth in the built-in cryptocurrency mechanism (i.e., proof-of-stake), but cryptocurrencies are usually unavailable in permissioned blockchains. Consequently, to meet such a requirement, one has to unrealistically assume that almost all member nodes (>90%) are honest (calculated in §2).

Fortunately, recent work (e.g., Microsoft CCF [20], REM[21]) shows that the code integrity feature of SGX can regulate the misbehavior of blockchain nodes. For instance, a recent implementation [23] of MinBFT leverages SGX to ensure that a node cannot send conflicting messages to different nodes and is incorporated into Hyperledger [10].

We present **EGES** [1] the first efficient consensus protocol that can tackle targeted DoS or partition attacks for a permissioned blockchain. **EGES** adopts the dynamic committee merit to select a distinct committee for confirming each block. To defend against DoS or partition attacks targeting the committees, we leverage the integrity and confidentiality features of SGX to present a new abstraction called **stealth committee**.

**EGES**’s stealth committee has two new features. First, **EGES** selects a stealth committee in SGX: the selection progress has no communication among committee nodes, and the selection result cannot be predicted from outside SGX. This ensures that a committee node stays stealth (cannot be targeted by the attacker) before sending out its protocol messages. Second, when nodes in a committee are trying to confirm a block, **EGES** hides these committee nodes into a large pool of fake committee nodes that behave identically as the real ones observed from outside SGX, so that an attacker cannot identify the real committees.

However, even equipped with SGX and stealth committee, it is still challenging to efficiently ensure both consistency (i.e., no two member nodes confirm conflicting blocks) and reasonable liveness (i.e., allow non-empty blocks to get confirmed) in the asynchronous Internet due to the FLP impossibility [24]. Specifically, suppose a committee node $x$ for the $n^{th}$ block fails to receive the $(n-1)^{th}$ block after a timeout, $x$ cannot distinguish whether it is because the committee for the $(n-1)^{th}$ block failed to confirm the $(n-1)^{th}$ block, or because $x$ itself does not receive the confirmed $(n-1)^{th}$ block due to network problems. As the committee nodes for the $(n-1)^{th}$ block may be under DoS attacks and be unreachable, $x$ must have a mechanism to distinguish these two scenarios in order to maintain both consistency and reasonable liveness in **EGES**.

**EGES** tackles this challenge using simple probability theory. **EGES**’s committee for each block contains one proposer and $n_A$ (e.g., 300) acceptors, randomly and uniformly selected from all nodes. The proposer broadcasts its block proposal to all nodes by P2P broadcasts and seeks quorum ACKs from the acceptors. **EGES** models the randomly selected acceptors as a sampling of the delivery rate of the proposal in the P2P overlay network [11]. In the previous example, **EGES** confirms the proposal for the $(n-1)^{th}$ block only if the proposal is delivered to a large portion of member nodes; if multiple rounds of the sampling show that very few nodes have received that proposal for the $(n-1)^{th}$ block, nodes in **EGES** consistently confirm the $(n-1)^{th}$ block as an empty block (with an overwhelming probability).

In sum, **EGES** efficiently enforces consistency and can defend against targeted DoS or partition attacks. Specifically, **EGES** defends against such attacks by (1) letting committee nodes stay stealth before they start achieving consensus for a block, (2) using fake committee nodes to conceal real committee nodes while they are achieving consensus for a block, and (3) switching to a different committee and consistently confirming a block even if the attacker luckily guesses most real committee nodes for this block.

In essence, **EGES**’s stealth committee is a moving target defense approach [25, 26]. **EGES** unpredictably replaces the critical components (i.e., the committee) in a large system so that a DoS attacker cannot launch effective attacks targeting these components. **EGES**’s consensus protocol also achieves good efficiency because confirming a block in a gracious run (e.g., the proposer can reach most acceptors) only involves two P2P broadcasts and a half UDP round trip [35]. We provide a rigorous analysis of **EGES**’s DoS-resistance and proof of **EGES**’s consistency guarantee in §5.

We implemented **EGES** using the codebase from Ethereum [27] and compared **EGES** with nine consensus protocols for blockchain systems, including five state-of-the-art efficient BFT protocols for permissioned blockchains (BFT-SMaRt [28], SBFT [3], HotStuff [5]), two SGX-powered consensus protocols for permissioned blockchains (Intel-PoET [30] and MinBFT [31]), the default consensus protocol in our codebase (Ethereum-PoW [27]), and two permissionless blockchains’ protocols that run on dynamic committees (Algorand [11] and Tendermint [17]). We ran **EGES** on both our cluster and AWS. Evaluation shows that:

- **EGES** is robust. Among all consensus protocols for permissioned blockchains, **EGES** is the only protocol that can defend against targeted DoS and network partition attacks, by both a theoretical analysis [10] and evaluation [8, 2].
- **EGES** is efficient. **EGES** confirms a block with 3000 transactions in less than two seconds in typical geo-distributed settings, comparable to evaluated consensus protocols that cannot tolerate targeted DoS attacks.
- **EGES**’s throughput and latency are scalable to the number of nodes. When running 10k nodes, **EGES** showed 2.3X higher throughput and 16.8X lower latency than Algorand with 10k nodes.

### Table 1: Comparison of **EGES** to baseline protocols

<table>
<thead>
<tr>
<th>Protocol name</th>
<th>DoS resistance</th>
<th>with SGX?</th>
<th>consensus model</th>
<th>number of nodes</th>
<th>tput (txn/s)</th>
<th>confirm latency (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EGES</td>
<td>high</td>
<td>yes</td>
<td>hybrid</td>
<td>300</td>
<td>3226</td>
<td>0.91</td>
</tr>
<tr>
<td>Algorand</td>
<td>high</td>
<td>no</td>
<td>BFT</td>
<td>10k</td>
<td>2624</td>
<td>1.15</td>
</tr>
<tr>
<td>PoET</td>
<td>high</td>
<td>yes</td>
<td>Hybrid</td>
<td>100</td>
<td>149</td>
<td>45.2</td>
</tr>
<tr>
<td>Ethereum</td>
<td>high</td>
<td>no</td>
<td>BFT</td>
<td>100</td>
<td>178</td>
<td>82.3</td>
</tr>
<tr>
<td>SBFT</td>
<td>medium</td>
<td>no</td>
<td>BFT</td>
<td>62</td>
<td>1523</td>
<td>1.13</td>
</tr>
<tr>
<td>MinBFT</td>
<td>low</td>
<td>yes</td>
<td>Hybrid</td>
<td>64</td>
<td>2478</td>
<td>0.80</td>
</tr>
<tr>
<td>BFT-SMaRt</td>
<td>low</td>
<td>no</td>
<td>BFT</td>
<td>10</td>
<td>4512</td>
<td>0.67</td>
</tr>
<tr>
<td>Tendermint</td>
<td>low</td>
<td>no</td>
<td>BFT</td>
<td>64</td>
<td>2462</td>
<td>1.31</td>
</tr>
<tr>
<td>HotStuff</td>
<td>low</td>
<td>no</td>
<td>BFT</td>
<td>64</td>
<td>2686</td>
<td>2.65</td>
</tr>
<tr>
<td>HoneyBadger</td>
<td>low</td>
<td>no</td>
<td>BFT</td>
<td>32</td>
<td>1078</td>
<td>9.39</td>
</tr>
</tbody>
</table>

1. **EGES** stands for Efficient, GEneral, and Scalable consensus.

---

**TABLE 1**: Comparison of **EGES** to baseline protocols. Analysis of DoS resistance is in [35], evaluation setup is covered in [35].
We discuss targeted DoS and network partition attacks which calls the hardware pseudo-random generator through which we will formally define in §3.

Our contribution is two-fold. First, EGES leverages SGX to explore the new design point of tackling DoS attacks while enforcing both consistency and reasonable liveness (including efficiency) for a permissioned blockchain in the asynchronous Internet. Second, we designed the new stealth committee abstraction and implemented EGES’s consensus protocol. EGES’s source code is available on github.com/hku-systems/eges. EGES can facilitate the deployments of various mission-critical, DoS-resistant permissioned blockchain applications on the Internet (e.g., e-voting [32] and payment [8]).

In the rest of the paper, §2 introduces EGES’s background and motivation; §3 defines the model of EGES; §4 gives a high-level overview of EGES; §5 introduces EGES’s consensus protocol; §6 analyzes the safety and liveness of EGES; §8 shows our evaluation, and §9 concludes.

2 Background and Related Work

We discuss targeted DoS and network partition attacks together in this paper because these two attacks cannot be effectively distinguished in an asynchronous network. Particularly, when a node cannot reach a remote node, the node cannot determine whether it is because the remote node is under DoS attacks or because these two nodes are partitioned in the network. Therefore, EGES maintains consistency by handling both cases together.

When discussing targeted DoS or partition attacks in this subsection, we assume that the attacker has an attack budget $B$ (e.g., $B = 300$): the attacker can adaptively target $B$ nodes at a time. This model is the same as Algorand’s, which we will formally define in §3.

2.1 Intel SGX

Intel Software Guard eXtension (SGX) [22] is a hardware feature on commodity CPUs. SGX provides a secure execution environment called enclave, where data and code execution cannot be seen or tampered with from outside. Code outside enclaves can enter an enclave by ECalls, and SGX uses remote attestations [22] to prove that a particular piece of code is running in an enclave on a genuine SGX-enabled CPU. SGX provides a trustworthy random source (sgx_read_rand), which runs the hardware pseudo-random generator through the RDRAND CPU instruction seeded by on-chip entropy sources [22]. Previous studies show that this random source complies with security and cryptographic standards and cannot be seen or tampered with from outside enclaves [33], [34].

Recent work shows that SGX can be leveraged to improve diverse aspects of blockchain systems. Intel’s PoET [30] replaces the PoW puzzles with a trusted timer in SGX; EGES is more efficient than PoET [35]. REM [21] uses SGX to replace the useless PoW puzzles with useful computation (e.g., big data), orthogonal to EGES. Microsoft CCF [20] (originally named CoCo) is a permissioned blockchain platform using SGX to achieve transaction privacy, but it does not include a DoS-resistance approach. SciFi [35] uses SGX’s attestations to establish reliable identities of nodes and maintains their identities on the blockchain, which is adopted in EGES [7.1].

Ekiden [56] and ShadowEth [57] offload the execution of smart contracts to SGX-powered nodes to avoid redundant execution and to preserve privacy; TEEChain [55] uses SGX to build an efficient and secure off-chain payment channel; Town Crier [59] uses SGX to build a trustworthy data source for smart contracts; Tesseract [40] uses SGX to build a cross-chain coin exchange framework; Obscuro [41] uses SGX to improve bitcoin’s privacy; these systems do not focus on consensus protocols and are orthogonal to EGES.

2.2 Consensus for Permissioned Blockchains

We briefly introduce recent notable consensus protocols for permissioned blockchains, which are also EGES’s evaluation baselines. Overall, all these protocols run on a static committee. To ensure liveness under a DoS attacker with an attack budget of $B$, these protocols must scale to $3 \times B + 1$ nodes (for BFT protocols) or $2 \times B + 1$ nodes (for SGX-powered protocols). However, to our best knowledge, no existing protocol can achieve such scalability.

BFT-SCaRt [2] is an optimized implementation of PBFT [3]. As each node broadcasts consensus messages to all other nodes, BFT-SCaRt has $O(n^2)$ message complexity to the number of committee nodes, resulting in poor scalability. Its paper [2] only evaluated up to 10 nodes. SBFT [1] is a scalable BFT protocol that uses a new type of committee nodes called collectors. Node sends its consensus messages to only $e$ (usually $e < 8$) explicit collectors who will then broadcast a combined message using the threshold signature. SBFT’s fast path can commit a block if fewer than $c$ nodes have failed; however, SBFT’s performance drops dramatically if an attacker targets the $c$ collectors [36].

HotStuff [5] is a BFT protocol optimized for frequent leader changes, and Libra [8] leverages Hotstuff to tolerate targeted DoS attacks on leaders. However, since Hotstuff reports a near-linear increment of latency with an increasing number of nodes, it only evaluated up to 128 nodes, where an attacker can DoS attack or partition one-third of all nodes rather than finding the leader. HoneyBadger [29] uses randomization to remove the partial synchrony assumption of PBFT. However, both its paper and our evaluation shows that HoneyBadger achieves high latency due to multiple rounds of its asynchronous byzantine agreements.

MinBFT [31] is an SGX-powered BFT protocol that has the same fault model as EGES. MinBFT reduces the number of rounds in PBFT and can tolerate more node failures (one-half instead of one-third), but MinBFT still has $O(n^2)$ message complexities, so its performance is not scalable to the number of nodes.

2.3 Consensus for Permissionless Blockchains

Existing permissionless blockchains can be divided into two categories based on how they confirm blocks. The
first category confirms block with variants of the longest-chain rule (i.e., Nakamoto consensus [11]), including BitCoin [11], Ethereum [27], BitCoin-NG [42], Snow-White [43], Ouroboros [44], Paros [45], Genesis [46], and GHOST [47]. Specifically, each node asynchronously selects the longest chain it received and confirms a block when there are \( k \) blocks succeeding it. However, waiting for \( k \) more blocks leads to a long confirm latency, and previous work [48] shows that this \( k \) must be large enough to ensure consistency. Moreover, the longest-chain rule cannot ensure consistency under partition attacks [49], [13], [11]. Intuitively, during a network partition, each partition will independently grow a chain; if these chains diverge for more than \( k \) blocks, nodes in different partitions will confirm conflicting blocks.

The second category of permissionless blockchains confirms blocks using the committee-based BFT approach, which can confirm a block as soon as the BFT consensus is achieved. This category includes Algorand [11], ByzCoin [18], Tendermint [17], and PeerCensus [50]. These systems select distinct (dynamic) committees for different blocks based on the content (e.g., nodes’ wealth) on the blockchain for fairness and for handling nodes joining or leaving. Similar to EGES, these systems run a tailored consensus protocol (BA* in Algorand [11], Tendermint [17], Tenderbake [51], and Tenderand [52]) on dynamic committees to confirm blocks.

However, these protocols cannot be ported to a permissioned blockchain because of the tight coupling with cryptocurrency. For instance, although Tendermint [53] and Tenderbake [51] are described as stand-alone BFT protocols, they assume that in any committee, fewer than one-third of nodes are malicious. In a permissioned blockchain without cryptocurrency, if we want to ensure that any randomly selected committee (say 100 nodes) from a large number of (say 10k) nodes meets this requirement with overwhelming probability (> \( 1 - 10^{-10} \)), we need to assume over 91% of all nodes being honest (by the hypergeometric distribution), which is an overly-strong assumption for a practical large-scale blockchain system (e.g., a global payment system [8]) on the Internet.

Moreover, these systems (except Algorand) cannot ensure liveness under targeted DoS attacks because they select committees in a predictable way so that all nodes can verify the identities of committees. For instance, ByzCoin [19] lets the proof-of-work winners of recent blocks be the committee. However, these nodes with explicit identities are easily targeted by a DoS attacker, and ByzCoin may lose liveness permanently [19] if more than one-third of these nodes are attacked. Algorand defends against targeted DoS attacks by letting each node use verifiable random functions to determine its committee membership. We provide a detailed comparison showing why EGES is more efficient than Algorand in [54].

3 System Model
EGES is a consensus protocol for a permissioned blockchain running on \( M \) member nodes (nodes for short) connected with an asynchronous network. Each node is equipped with an attested SGX enclave running the EGES protocol [54].

EGES adopts the hybrid fault model used in existing SGX-powered consensus protocols [54], [51], [55], where each node has a trusted module (i.e., the SGX enclave) that will only fail by crashing, and all other components can behave arbitrarily.

EGES has the following design goals:

- **Safety (consistency).** EGES ensures safety in an asynchronous network. Formally, if a node confirms a block \( b \) as the \( n^{th} \) block on the blockchain, the probability that another node confirms \( b' \neq b \) as the \( n^{th} \) block is overwhelmingly low (< \( 10^{-10} \)).

- **DoS-resistance (liveness).** In addition to safety, EGES can make progress (i.e., allow non-empty blocks to be confirmed) with two additional assumptions about the DoS attacker’s capability as described below.

SGX’s threat model. EGES has the same threat model for SGX as typical SGX-based systems [56], [57], [58], [23], [59]. We trust the hardware and firmware of Intel SGX, which ensures that code and data in an enclave cannot be seen or tampered with from outside. We trust that the remote attestation service can identify genuine SGX devices from fake ones (e.g., emulated with QEMU). Side-channel and access pattern attacks on SGX are out of the scope of this paper. Moreover, the adversary cannot break standard cryptographic primitives, including public-key based signatures and collision-resistant hash functions.

Communication model. EGES maintains safety in an asynchronous network, where network packets can be dropped, delayed, or reordered arbitrarily. Nodes may be nonresponsible, either due to going offline or due to targeted DoS attacks (e.g., botnet DoS attacks [15]) by a DoS attacker. When a node cannot reach a remote node, the node cannot determine whether the remote node is under DoS attack (or is offline) or the network packets are delayed.

To achieve liveness, EGES has the “strong synchrony” assumption, same as Algorand [11]. Specifically, EGES assumes that messages between two nodes not under DoS attacks can be delivered within a known time bound, and the assumptions about the attacker’s capability are described below.

Nodes are connected with a P2P overlay network, same as existing large-scale blockchain systems [11], [11]. Specifically, each node has a P2P module connecting to a random set of other nodes and relays messages using the gossip protocol [50].

A node’s P2P module is outside SGX and can be controlled by the attackers; the attacker can partition some nodes from other nodes [12], [13] or selectively pass consensus messages to nodes’ SGX enclaves. However, such manipulations are already included in EGES’s asynchronous network assumption. For safety, EGES leverages the sampling merit to estimate the delivery rate of a specific block proposal and derives overwhelming probability, regardless of how nodes are connected. For liveness, the adversary can control the P2P modules of a number of nodes with the restriction of the adversary’s attack budget described below.

Capability of DoS attackers. EGES has three assumptions on the capability of a DoS attacker, same as Algorand [11] and existing move target defense (MTD) systems [23], [24].

First, the adversary has a targeted attack budget \( B \) (e.g., \( B = 300 \) or 10% of the total number of nodes): the adversary cannot constantly cause more than \( B \) targeted nodes in EGES to be nonresponsive. This budget is adaptive: the adversary can attack different nodes at different times, but the number
of attacked nodes at a time cannot be constantly larger than $B$.

Second, the attacker can conduct ubiquitous DoS attacks (without targeting specific nodes) or partition a number of nodes from other nodes (e.g., by manipulating nodes’ P2P modules [13, 12]). However, the P2P overlay network should have a large enough portion (e.g., 65%) of nodes connected. We provide a quantitative analysis of how EGES can preserve liveness under such attacks in [§6.3].

Third, the adversary cannot constantly succeed in mounting an attack targeting a node within the time window for the node to send out an EGES protocol message. Specifically, EGES protocol messages are larger than the network maximum packet size and are fragmented into multiple packets; an EGES committee node’s identity is unknown to the adversary before sending out the first packet, and we assume that the adversary cannot mount targeted DoS attacks until the node sends out all packets belonging to this message (at most hundreds of kB and can be sent within one second).

EGES already assumes a strong enough attacker for practical distributed systems on the Internet. As pointed out by Algorand [11], a more powerful adversary than our model usually controls the internet service provider and can prevent all EGES nodes from communicating at all: no practical system can ensure liveness under such a strong adversary, and such attacks can be easily detected. We will provide a rigorous analysis of EGES’s DoS resistance in [§6.2].

4 EGES’S HIGH-LEVEL IDEA

EGES has three important features to achieve DoS resistance. First, EGES randomly selects a distinct group of committee for each block. The selection is done inside the SGX enclaves of a previous committee, and the selection result is encrypted on the confirmed blockchain. By doing so, a committee node can determine its committee membership without interactions with other nodes, making it stay stealth before trying to achieve consensus on its block.

Second, when a committee is achieving consensus for a given block, EGES uses fake committee nodes to conceal the real ones by sending dummy messages. Since whether a node is a real committee node is only known within the node’s SGX enclave, and the dummy messages are of the same format as real ones, a DoS attacker cannot distinguish the real committee nodes from the fake ones. Therefore, the attacker must have an unrealistic large attack budget to attack all the real and fake committees; otherwise, he has to randomly guess who are the real ones.

Third, even if the attacker luckily guesses the real ones (and he may eventually succeed if tried persistently), EGES can ensure safety with overwhelming probability leveraging the delivery rate on all nodes of the unique proposal for each block. Specifically, even if a committee cannot confirm its own block, committees for subsequent blocks can help to consistently confirm this block by repetitive querying. This feature is in contrast to most existing consensus protocols (i.e., all except Algorand [11]), where the system must wait statically until a quorum of nodes become reachable.

EGES’s consensus protocol has three parameters, $n_A$ (default 300), $\tau$ (default 99%), and $D$ (default 4), where $n_A$ is the number of acceptors, $\tau$ is the quorum ratio, and $D$ is the finalize depth for an empty block. We will show how to select these parameters in [§6.3] and how these parameters affect EGES’s performance in [§8.3].

For each block index $n$, EGES selects only one committee from all nodes. The committee contains two types of nodes: one proposer ($P_n$) and a group of stealth acceptors ($A_n$) with the count of $n_A$. EGES ensures the following invariant (see [§6.1] for proof).

Invariant 1. For any block index $n$, at most one unique block proposal (proposal$_n$) is generated; a node can only confirm proposal$_n$ or a default empty block (empty$_n$) as its $n$th block.

EGES uses different steps to confirm proposal$_n$ or empty$_n$. We make the steps of confirming proposal$_n$ as lightweight as possible for high efficiency in gracious runs.

Confirming proposal$_n$. In a gracious run, EGES confirms proposal$_n$ in three steps: $P_n$ broadcasts proposal$_n$ with a propose request through the P2P network; acceptors send ACKs to $P_n$ after receiving proposal$_n$; $P_n$ broadcasts a finalize message confirming proposal$_n$ once receiving ACKs from a quorum ($\tau \times n_A$) of acceptors.

Confirming empty$_n$. For consistency, when confirming empty$_n$, EGES must ensure that no node has confirmed proposal$_n$. Existing consensus protocols on static nodes use view-change protocols [61, 3, 17, 5] that count how many nodes have sent out ACKs and leverage the quorum intersection property [62] to determine whether a proposal may have been confirmed. However, in EGES, this method is not viable because EGES must ensure liveness even if most nodes in $A_n$ are DoS attacked after sending out their ACKs.

EGES’s protocol for confirming empty$_n$ leverages the idea of repeated sampling to predicate that the probability of proposal$_n$ having been confirmed is overwhelmingly low. If proposal$_n$ is confirmed on some nodes, proposal$_n$ should have been delivered to a large-enough portion of nodes in the P2P network; this is because confirming if proposal$_n$ needs quorum ACKs from nodes in $A_n$, and $A_n$ is uniformly selected from all nodes. Therefore, if one repetitively ($\Omega(D)$) samples many nodes ($\Omega(T \times n_A)$ from all nodes, and no node has received proposal$_n$, she can predicate only a small portion of (or no) nodes have received proposal$_n$, and thus the probability of proposal$_n$ having been confirmed is overwhelmingly low.

To be DoS-resistant, these multiple rounds of checking must be initiated by different nodes, so EGES lets the proposers for subsequent blocks (i.e., $P_{n+1}$, $P_{n+2}$, etc.) do such samplings at the same time of seeking ACKs for their own proposals. Suppose $P_n$ failed before sending out proposal$_n$, the next 4 proposers will all report that no node has received proposal$_n$, so empty$_n$ can be confirmed as empty together with proposal$_{n+1}$-proposal$_{n+4}$.
Comparison with Algorand. EGES and Algorand both select a distinct committee for each block in an unpredictable way, and both use the delivery rate of a block proposal to confirm a block. Algorand leverages its built-in cryptocurrency to incite committee nodes to follow its protocol (i.e., proof of stake). However, even if one runs Algorand within SGX in a permissioned blockchain, there are still two major design differences making EGES more efficient than Algorand.

First, Algorand uses verifiable random functions (VRF) to determine committees, so it can only control the expected count of proposers for each block without an exact number (1 \( \sim \) 70) in their experiment [11]. This design makes Algorand’s consensus protocol not responsive [3]. Informally, a responsive protocol lets nodes wait for a number of messages rather than a large amount of time in each protocol step, which ensures a good performance when the network is in good condition. For each block, Algorand selects 1 \( \sim \) 70 proposers, and each proposer broadcasts a block proposal with a distinct priority level. Then, Algorand selects one of these proposals by letting nodes vote for the received proposal with the highest priority. Since the total number of proposers is unknown, each node must wait for a conservatively long time (e.g., 10s) before voting to ensure it received most proposals. In contrast, EGES selects one proposer for each block, without the necessity for the selection progress, and EGES’s protocol is responsive [3].

Second, EGES adopts an optimistic design while Algorand adopts a pessimistic design. Specifically, Algorand uses a heavy step for both confirming a non-empty block and confirming an empty block. In contrast, EGES optimistically makes its gracious runs (i.e., confirming proposal\(_i\)) fast and shifts the burden of maintaining consistency to the rare failure cases (i.e., confirming empty\(_i\)).

Why does EGES use SGX? EGES chooses to use SGX for three main reasons. First, EGES uses SGX to regulate the behaviors of randomly selected committee nodes. As discussed in [2.3] using randomly selected committees in a permissioned blockchain may result in an unsolvable byzantine problem where more than one-third of committee nodes are malicious [9].

In EGES, each node have a private key that is only visible within the node’s SGX enclave, and the corresponding public key works as the node account saved in all other nodes’ SGX enclaves [2.1]. A valid protocol message must carry a valid signature using the sender node’s private key, proving that the message is generated in the sender node’s enclave with code integrity. By doing so, a node cannot equivocate (i.e., sending conflicting messages to different other nodes) or forge protocol messages (e.g., a proposer sending out a finalize message without receiving quorum ACKs).

Second, EGES leverages SGX to make its committee’s identities stealth: only a node’s enclave knows whether itself is a committee member for the current block. This not only enables EGES to maintain practical liveness under DoS attacks but, more importantly, makes EGES’s consistency model resistant to targeted attacks. Specifically, EGES leverages probability theory to model randomly selected acceptors as a uniform sampling of the delivery rate of a block proposal in the P2P network (same as Algorand [11]). If acceptors’ identities are public, an attacker can selectively transfer or drop packets towards them, breaking EGES’s safety.

Third, the usage of SGX enables EGES to select a stealth committee with a known count. As discussed above, this makes EGES more efficient than Algorand’s.

5 EGES Consensus Protocol

5.1 Protocol Preliminaries

Block structure. EGES adds one data field to the block structure of common blockchain systems [27], [1]: the encrypted committee identities for a future block [35.2]. EGES is oblivious to how transactions are stored or executed.

Each node’s local states. Each EGES node maintains three major local states: a local blockchain (the chain), a proposal cache (the cache), and a set of learntProposals. The cache is maintained in the node’s EGES enclave. When the node receives proposal\(_i\), it puts the proposal into the cache, in case the committees of future blocks query the delivery rate of proposal\(_i\).

Block status. Each block in a node’s chain has three states: undecided, finalized, and confirmed. An undecided \( n^{th} \) block can only be empty\(_n\). A node appends empty\(_n\) to its chain when the node triggers a timeout waiting for the finalize message for the \( n^{th} \) block; the block is in the undecided state because the node cannot determine (for now) whether it should confirm proposal\(_i\) or empty\(_n\).

A finalized \( n^{th} \) block can be either proposal\(_i\) or empty\(_n\). EGES ensures that, if a node’s \( n^{th} \) is finalized as one choice, no other nodes will finalize the \( n^{th} \) block as the other choice. A node appends finalized proposal\(_i\) if it receives the finalize message for proposal\(_i\); a node changes the empty\(_n\) from the undecided state to finalized if the node can predicate that no node has finalized proposal\(_i\) [35.3].

A node confirms its finalized \( n^{th} \) block if all blocks with indices smaller than \( n \) in its chain are finalized. Note that although EGES may finalize blocks out of order, EGES confirms blocks sequentially, same as typical blockchains [27], [10].

The chain on each node is divided into two parts: the confirmed part and the unconfirmed part. We use MC to represent the maximum confirmed index and \( U \) to represent the indices of undecided blocks in chain. The confirmed part of chain (i.e., indices \( \leq MC \)) are cryptographically-chained by hash values and can be saved out of the enclave and get executed, while unconfirmed parts are saved in the EGES enclave.

The learntProposals is the set of known proposals for undecided blocks on this node and is saved in EGES enclave.

Membership and key managements. In EGES, each node \( i \) has a key pair \( \langle pk_i, sk_i \rangle \), with the public key \( pk_i \) as its account, and its secret key \( sk_i \) is only visible within its enclave: even this node’s administrator cannot see the plain-text of its secret key. We use the notations from PBFT [3]: we denote a message \( m_1 \) sent to node \( i \) encrypted by \( i \)’s public key \( pk_i \) as \( \{ m_1 \}_pk_i \); we denote a message \( m_2 \) generated by node \( i \)’s enclave and signed by \( sk_i \) as \( \{ m_2 \}_sk_i \). For efficiency, EGES only signs on message digests.

To ease understanding, in this section, we assume that EGES runs on a fixed membership, where all nodes’ accounts (public keys) are loaded to nodes’ EGES enclaves a priori, and all nodes’ EGES enclaves are attested. We will show how EGES supports dynamic membership and attestations in [37].
5.2 Selecting a Stealth Committee

For each block, EGES selects a committee, including one proposer and \( n_A \) acceptors, in an unpredictable way without communication among nodes.

The committee members for the \( n^{th} \) block is selected in the EGES enclave of \( P_{n-lb} \), and these committee nodes’ identities are encrypted in the \( (n-lb)^{th} \) block. \( lb \) (lookback) is a system parameter and needs to be large enough (e.g., the number of blocks confirmed in days) to ensure that even when the network condition is poor, and new blocks cannot be confirmed in time, EGES can still derive committees for future blocks. We assume that the first \( lb \) committees’ identities are encrypted in the genesis (0\(^{th}\)) block by a trusted party, or that the blockchain is bootstrapped in a controlled domain for at least \( lb \) blocks. Note that the value of \( lb \) does not affect EGES’s safety.

Occasionally, a node may be selected as the committee for a future block and then leave the system, which EGES already tolerates as a failed node. If the \( (n-lb)^{th} \) block happens to be an empty block, EGES uses the committee identities encrypted in the \( (n-2lb)^{th} \) block (and identities in the \( (n-3lb)^{th} \) block if the \( (n-2lb)^{th} \) block is also empty, recursively). Although this proposer’s identity is already explicit when confirming the \( (n-lb)^{th} \) block and may be targeted, EGES can tolerate it as a failed proposer and uses subsequent committees to confirm empty \( n \).

\[ P_{(n-lb)} \] selects the committee for the \( n^{th} \) block with two steps, which are done in \( P_{(n-lb)} \)'s EGES enclave to ensure both integrity (i.e., an attacker cannot control the selection) and confidentiality (i.e., an attacker cannot know the selection result). In the first step, \( P_{(n-lb)} \) randomly selects the committee members from all member nodes following the uniform distribution. Recall that the member list is loaded in the EGES enclave on each node \([35.1]\), so \( P_{(n-lb)} \) simply selects \( n_A+1 \) nodes from the list using the SGX’s trustworthy pseudo-random number generator as the random source, which has been shown to be cryptographically-secure and cannot be seen or tampered with from outside enclave \([32.1]\).

---

**TABLE 2: EGES’s consensus messages’ fields.** Blue fields are only used in the checking mode and are left as nil in the normal mode.

**Algorithm 1: the proposer for the \( n^{th} \) block**

1. \( n_A \) ← the number of acceptors
2. \( blk \) ← the content of the \( n^{th} \) block to propose
3. \( \text{function normalPropose()} \)
4. \( \text{bcast (Propose, blk, MC, nil, nil)} \)\(_{σ_{mc}} \)
5. \( \text{upon receiving (ACK, n, pk, nil)} \)\(_{σ} \)
6. \( \text{ACKs.insert(pk)} \)
7. \( \text{if ACKs.count >= } τ \times n_A : \) \( \text{bcast (Finalize, n, nil, nil)} \)\(_{σ_{mc}} \)
8. \( \text{function checkPropose()} \)
9. \( \text{u}_m \) ← max(\( U \))
10. \( \text{if cached}_u_m |\) \( |\) learntProposals[\( u_m ] |\) \( |\) nil = nil ; \)
11. \( \text{upon receiving (ACK, n, pk, nil)} \)\(_{σ} \)
12. \( \text{else: proposal}_u_m ← \) \( \text{the proposal for index } u_m \)
13. \( \text{bcast (Propose, blk, MC, proposal}_u_m ) \)\(_{σ_{mc}} \)
14. \( \text{learned = []} \)
15. \( \text{upon receiving (ACK, n, pk, notifications)} \)\(_{σ} \)
16. \( \text{ACKs.insert(pk)} \)
17. \( \text{learned.insert(notifications)} \)
18. \( \text{if ACKs.count >= } τ \times n_A : \) \( \text{bcast (Finalize, n, proposal}_u_m , \text{learned)} \)

---

**Algorithm 2: all nodes’ action**

\( /* \) All message senders’ memberships and signatures are verified, omitted in all algorithms for brevity \( */ \)
1. \( \text{upon receiving msg = (Propose, n, blk, MC, nil, nil)} \)\(_{σ} \)
2. \( \text{cache}[n] = \) msg
3. \( \text{if } MC < MC_{\text{p}} \text{ then ask peers for missing blocks} \)
4. \( \text{if hash}(sk_i, n) > \) threshold : \( \text{Reply fake (cover) ACKs message ‘false’ } \)
5. \( \text{upon receiving (Finalize, n, u_m, learned)} \)\(_{σ} \)
6. \( \text{if } u_m \neq \) nil :
7. \( \text{chain}[u_m].status ← \) finalized
8. \( U = U \setminus u_m \)
9. \( \text{if } |U| = 0 : \text{Confirm chain up to index } n (MC_\neg n) \)
10. \( \text{else:} \)
11. \( \text{learntProposals.insert(learnt)} \)
12. \( \text{for } u \in U \) \( \text{in descending order} : \)
13. \( \text{if canFinalize(u)} : \)
14. \( \text{chain}[u].status = \) finalized
15. \( \text{else: break // Empty blocks must be finalized in descending order.} \)
16. \( \text{upon timeout waiting for next block} \)
17. \( \text{chain.append(empt} \text{y, status = undecided)} \)
18. \( \text{function canFinalize(u)} : \)
19. \( \text{count = 0} \)
20. \( \text{for i = u + 1; i \leq chain.length; ++i :} \)
21. \( \text{if chain[i] = \) nil : \( \text{count++ ;} \)
22. \( \text{if count > D : return true} \)
23. \( \text{else: return false} \)

**Algorithm 3: an acceptor for the \( n^{th} \) block**

1. \( \text{upon receiving (Propose, blk, MC, U, proposal}_u_m ) \)\(_{σ} \)
2. \( r = \text{rand()} \)
3. \( \text{if } MC < MC_\text{p} \text{ then notify proposer to catch up (omitted)} \)
4. \( \text{if } |U| = 0 : \)
5. \( \text{reply (\{ACK, n, pk, nil, true, r\})} \)\(_{σ_{mc}} \)
6. \( /* \) Only the leader(i) can decrypt this message \( */ \)
7. \( \text{notifications = []} \)
8. \( \text{for } u \in U \) \( : \)
9. \( \text{if cached}_u \) \( |\) learnt : \( \text{notifications.append(cache[u]} \)
10. \( \text{reply (\{ACK, n, notifications, true, r\})} \)\(_{σ_{mc}} \)

---

Fig. 2: EGES’s consensus protocol.
In the second step, for each selected committee node, \( P_{(n-lb)} \) generates one certificate, which is the cipher-text of the concatenation of a predefined byte string and a random nonce (for making the cipher-text unpredictable), encrypted with that committee node’s public key. Then, \( P_{(n-lb)} \) includes these \((n_A+1)\) certificates in the \((n-lb)\)th block’s proposal. The first certificate is for the proposer, and the other \( n_A \) certificates are for acceptors. When a node confirms this block, it tries to decrypt one of these certificates using its own secret key in its enclave; if the node can get the predefined string, it predicates that it is a committee node for the \( n \)th block.

Despite using asymmetric cryptography, this mechanism is efficient in EGES because both encryption and decryption are done asynchronously off the consensus’s critical path. For encryption, since \( P_{(n-lb)} \)’s enclave knows it is the proposer for the \((n-lb)\)th block after confirming the \((n-2lb)\)th block, \( P_{(n-lb)} \) starts selecting the committees and encrypting the certificates as soon as confirming the \((n-2lb)\)th block. Similarly, the decryption is also off the critical path as the decryption result is used \( lb \) blocks later.

EGES’s committee selection mechanism is unpredictable and non-interactive because: (1) the random source cannot be seen or tampered with from outside the enclave of \( P_{n-lb} \) and the certificates can only be verified within a selected committee node’s enclave; and (2) the selection process is solely done within the EGES enclave of \( P_{n-lb} \). These two features ensure the committee nodes’ identities are not exposed during the selection, so the committee nodes cannot be targeted before sending out protocol messages for the \( n \)th block.

**Discussions.** EGES selects only one proposer for each block to achieve good efficiency: EGES only needs to achieve a binary consensus on whether to confirm the unique proposal by this proposer or a default empty block. For acceptors, however, an alternative design is to let each node independently determine whether it is an acceptor for the current block with a probability, and EGES only knows an expected total count. However, this alternative design will lead to a much larger quorum ratio (i.e., \( \tau \)) to ensure safety and thus worse liveness (quantitative analysis in §5.3).

### 5.3 Confirming a Block

A proposer \( P_n \) has two operation modes: normal mode and checking mode. \( P_n \) is in the normal mode if all blocks in its chain before \( u \) are confirmed (i.e., \( U = \emptyset \)), and \( P_n \) tries to confirm \( proposal_{n_u} \) quickly. Otherwise, \( P_n \) is in the checking mode: while proposing \( proposal_{n_u} \), it also checks the status of the undecided blocks in its chain.

**Normal mode.** Algorithm 1 Line 3–7 shows how a normal mode \( P_n \) tries to confirm \( proposal_{n_u} \) in a gracious run. First, \( P_n \) broadcasts a propose request through the P2P network carrying \( proposal_{n_u} \) and its MC. The MC value helps nodes align confirmed parts of their chain: if a node’s MC is smaller than the proposer’s, the node asks for the missing confirmed blocks from its peers [Algorithm 2 Line 3]. Upon receiving this propose request, an acceptor replies an ACK using UDP directly to \( P_n \) [Algorithm 3 Line 3]. Second, \( P_n \) waits for quorum \((\tau \times n_A)\) ACKs from \( \alpha_{n_u} \). \( P_n \) does not know which nodes are acceptors, but EGES’s ensures that a non-acceptor cannot send valid ACKs [43]. Third, \( P_n \) broadcasts a finalize message; on receiving this message, a node finalizes \( proposal_{n_u} \).

**Checking mode.** \( P_n \) is in the checking mode if it has undecided blocks (i.e., \( U \) is non-empty), and its workflow is shown in Algorithm 1 Line 12–19. \( P_n \) checks the status of its undecided blocks and tries to finalize them (if possible) by adding additional fields to the propose message.

Each \( u \in U \) in \( P_n \)’s chain is categorized into one of the two types: (1) if \( P_n \) has learnt the unique \( proposal_{u_m} \) either from the propose messages from \( P_n \) from the notifications of other nodes, we call \( u \) a “known undecided” block; (2) otherwise we call \( u \) “unknown undecided”. For each unknown undecided block \( u \), \( P_n \) tries to learn \( proposal_{u_m} \) from \( \alpha_{n_u} \). If \( P_n \) learns the proposal, \( P_n \) carries it in the finalize message in order to let subsequent proposers finalize it; otherwise, \( P_n \) carries a message stating that most acceptors in \( \alpha_{n_u} \) never received \( proposal_{u_m} \), and a node receiving this message finalize \( empty_{n_u} \) if the node received such messages form more than \( D \) consecutive proposers (Algorithm 2 Line 18).

For known undecided blocks, \( P_n \) helps to finalize only \( proposal_{u_m} \) where \( u_m = max(U) \) and leaves other blocks for subsequent proposers. We will explain later that only finalizing \( u_m \) is essential for EGES’s safety (Figure 5).

Figure 4 shows how a proposer \( P_{202} \) helps to finalize an undecided proposal \( proposal_{100} \). Suppose \( P_{100} \) failed just before broadcasting its finalize message. Therefore, the 100th block’s state is undecided among all nodes. Then, \( P_{101} \) learns proposal \( proposal_{100} \) and carries it in its finalize message, and \( P_{102} \) learns it. Then \( P_{102} \) finalizes proposal \( proposal_{100} \) together with proposal \( proposal_{102} \). Moreover, since all blocks before 102 are finalized, the chain is confirmed up to 102.

Figure 4 shows how an undecided block is finalized as empty. Suppose \( P_{200} \) failed before broadcasting its proposal, and \( D = 4 \). When \( P_{201} \sim P_{204} \) asks whether their acceptors (\( \alpha_{201} \sim \alpha_{204} \)) receive \( proposal_{200} \), they get no positive answers. Therefore, these four blocks are all finalized, carrying a message stating that four samplings have been conducted on the delivery rate of \( proposal_{200} \), but no replied node has received \( proposal_{200} \). This indicates that the probability that \( proposal_{200} \) is finalized at some nodes is overwhelmingly low. Therefore, a node can independently finalize \( empty_{200} \) after which the chain is confirmed up to 204.

Figure 5 shows why it is essential that a checking mode proposer can only finalize \( proposal_{u_m} \) where \( u_m = max(U) \). Suppose we remove this restriction, and we consider the following scenario. (1) \( P_{200} \) fails after broadcasting \( proposal_{200} \) and only very few nodes received it: none of \( P_{201} \sim P_{204} \) learns \( proposal_{200} \). (2) The network is divided into two partitions A&B just before \( P_{204} \) broadcasting its finalize message; \( P_{204} \) is in partition A, so nodes in partition A confirm \( proposal_{204} \) and confirm \( empty_{200} \). (3) \( P_{205} \) and \( P_{206} \) are in partition B, so they timeout waiting for the finalize message for \( proposal_{204} \) and mark the 204th block as undecided. (4) \( P_{205} \) learns \( proposal_{200} \) from one node from \( \alpha_{205} \) (who happens to be in the very few nodes) and carries \( proposal_{200} \) with its finalize message, which is learnt by \( P_{206} \). (5) \( P_{206} \) finalizes \( proposal_{200} \) and causes inconsistency: nodes in partition A confirm \( empty_{200} \), while nodes in partition B confirm \( proposal_{200} \).

The inconsistency happens because, without this restriction, when a node finalizes \( empty_{n_u} \) it only ensures proposers for blocks with index \( n + D \) has not finalized \( proposal_{u_m} \).
adding this restriction helps to ensure that proposers for blocks with index \( n + D \) cannot finalize \( \text{propos}1_n \). Specifically, if this restriction presents in the previous example, nodes in partition B need to first finalize the empty\(_{204}\) finalizing \( \text{propos}2_{204} \). However, since \( \text{propos}2_{204} \) has already been delivered to a large portion of nodes, this inconsistency cannot happen. We show the proof in §6.1.

### 5.4 Handling DoS Attacks Targeting Proposers

In EGES, a proposer stays stealth before proposing its block, but its identity becomes explicit after broadcasting its proposal. If the proposer is DoS attacked at this time, this block cannot be finalized in time, impairing EGES’s liveness.

To mitigate this problem, we propose a new role of nodes called arbiter. An arbiter for a block index \( n \) does not generate new block proposals but only helps the proposer to finalize its proposal. For each block index \( n \), EGES has many arbiters (larger than the attack budget \( B \) ) doing the same tasks to tolerate DoS attacks on them.

Since we do not need to know the exact number of arbiters count and their identities, EGES simply let each node’s EGES enclave independently determine whether it is an arbiter for the \( n \)th block with a probability of \( p_n \) after receiving \( \text{propos}1_n \). An arbiter for the \( n \)th block broadcasts an arbit request following the same protocol as the proposer (Algorithm 1), and an acceptor responds to an arbit request with the same logic responding to a propose message.

**Discussions.** With the help of arbiters, a proposer’s critical task is only to send out its propose request, and the arbiters can help to finalize it. However, responding to both proposer and multiple arbiters makes acceptors targets of DoS attacks. Therefore, EGES lets normal nodes also randomly send out fake (dummy) ACKs to cover the real acceptors. (Algorithm 2 Line 4). Since real or fake ACKs are all encrypted with \( P_n \)’s public key, only \( P_n \)’s EGES enclave can decrypt them within its enclave and distinguish the real ones, so an attacker cannot know who are the real acceptors.

### 6 Security Analysis

#### 6.1 Safety with Overwhelming Probability

EGES ensures safety with overwhelming probability (i.e., \( > 1 - 10^{-10} \)). Formally, if a node confirms a block \( b \) as the \( i \)th block on the blockchain, the probability that another member node confirms \( b' \neq b \) as the \( i \)th block is \( < 10^{-10} \).

We prove the safety guarantee of EGES by induction: suppose EGES guarantees safety from the 0th block to the \( (n - 1) \)th block (hypothesis 1), and we prove that there is only one unique block that can be confirmed as the \( n \)th block among nodes in the blockchain. The base case is trivial because all nodes start from the same 0th block.

**Lemma 1.** If two nodes have the same maximum confirmed block in their chain (i.e., \( MC = n - 1 \) due to hypothesis 1), then during consensus for the \( (n + i) \)th block where \( i \geq 0 \), as long as \( MC \) is not changed, these two nodes see the same member list.

**Proof.** Proving this lemma is trivial if EGES works on a fixed member list, and we will show in §7.1 that EGES’s protocol for dynamic memberships also ensures this lemma.

**Lemma 2.** (Invariant [I] in §4): at most one proposal can be generated for the \( n \)th block.

**Proof.** This lemma is proved by two steps. First, as the proposer for the \( n \)th block is encrypted in the \( (n - lb) \)th block, and the \( (n - lb) \)th block is the same among nodes because of hypothesis 1. Therefore, there is only one proposer (may have failed) for the \( n \)th block. Second, this proposer generates at most one proposal, and non-proposer nodes cannot generate valid proposals for the \( n \)th block because EGES’s consensus module runs in SGX.

**Proof of the induction step.** In EGES, each block has only two choices (Lemma 2), and a confirmed block must be first finalized (§5.1). Therefore, it is sufficient to prove the following proposition 1: the probability that one node finalizes empty\(_n\) (event A), and another node finalizes \( \text{propos}1_n \) (event B) is overwhelmingly small.

For event A, suppose node \( X \) finalizes empty\(_n\). We use \( f_m \) to denote the maximum finalized block index on node
X. Consider blocks with indices in \([n + 1, f_m]\). Since EGES finalizes empty blocks in descending order (Algorithm 2 Line 12-15), there are no undecided blocks in \([n + 1, f_m]\), and we can have another level of induction by supposing blocks finalized as empty in \((n + 1, f_m]\) are finalized consistently (name it hypothesis 2). For event B, proposa1_n can be finalized either by \(P_n\) (call it event B1) or by subsequent proposers that have learnt this proposal (event B2).

First, we prove that the probability that event A and event B1 happen together is overwhelmingly low. Suppose that a port \(p\) of all \(M\) EGES nodes received and cached the proposa1_n, and we calculate the probability for event B1. We use \(\Re\) to denote the number of acceptors for the \(n^{th}\) block that re Educação proposa1_n. Since proposa1_n is broadcasted in EGES’s P2P network and the stealth acceptors are selected uniformly, \(\Re\) follows hypergeometric distribution \(\Re \sim H(M, n_A, p \times M)\). Thus, the probability that \(P_n\) finalizes proposa1_n is

\[
\text{Prob}(B1) = \text{Prob}(\Re > \tau \times n_A)
\]

We then calculate the probability of event A. Event A infers that after the \(n^{th}\) block, there are at least \(D\) non-empty blocks that are finalized and carrying \(n\) in the undecided list. This means each proposer of these \(D\) blocks received \((\tau \times n_A)\) ACKs from their acceptor group, and none of these acceptors sending those ACKs has received proposa1_n. For each of the \(D\) blocks, the number of acceptors \(\Re\) not receiving proposa1_n follows hypergeometric distribution \(\Re \sim H(M, n_A, (1 - p) \times M)\). Thus, the probability that \(P_n\) finalizes proposa1_n is

\[
\text{Prob}(A) = (\text{Prob}(\Re > \tau \times n_A))^D
\]

The calculation shows that the probability of event A and event B1 happening together \(\text{Prob}(A) \times \text{Prob}(B1)\) is overwhelmingly low for any delivery rate \(p\) by setting \(\tau\) and \(n_A\) (S5.3). For instance, our evaluation chose \(\tau\) as 59%, \(D\) as 4, \(n_A\) as 300, \(M\) as 10K, and the probability of EGES enforcing safety is \(1 - 10^{-9}\). In real deployments, \(M\) may change due to membership changes; however, when \(M\) is much bigger (e.g., 20X) than \(n_A\), this probability is not sensitive to \(M\) because hypergeometric distribution is approximate to binomial.

For the second step, we prove that event A and event B2 cannot happen together. For event B2, we suppose that proposer \(P_i\), where \(i > n\), learns and finalizes proposa1_n. We discuss by comparing \(i\) and \(f_m\) and derive contradictions. If \(i \leq f_m\), hypothesis 2 infers that \(X\) did not finalize empty, so proposa1_n is finalized together with proposa1_n at node \(X\), contradicting to event A. Else if \(i > f_m\), since a proposer can only finalize the maximum index in its local \(U\) list (Algorithm 1 Line 9), for node \(P_i\) we can predicate that blocks with index in \([n + 1, i)\) are finalized. Due to hypothesis \(\Re\) blocks within \([n + 1, f_m]\) are finalized the same as node \(X\), and therefore \(P_i\) should also finalize the \(n^{th}\) block as empty, causing contradiction.

Putting the two steps together, we proved proposition 1, and thus proved the induction step that the \(n^{th}\) block must be confirmed consistently among nodes with overwhelmingly high probability. Therefore, EGES ensures safety with overwhelmingly high probability.

### 6.2 Liveness under Targeted DoS Attacks

EGES can defend against DoS attacks and partition attacks targeting committee nodes under the threat model in §3. Since from a single node’s point of view, when it cannot reach a remote node, it cannot distinguish whether a remote node is under DoS attack or is partitioned, EGES handles these two attacks altogether.

EGES has two types of committee nodes for each block, a proposer and \(n_A\) acceptors, randomly selected from all nodes in the system. Because of EGES’s stealth committee abstraction (S5.2), the identity of each committee node is unknown to attackers outside SGX enclaves before the node sending out its first protocol message.

We first discuss acceptors. An acceptor sends ACK messages to both the proposer and arbiters, but its identity becomes explicit after sending out its first ACK message, so EGES uses fake acceptors to conceal the real ones. If observed from outside enclaves, the fake acceptors behave identically as real ones so that an attacker cannot differentiate the real acceptors and attack them. EGES achieves this with three design points.

First, fake acceptors are randomly selected from all nodes for each block so that an attacker cannot determine the fake acceptors by monitoring network packets (S5.2). Second, real and fake acceptors’ EGES enclaves respond to protocol messages (proposta and arbit) in the same way if observed outside their enclaves (S5.3), so that an attacker cannot distinguish real or fake acceptors by watching their behaviors. Third, all messages from real or fake acceptors are of the same format, encrypted with the receiver’s public key (S5.3) and can only be decrypted in the receiver’s enclave, so that an attacker cannot differentiate real acceptors from fake ones by watching the packet content.

Therefore, if an attacker targeting acceptors in EGES, it can only randomly select \(B\) nodes from both real acceptors and fake acceptors. For instance, if EGES has \(n_A = 300\), \(\tau = 59\%\) and has (expected) 600 fake acceptors; if the attacker’s attack budget is 300, the probability that the attacker can luckily attack more than \((1 - \tau) \times n_A\) real acceptors for one block is 0.3%. Moreover, even if the attacker is so lucky that it successfully guessed more than \((1 - \tau) \times n_A\) acceptors for some block (say \(n_B\)), EGES can still consistently determine whether to confirm proposa1_n or empty_n, using the committees for subsequent blocks (S5.3). In other words, the attacker must constantly be so lucky to make EGES lose liveness.

Then we discuss proposers. The identity of a proposer (say \(P_n\)) becomes explicit after broadcasting its proposa1_n and may be targeted attacked when it is waiting for quorum ACKs. However, since EGES has many (larger than \(B\)) arbiters that can help to finalize the proposa1_n, attacking \(P_n\) will not affect EGES’s liveness. EGES lets each node independently determine whether it is an acceptor for each block (S5.4) so that an attacker cannot keep attacking them. Note that one single proposer or arbiter is enough for EGES to stay alive because any of them can finalize the current \(n^{th}\) block.

Note that EGES’s targeted attack model (S5) handles only DoS or partition attacks targeting specific EGES nodes. A more powerful attacker may also target EGES’s major communication links. From the protocol aspect, EGES avoids such vulnerabilities in the Network layer (in the OSI model [63]) by using distinct committees for different blocks: EGES’s protocol traffic is spread among the whole P2P network rather
We selected the Golang implementation of Ethereum (i.e., geth) as our codebase because geth is heavily tested on the Internet. We leveraged the P2P libraries from geth and rewrote the functions for generating, verifying, and handling new blocks. Since SGX only provides SDKs in C/C++, we used CGo to invoke ECalls. We modified 2073 lines of Golang code and implemented the consensus protocol for 1943 lines of C code. For asymmetric key based encryption, we used ECC-256 from the API provided by the SGX SDK.

Each EGES node has three modules: a consensus module running the EGES consensus protocol and storing nodes’ member list [5], a P2P module connecting to a random set of peers and relaying messages using the Gossip [60] protocol, and a blockchain core module storing confirmed parts of chain and transactions submitted by clients.

As stated in [5], only the consensus module runs in the node’s SGX enclave. We put the P2P module outside SGX because this module is only responsible for relaying messages and putting it inside SGX cannot effectively bring more benefits: a malicious node can still selectively drop packets at the network layer. We put the blockchain core module outside enclave because immutable properties of the hash-connected blockchain make this module’s actions easily verifiable, and putting this module in enclave will consume much memory.

In EGES, a node may finalize a block after knowing its preceding blocks. Therefore, when an EGES proposer proposes a block or a node finalizes a block, it leaves the block’s field of “hash value for the previous block” empty, and EGES’s enclave computes this field when confirming this block. In other words, EGES actually achieves consensus on a totally ordered sequence of transactions, same as Hyperledger Fabric [10], and encapsulate these baches into a hash-chain of blocks while confirming them.

### 7.1 Membership and attestation

To support dynamic memberships, EGES leverages the idea from SCIFER [59] to record the joining of new nodes as transactions on the blockchain. This mechanism ensures Lemma 1 because all updates to the member list are only determined by confirmed blocks.

When a node \( i \) wants to join the system, it needs to find a member node \( j \) to do attention [52] through an out-of-band peer discovery service. We assume that node \( i \) knows the genesis (0th) block, so node \( i \) can inductively verify the blockchain, without relying on whether peer \( j \) is malicious.

A node \( i \) joins EGES with three steps. First, \( i \) launches its EGES enclave, which generates its account \((pk_i, sk_i)\) and creates the hardware monotonic counter \( c \) for defending forging attacks [72]. The node’s account is securely saved to permanent storage using SGX’s seal mechanism [22] for recoveries from machine failures (e.g., power off). Then, \( i \) sends a join request to \( j \). Second, \( j \) does a standard SGX

### 6.3 Parameter Selection

Figure 6 shows the relation between \( \tau \) and \( D \) in order to ensure safety. With a smaller \( \tau \), a proposer \( P_n \) can finalize proposals \( P_n \) after collecting fewer ACKs from acceptors, so subsequent proposers need more rounds of checking (larger \( D \)) when trying to finalize the empty \( y_n \).

The \( \tau \) and \( D \) values also affect EGES’s ability to achieve liveness (confirm non-empty blocks) on network partitions (or ubiquitous DoS attacks). We quantify this ability to the “minimum largest connected component size” (cc%) required in the P2P graph, provided that nodes in a connected component can reach each other before a timeout. A smaller cc means that EGES is more robust to partition and ubiquitous DoS attacks. From a mathematical aspect, as long as the probability of finalizing a proposal is non-zero, the probability \( p_D \) that \( D \) consecutive proposals are successfully finalized is always larger than zero, inferring that eventually EGES can achieve liveness. However, we conservatively calculate the required \( cc \) to make the \( p_D \) larger than 5% for practical liveness, as shown in Figure 7. In our evaluation, we chose \( \tau \) as 59%, \( D \) as 4, \( n_A \) as 300, which ensures both safety and achieves good liveness on partition attacks.

Figure 8 shows the parameter selections if EGES does not use its stealth committee mechanism, but lets each node independently determine whether it is an acceptor with the probability of \( M/n_A \), with \( n_A \) being the expected number of acceptors for each block. If EGES makes such a design choice, the relation becomes a binomial distribution with the probability of \( p \times M/n_A \), and other distribution changes similarly. As shown in Figure 8, EGES would need a larger quorum size \( \tau \times n_A \), and worse liveness on network partition.

### 7 Implementation

We selected the Golang implementation of Ethereum (i.e., geth) as our codebase because geth is heavily tested on
On appending a block.

On confirming a block.

On receiving a propose request.

**Algorithm 1 [Line 4]**. If it is an arbiter for the $n^{th}$ block, it will also start working as an arbiter (5.4).

7.3 Handling Attacks on Enclaves

**Forking attacks.** In the P2P scenario, one challenge is enclave forking attacks [65]. EGES must permit a node to reuse its sealed account (§7.1) in case the node restarts its machine. However, a malicious node can create multiple copies of EGES enclaves with the same account $pk_i$, directs different messages to them, and lets them generate conflicting messages (e.g., block proposals). Existing defending techniques [65], [66] work in the client-server manner, where clients attest and communicate to only a single server. These techniques are not suitable for P2P settings because they will need every two EGES nodes to connect and attest each other.

EGES defends such attacks using SGX’s platform counter [22], which is monotonic among all enclaves on the same machine. When a node launches its EGES enclave, the enclave increments and read this counter value $c$ and enclose this $c$ to its registration transaction: the node’s membership is bound to the enclave with counter value $c$ but not the account $pk$. When the enclave sees a registration with the same account but a higher counter value, it quits automatically.

**Premature timeout.** EGES uses the timeout mechanism to suspect node failures and to achieve liveness. A premature timeout does not affect EGES’s safety because EGES’s safety argument does not rely on timing constraints (§6.1).

However, a premature timeout may affect EGES’s performance, and we implemented a simple mechanism to mitigate this problem by leveraging the trusted timer API `sgx_get_trusted_time` provided by the SGX platform. This API provides a verifiable timer with the granularity of seconds. To avoid the heavy time cost (tens of milliseconds) caused by frequently checking this timer, EGES maintains an untrusted timer outside. When the untrusted timer is triggered, the unused code invokes the timeout logic in EGES’s consensus module in the enclave, and the consensus module checks the trusted timer before invoking EGES’s consensus logic.

8 Evaluation

**Evaluation setup.** Our evaluation was done on both our own cluster with 30 machines and the AWS cloud, with parameters shown in Table 3. In our cluster, each machine has 40Gbps NIC, 2.60GHz Intel E3-1280 V6 CPU with SGX, 64GB memory, and 1TB SSD. On AWS, we started up to 100 c5.18xlarge instances (VMs) running in the same city, each of which has 72 cores, 128GB memory, and up to 25 Gbps NIC. We ran up to 100 EGES nodes on each VM (10k nodes in total), with each EGES node running in a docker container.

To evaluate EGES and baseline protocols in a typical georeplicated setting, while running EGES on both our cluster and AWS, we emulated the world-scale Internet by using the Linux traffic control (TC) command to limit the RTT between every two nodes to a random value between 150ms and 300ms. These settings are comparable to Algorand’s setting on AWS. As AWS does not provide SGX hardware, we ran EGES in the SGX simulation mode on AWS and in the SGX hardware mode on our cluster; we show that EGES’s performance in simulation mode is roughly the same as...
We evaluated EGES with nine consensus protocols for blockchain systems, including five state-of-the-art efficient BFT protocols for permissioned blockchains (BFT-SMaRt [28], SBFT [4], HoneyBadger [29], and HotStuff [5]), two SGX-powered consensus protocols for permissioned blockchains (Intel-PoET [30] and MinBFT [31]), the default consensus protocol in our codebase (Ethereum-PoW [27]), and two permissionless blockchains’ protocol that runs on dynamic committees (Algorand [11] and Tendermint [17]). A detailed description of these protocols is in §8.2.

Since Algorand’s open-source code is still under development, and we were unable to deploy its latest release [67] to the same scale as EGES and Algorand’s own paper (i.e., 10k nodes), we took Algorand’s performance from Figure 5 in its paper [11]. To make the comparison fair, we make EGES’s network setting more rigorous than Algorand’s: Algorand divided nodes into multiple cities where intra-city packets have negligible latency, while EGES lets the RTT among any two nodes be at least 150ms.

For all evaluated protocols, we measured their performance when each of them reached peak throughput. For an apple-to-apple comparison of latency, we adopted Algorand’s method to measure transactions’ server-side confirmation time: from the time a transaction is first proposed by a committee node to the time the transaction is confirmed at this node, excluding the time for clients’ transaction submissions. We measured the server-side instead of the client-side latency because this method precludes the disturbance of client behaviors as these protocols run on different blockchain frameworks. For instance, in Ethereum, PoET (running on Hyperledger Sawtooth [30]), and EGES, a client submits a transaction to a random node, and the transaction is disseminated via P2P networks; in BFT-SMaRt, a client submits transactions to a fixed node (i.e., the leader); in Algorand, a consensus node directly packs a block with a fixed amount of data (e.g., 1MB) instead of using separate transactions.

We set EGES’s transaction size to 250 bytes, a typical transaction size for general data-sharing applications [65]. Since Algorand reported throughput on block size, we convert it to txn/s by assuming the same size of transactions as EGES’s. The transaction sizes for the other eight baseline protocols are either equal to or smaller than that of EGES. Our evaluation focuses on these questions:

- §8.1: Is EGES efficient and scalable?
- §8.2: What is EGES’s performance under targeted DoS attacks?
- §8.3: How sensitive is EGES to its parameters?

Our evaluation focuses on these questions:

- §8.1: Is EGES efficient and scalable?
- §8.2: What is EGES’s performance under targeted DoS attacks?
- §8.3: How sensitive is EGES to its parameters?
proposer node entered its SGX enclaves on finalizing a block. Since each ECall only takes around 3us [22], and EGES’s proposer only did 97 ECalls on average for each block, running in SGX hardware mode and simulation mode makes little difference for EGES’s performance.

<table>
<thead>
<tr>
<th>blk size</th>
<th>txns/blk</th>
<th># ECalls</th>
<th>CPU usage</th>
<th>network usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>750 KB</td>
<td>3000</td>
<td>97</td>
<td>12.4%</td>
<td>15.53 Mbps</td>
</tr>
</tbody>
</table>

TABLE 4: Proposer’s micro-events for finalizing a block.

Scalability. To evaluate EGES’s scalability, we ran 100-1000 nodes on AWS and evaluated its confirm latency with the same block size as in the cluster evaluation. Figure 10 shows the result. The latency is divided into two parts. The figure shows that the seeking for quorum ACKs phase of EGES is the dominant factor because it broadcasts the proposed 750KB block on the P2P network. Fortunately, a P2P broadcast latency is proportional to approximately the log of the number of nodes [70], indicating EGES’s reasonable scalability. The increase rate was slightly greater than the log scale because 100 nodes were run in one VM with CPU and NIC contentions. While running on AWS, EGES’s latency was slightly faster than on our cluster, because AWS CPUs are faster.

Compared to Algorand’s performance in Table 1, EGES showed 2.3X higher throughput and 16.8X faster latency than Algorand. This is due to two reasons. First, Algorand’s VRF-based method selects multiple proposers for each block, and Algorand uses a reduction step to select one proposal by these proposers. Moreover, as the VRF-based approach cannot control the exact number of proposers, nodes must wait for a conservatively long time in the reduction step [84]. In contrast, EGES’s stealth committee abstraction selects one proposer for each block, without the need for such a reduction step. Second, EGES’s consensus protocol has only two rounds in gracious runs to confirm a block [85].

8.2 Performance on DoS Attacks

To evaluate EGES’s robustness under DoS attacks, we ran EGES with 1000 nodes on AWS with \( n_A = 100 \), and conducted targeted DoS attacks that are compliant with our attack model [85]: we assumed the attacker’s budget \( B = 10\% \) of total nodes, and we set the expected count for fake acceptors and arbiters to be 200 and 50 correspondingly. Each time we targeted the current proposer and 99 arbiters or real/fake acceptors (because we cannot distinguish real acceptors). For each attack, we blocked all communication from the attacked nodes for 20 seconds.

We deem such attacks to be powerful enough, as no existing protocols for permissioned blockchain can maintain liveness under such powerful attacks. As shown in [8.3] existing consensus protocols, which ran on static committee nodes, lost liveness until the DoS attack ended. In contrast, each time after we attacked 100 nodes, EGES’s throughput had a temporary drop and recovered before the DoS attack ended, which shows that EGES can ensure practical liveness under such powerful attacks.

After the first attack, the line started to go up after 11.3s, much slower than the other attacks (about 3.1s). We inspected the execution log and found that the slow recovery was because the proposer for the next block happened to be attacked, and EGES waited until \( D \) more blocks to confirm that block as empty. After the second attack, the line took about 7.2s to go up. This is because most real acceptors happened to be attacked together with the proposer, which makes the arbiters failed to finalize the block for the proposer [84]. For the other three attacks, the arbiters successfully helped corresponding proposers to finalize their blocks quickly.

To evaluate EGES performance on network partitions, we manually divided the network into two partitions at 200s and reconnected them at 400s, with one partition containing 80% nodes and the other containing 20% nodes. Figure 11b shows the throughput measured in the large partition. Overall, the large partition maintained liveness during the partition. The small partition did not succeed in confirming any block during the partition and caught up after the network reconnected, preserving safety. There are two obvious throughput drops in the figure, which are caused by the pre-designated proposers being in the small partition, and EGES confirmed empty blocks for them. Note that EGES may temporarily lose liveness in catastrophic partitions (e.g., 50-50 or 40-30-30 partitions) but can preserve safety. §8.3 shows a quantitative analysis of how EGES can preserve liveness under network partitions.

8.3 Sensitivity

EGES’s throughput and confirm latency depend on three important protocol parameters, the number of acceptors, the number of fake acceptors, and block size. Figure 13 shows the sensitivity results. EGES’s performance turns out to be insensitive to the first two parameters because the latency is dominated by the time for broadcasting the new block on the P2P network.

Figure 12 shows EGES’s performance sensitivity on block size. When the block size was larger, EGES’s throughput did increase, but its block confirm latency also increased. In our evaluation, we set EGES’s block size to be 750KB, which is a near-optimal setting for both throughput and latency.

8.4 Comparison to BFT-SMaRt and SBFT

Since BFT-SMaRt with 10 (committee) nodes was faster than EGES with 100 acceptors, we evaluated both of them on a different number of nodes because more such nodes can tolerate more faults and DoS attacks. Figure 14a shows the results using the same setting for both systems (e.g., in our cluster, TC disabled, and the same number of transactions in each batch). Overall, EGES throughput was stable because...
the number of acceptors affects little on the latency in the seeking for quorum ACKs phase. BFT-SMaRt's throughput drops dramatically because its protocol involves a quadratic number of messages on the number of ordering nodes.

Figure 13 also shows SBFT's performance. In the non-geo-replicated mode, when the number of nodes increased from 4 to 62, SBFT's throughput dropped from 38.2K to 6.9K transactions/s. This is because SBFT's collectors need to collect more messages and to verify their signatures, so the time spent in collectors increased from 2.5ms to 13.1ms.

Figure 14 shows the performance comparison of GES, BFT-SMaRt, and SBFT in the geo-replicated setting. GES's throughput was at least 3.4X larger than both systems on 62 nodes. BFT-SMaRt's performance's trend was similar to the no-delay setting because of its PBFT all-to-all broadcasted messages. SBFT's throughput also dropped dramatically because some nodes became stragglers for the collectors due to the varied RTT. Since SBFT's fast path can only tolerate a small number of straggler nodes (usually two [2,2]), we observed that 87% of the consensus rounds in SBFT have reverted to the slow path (PBFT).

More importantly, GES can safely switch its acceptor groups across blocks, and it tolerated various failure scenarios, including DoS attacks (Figure 11a). For comparison, we evaluated the performance of BFT-SMaRt and SBFT on node failures (i.e., DoS attacks targeting consensus nodes). Figure 14 shows the result of BFT-SMaRt with its default 10-node setting. We randomly killed one node on each vertical line. The third time we killed its leader coincidentally, so there was a noticeable performance drop. BFT-SMaRt's throughput dropped to zero after we killed the fourth node. For SBFT (Figure 14d), we started with 62 nodes and killed 7 nodes every time. Since SBFT's fast path can only tolerate two crashed or straggler nodes, its throughput dropped significantly (reverted to PBFT) after the first kill.

Overall, GES is complementary to BFT-SMaRt and SBFT: BFT-SMaRt is the fastest in a small scale; SBFT has better scalability, but its high performance requires a synchronous network (stated in their paper). GES achieved reasonable efficiency and DoS resiliency in a geo-replicated setting.

8.5 Discussions
GES has two limitations. First, GES requires a joining node to have an SGX device. We deem this requirement reasonable because SGX is available on commodity hardware, and both academia and industry are actively improving the security of SGX. Recent public blockchains [21] and permissioned blockchains [39], [20], [31] also use SGX. Second, GES targets large-scale, geo-distributed permissioned blockchain systems (e.g., a global payment system [3]), while for small-scale deployments (e.g., supply chain among a few small companies) in a single data center, existing consensus protocols (e.g., BFT-SMaRt) are more suitable.

GES can incite the deployment of more Internet-scale applications that highly desire the DoS resistance feature of GES, including e-voting, decentralized auction, and payment systems. Moreover, the attested SGX enclaves on GES nodes bring the potential to port existing centralized SGX-powered applications on to GES, including SGX-protected distributed database [50], [57], [71] and SGX-powered anonymous networks [22]. For instance, one can deploy an GES-ToR application by letting the GES enclave on each node do a local attestation for an SGX-ToR enclave [22] and letting GES nodes maintain the ToR directory service on the blockchain. By doing so, GES-ToR removes SGX-ToR's centralized directory service that is vulnerable to DoS attacks or service censoring.

9 Conclusion
We have presented GES, the first efficient permissioned blockchain consensus protocol that can tolerate targeted DoS and partition attack. GES achieves comparable performance to existing permissioned blockchain's consensus protocols while achieving much stronger robustness. GES is carefully implemented with two promising distributed applications, greatly improving the reliability and security of their legacy, centralized versions. GES's source code is available on [github.com/hku-systems/eges](https://github.com/hku-systems/eges)


Xusheng Chen received his Bachelor degree in HKU. He is currently a PhD student in Computer Science of HKU (2017-now). He is under the supervision of Dr. Heming Cui. His research interests include distributed consensus protocols, distributed systems and system security.

Shixiong Zhao received his Bachelor degree in HKU and his master degree in HKUST. He is currently a PhD student in Computer Science of HKU (2017-now). He is under the supervision of Dr. Heming Cui. His research interests include distributed systems for high performance computing, distributed systems and system security.
Jianyu Jiang is currently a third year PhD student in Computer Science Department at The University of Hong Kong. He is working on topics in large scale computation platform under the supervision of Dr. Heming Cui. Jianyu received his Bachelor's Degree in Computer Science Department at Xi’an Jiaotong University, under the supervision of Professor Qi Yong.

Haoze Song received the BS degree from Department of Computer Science, University of Science and Technology of China, in 2020. He is currently working towards the MPhil degree in Computer Science at HKU. His research interests mainly focus on distributed system and distributed computing.

Cheng Wang received his PhD from the University of Hong Kong and B.Eng degree from Shanghai Tongji University. His research interests lie in distributed systems, with a particular focus on fault tolerance. He is currently working in Huawei Ltd.

Tsz On Li received his Bachelor degree in HKU. He is currently an MPhil student in Computer Science of HKU (2019-now). He is under the supervision of Dr. Heming Cui. His research interests include differential privacy and big data systems.

T-H. Hubert Chan T-H. Hubert Chan is an Associate Professor at the Department of Computer Science at the University of Hong Kong. He completed his PhD in Computer Science at Carnegie Mellon University in 2007. His main research interests are approximation algorithms, discrete metric space, privacy and security inspired problems.

Fengwei Zhang is an Associate Professor at Department of Computer Science and Engineering at Southern University of Science and Technology (SUSTech). His primary research interests are in the areas of systems security, with a focus on trustworthy execution, hardware-assisted security, debugging transparency, transportation security, and plausible deniability encryption. Before joining SUSTech, he spent four wonderful years as an Assistant Professor at Department of Computer Science at Wayne State University.

Xiapu Luo received his B.S. in Communication Engineering and M.S. in Communications and Information Systems from Wuhan University. He obtained his Ph.D. degree in Computer Science from the Hong Kong Polytechnic University, under the supervision of Prof. Rocky K.C. Chang. After that, he spent two years at the Georgia Institute of Technology as a post-doctoral research fellow advised by Prof. Wenke Lee. His current research interests include Network and System Security, Information Privacy, Internet Measurement, Cloud Computing, and Mobile Networks.

Sen Wang received the B.S. degree from the University of Science and Technology of China (USTC) in 2005, the M.S. degree from the Chinese Academy of Sciences (CAS) in 2008, and the Ph.D. degree from Tsinghua University in 2014, all in computer science. From 2014 to 2019, he was a lecturer and then an associate professor at Chongqing University, China. Currently, he is a senior researcher at Huawei, Hongkong. His research interests include information-centric networking, Federated Learning and AI for System.

Gong Zhang is a chief architect researcher scientist, director of the Huawei Future Network Theory Lab. His major research directions are network architecture and large-scale distributed systems. He has abundant experience on system architect in networks, distributed system and communication system for more than 20 years. He has more than 90 global patents.

Heming Cui is an associate professor in computer science of HKU. His research interests include operating systems, programming languages, distributed systems, and cloud computing, with a particular focus on building software infrastructures and tools to improve reliability and security of real-world software. Homepage: https://i.cs.hku.hk/heming/. He is a member of IEEE.