GECC: Scalable, Efficient, and Consistent Consensus for Blockchains

Xusheng Chen, Shixiong Zhao
2018.05.31

Abstract
Blockchain technology has the potential to deploy broad decentralized applications, greatly improving their security and reliability. Unfortunately, despite much effort, no existing public blockchain protocol can ensure strong consistency with a high efficiency, while reliability of permissioned blockchain relies on consortium members.

We present GECC, an initial blockchain protocol and its runtime system by leveraging the Intel Software Guard eXtensions (SGX) hardware. GECC achieves privacy and security for permissioned blockchains and leveraging Intel SGX. GECC can append a block with only one and a half network round-trips and two P2P broadcasts. We carry a proof sketch to show that GECC is strongly consistent.

1 Introduction
The emergence of blockchains makes it promising to deploy diverse decentralized applications (e.g., cryptocurrencies and storage services), greatly improving their security and reliability. A blockchain runs as a P2P network consisting of participating computing devices (nodes), and its correctness requires two crucial elements. First, it must tackle Sybil attacks, where an attacker can control the blockchain by spawning an arbitrary number of synonyms. Second, it needs a distributed consensus protocol to let nodes confirm one totally ordered chain of blocks, each containing a number of transactions.

To enable the deployments of general applications, an ideal blockchain should be highly efficient: the throughput and energy consumption of processing transactions should be comparable to those of traditional centralized services. Moreover, this blockchain should be strongly consistent: appended blocks are always confirmed.

Unfortunately, despite much effort, no existing blockchain consensus protocol can efficiently ensure strong consistency. Existing public blockchain consensus protocols belong to two main categories. First, Proof of Work (PoW) protocols [17, 51, 62] let nodes concurrently solve hash puzzles using huge computing power and compete for the longest chain. In PoW, only the longest chain is confirmed, and all the other nodes’ computation is discarded. This computation eliminates Sybil attacks with the cost of consuming excessive power but makes PoW suffer from poor efficiency. For instance, Bitcoin [51] consumes roughly the same electric power as Singapore but only has an average throughput of 7 transaction/s for worldwide users. Worse, previous work shows that PoW’s nodes will confirm inconsistent (forked) chains when Internet incurs temporary partitions.

To improve efficiency, the Proof of Stake (PoS) protocols [14, 23, 29, 33, 46] give blockchain nodes that possess more coins higher probability to append blocks, then a single node can be selected to append a block under this probability distribution with little power consumption. However, the selected node can maliciously append two conflicting blocks, leading to double-spend attacks and consistency violations. Algorand [29] mitigates this problem by presenting a new Byzantine agreement protocol, but it assumes the nodes that possess 80% of coins are honest (i.e., nodes follow the protocol).

These protocols need each node to prove its bets physically. Trusted Execution Environment (TEE) [31, 44] on commodity hardware (e.g., Intel SGX) makes it promising to efficiently build the trust base. If some code executes in SGX on one node, SGX can guarantee the integrity of the code and prove the integrity of the execution to another node. This can be used by a trustworthy consensus protocol to efficiently select nodes to append blocks. Recent blockchain consensus protocols use SGX in different aspects, including proving the identity of nodes (Scifer [6]), replacing PoW’s useless puzzles with useful computation (REM [64]), and replacing PoW’s puzzle answers on blocks with SGX generated random numbers (Proof-of-Luck [49]) so that the chain with the largest sum wins.

However, even with SGX, building an efficient and
strongly consistent blockchain protocol on the asynchronous Internet remains an open challenge. In a blockchain, it is fundamentally difficult for a node to distinguish whether remote nodes go offline or they are partitioned. Recent work shows that attackers can perform temporary partition attacks to make a blockchain fork. For instance, Proof-of-Luck is prone to partition attacks because the two partitioned group of nodes will confirm different chains with each partition’s largest sum.

We present GEEC, an strongly consistent blockchain consensus protocol, where an appended block is always confirmed. We implemented GEEC on the Ethereum [17, 62] blockchain platform. We evaluated GEEC with three blockchain systems: Ethereum [17, 62], EOS [27], and Intel-PoET [58]. We ran GEEC on both our cluster and the Tencent public cloud with popular blockchain workloads. Evaluation shows that:

- **GEEC** is efficient and scalable. Its throughput is comparable to Visa’s, 1.7X~88.5X higher than the evaluated blockchain protocols. Its throughput is scalable to 10K nodes on the Tencent cloud.
- **GEEC** is robust. It eliminates forks and maintains reasonable throughput against node offline, packet loss, and temporary partitions.

Our major contribution is GEEC, the first efficient and strongly consistent consensus protocol for blockchains. GEEC’s runtime system has the potential to deploy general applications, greatly improving their security and reliability.

The remaining of the paper is organized as follows. §2 introduces blockchain systems and TEEs. §3 gives a brief overview of GEEC. §4 introduces how GEEC bootstraps and maintains the member list, §5 introduces GEEC’s block producing protocol. §6 gives implementation details. §7 shows our evaluation, §8 introduces related work and §9 concludes.

## 2 Background

### 2.1 Blockchain and Its Consensus Protocols

Blockchain is a decentralized, highly available, and indestructible ledger that allows everyone to update it and verify its correctness. Blockchains are divided into private and public blockchains. Private blockchains (e.g., Hyperledger Fabric [11, 18, 60], Hyperledger Sawtooth [58], and RSCoin) know the identities of all participating nodes and runs a consensus algorithm (e.g., Raft [55] or BFT [19, 60]) to achieve consensus. Private blockchains achieve efficiency but they are not designed for nodes running in the Internet scale. This paper targets at public blockchains, which admit anyone to join the network and run a node.

Public blockchain are being developed to support more applications than just cryptocurrencies. For instance, Ethereum [17, 62] provides a smart contract mechanism where users can run Turing-complete deterministic applications (e.g., lottery systems). Moreover, digital voting, digital identity verification, and decentralized storage trading are being developed on public blockchains [15]. These promising applications often desire efficiency and strong consistency.

### 2.2 Intel SGX

Trusted Execution Environment (TEE) is used to build secure systems to defend privileged attacks (e.g., OS root users). Intel Software Guard eXtension (SGX) [31, 44, 45] is the most popular TEE product in commodity CPUs. SGX provides a secure execution environment called hardware enclaves, and the code can enter an enclave using ECalls. Memory (data and code) and cpu states in enclaves can not be tampered with or read by any code outside enclaves.

SGX provides two kinds of attestations [10, 22, 32] (local and remote) to prove that the particular piece of code is running in a genuine SGX-enabled CPU. In a local attestation, an enclave directly attests another enclave on the same machine using CPU instructions. In a remote attestation, SGX produces a report of measurements of the enclave (e.g., code, cpu generations) and signs it before returning it to a challenger. The challenger then connects to a Intel’s Attestation Service (IAS) and get a QUOTE to confirm that the code is running in a genuine Intel CPU. SGX also provides a linkable attestation mode for challengers to identify enclaves from the same machines. During attestations, the challengers can establish a secure communication channel with the help of key-exchange protocols (e.g., Diffie-Hellman Key Exchange [16]). All SGX-based blockchain systems [11, 16, 49, 58, 64] and GEEC uses attestations to build trust base among blockchain nodes.

SGX provides a trustworthy source of random number via its `sgx_read_rand` API [31] which calls the hardware based pseudorandom generator(PRNG) through RDRAND on Intel CPUs [22]. Previous studies show that this random number generator is safe and cannot be altered from outside the enclave [12, 30, 50].
3 Overview

3.1 GEEC’s Threat Model

GEEC allows any node with SGX to join its P2P network via its registration protocol (§4.2). GEEC has two design goals. First, strong consistency. With overwhelming probability, GEEC guarantees that no two nodes will see different sequences of blocks (i.e., no forks). Second, egalitarian. On expectation, each registered node should append the same number of blocks to the blockchain.

For SGX, GEEC has the same threat model as typical SGX-based systems [6,49,58,64]. We trust the hardware and software of the SGX and its remote attestation services. The code and data inside SGX are trusted. Besides, the random numbers generator in SGX is trusted (§2.2). Side-channel and access pattern attacks on SGX are out of the scope of this paper, but GEEC can handle DoS attacks. GEEC also makes standard assumptions on cryptographic primitives.

Unlike existing committee-based blockchain systems (e.g., Algorand [29] and Scifer [6]) which assume the honesty of most nodes, GEEC preserves strong consistency without this assumption. In GEEC, a node’s code running outside SGX is not trusted and can behave arbitrarily. To ensure reasonable liveness, same as Algorand, GEEC needs a vast majority (e.g., 70%) of its nodes to be online. To achieve this, GEEC provides an incentive mechanism (§4.2).

3.2 Architecture

GEEC’s block is committed by committees and each committee confirms one block. For each committee, C nodes are randomly selected as the committee members, and one committee node is elected as a proposer to append each block. GEEC has three modules running in the SGX on each node:

- **Registration module** (§4.2) handles the joining request of new nodes. When a remote node wants to join, the module first attests genuineness of the remote node, it then generates a signed registration transaction and broadcasts the transaction. Once the transaction is included in GEEC’s blockchain, the node joins successfully.

- **Consensus module** (§5) runs GEEC’s consensus protocol on all registered nodes. This protocol ensures at most one node is selected as the proposer to append each block. The consensus module of the proposer generates a signed proof to be included in the block by the proposer’s blockchain core, so that other nodes can validate the block.

4 Bootstrapping GEEC

4.1 Initialization of the Blockchain

To achieve decentralization, all configurations and setup of blockchain should be determined and delivered with the genesis (0th) block. In GEEC, the genesis block contains two key components, the first is normal blockchain configuration (GEEC parameters) and the second is the measurement of GEEC’s enclave (hash of enclave code $H_e$).

The creator (GEEC’s publisher) of the genesis block first registers the Intel IAS service and gets a credential $cre$, and then chooses one genesis consensus node which supports Intel SGX and installs GEEC’s enclave code. The creator does a remote attestation to the enclave and receives a QUOTE $Q_n$ (§2.2), which confirms the success of the attestation. After the attestation, GEEC’s registration module generates two asymmetric key pairs - the enclave account key pair $(pK_0, sK_0)$ as a per-node identity (account) and a key pair $(pK_{shared}, sK_{shared})$ shared among GEEC’s referee module for secret transactions. Then the creator generates the genesis block with $Block_0 = (pK_{shared}, member(pK_0, Q_0), H_e)$.

4.2 On-chain Node Join

In GEEC, the registration module on each registered node serves as a challenge for attesting a newly joining node $i$ with three steps. First, the new node sets up the enclave code (with hash $H_e$). The new node generates its account $(pK_i, sK_i)$ in local enclave. Then the new node broadcasts a join request to GEEC members. Second, a challenger $j$ who receives the join request starts a remote attestation. After the attestation succeeds with a returned QUOTE $Q_i$ from IAS, the challenger’s registration module transfers the shared key $(pK_{shared}, sK_{shared})$ and the credential $cre$ through the DHKE secure channel to node $i$. Third, the challenger broadcasts a registration transaction (§4.2) $(Sign_{sk}(member(pK_i, Q_i)), pk)$. Node $i$ joins the GEEC member list when the transaction is confirmed. GEEC only allows each CPU with SGX to join as one node by using linkable attestation mode (§2.2). To remove zombie nodes, this membership only persists for 1K blocks, which maintains reasonable online ratio and liveness for GEEC.

5 Block Appending Protocol

In a high level, GEEC’s protocol works in three steps. First (§5.1), from the last confirmed block, a committee and a group of acceptors are derived. The committee is explicit
to all nodes, and it will be reformed if a block cannot be appended with a timeout. Second (§5.2), the committee selects a unique proposer. Third (§5.3), to append next block, a proposer first learns whether there is a potentially confirmed block. If so, the proposer proposes the same block; otherwise, it can propose any block. When proposing a block, it needs to seek majority votes from the acceptors. To encourage nodes to join GEEC and to often stay online, GEEC gives block- appending rewards (transaction fees) to the nodes that participate in the protocol.

In GEEC, SGX guarantees that all nodes follow the protocol logic. Moreover, an attacker cannot generate a malicious protocol message presenting itself as a consensus participant to affect the protocol logic. This is because every GEEC node’s account \((pk)\) has been publicly stored on the blockchain, and only code running in the node’s SGX knows the corresponding \(sk\). Each GEEC protocol message is signed by \(sk\) and carries \(pk\), such that any GEEC node detect a malicious message.

### 5.1 Selecting Committee and Acceptors

With the joined node list stored on the blockchain, a strawman approach is to randomly select only one proposer each time (i.e., committee size is one). However, this approach has bad liveness because if the selected proposer is offline, many timeouts may happen until finding an online proposer. Therefore, GEEC takes a committee based approach because GEEC can efficiently find a proposer if a majority of the committee is online.

For each block, GEEC selects \(C\) committee members from the registered member list stored on the confirmed blocks. The committee formation protocol has two basic requirements. First, it needs to be verifiable so that any node can verify the members of the committee. Second, the selection needs to be unpredictable: a previous committee member cannot control the identities of next committee members, which prevents the system from being controlled by a small group of people.

GEEC’s committee selection protocol meets these two requirements. GEEC uses a random number \(r\) (§5.2) on the last confirmed block as a seed to a uniform sampling function to select committee members, with a committee version number \(C_v = 0\). Therefore, this committee is known to all nodes. If the committee cannot make progress, a new committee is formed after a timeout (15s in GEEC) on each node, and the \(C_v\) for the new committee is incremented by one. However, using the same seed and same member list will select the same group of committee and GEEC’s committee selection protocol will get stuck. Therefore, GEEC uses the hash of previous seed as the new seed to select a new committee. There may be different committees with different \(C_v\), among nodes at the same moment, which GEEC readily considers (§5.3).

Acceptors are selected by the proposer of last confirmed block in SGX. When it generates the block, it randomly selects \(N\) acceptors from the member list, generates \(N\) certificates using each selected acceptor’s \(pK\), and includes the \(N\) resultant certificates in the block. Therefore, a true acceptor can decrypt one certificate using its \(sK\) and confirms its identity.

### 5.2 Proposer Election

GEEC’s proposer election protocol elects one proposer for each block from a committee. A naive approach is to implement an existing election protocol (e.g., Raft [53]) running in SGX. However, there are two problems. First, Raft’s election protocol can have split-vote (no node gets majority votes) and need to retry. When runs on the Internet with 100 nodes, the problem becomes more severe and greatly degrades the performance. Second, this approach is not egalitarian among committee members. The reason is that Raft focuses on achieving fast consensus in a collaboration but not competing environment, where the node that starts the election first will likely become the leader.

To solve these problems, GEEC introduces a new, random number based proposer election protocol. GEEC lets each committee member to generate a trusted random number \(r\) inside SGX. During the election, a node will only vote for another node with a larger \(r\), and a node that receives votes from a majority of the committee members will win the election. We do not select the global largest number because this will require all nodes to be online.

To mitigate split-vote, GEEC’s election protocol introduces a representative mechanism. If a committee member \(A\) votes for \(B\), then \(A\) makes \(B\) be its representative. If \(B\) is able to get a majority of votes, then \(B\) will be elected, otherwise \(B\) can vote for other committee members representing both itself and \(A\). The detailed protocol works as follows with one network round-trip:

When a node receives the \((n - 1)_{th}\) block and finds itself in the committee with version \(C_v\) for the \(n_{th}\) block, the node tries to elect itself as the proposer. It first sends a \((\text{Sign}_{sk}(\text{elect}, r, n, C_v), pK)\) message to all other committee members using UDP, where \(r\) is the trusted random number generated in TEE and \(pK\) is the node’s account (§4.2). It then waits for votes from all other members. If it receives a vote from a remote node, it becomes the representative of the remote node; if it receives a majority of votes, it wins the election.
Algorithm 1: Proposer’s algorithm

```plaintext
function LEARN(C_v, N_acceptor):
  if C_v == 0 then
    blk ← GenerateBlock()
  else
    count ← 0
    BlkList ← []
    Broadcast(Sign_sk(learn, C_v, pk))
    while count < N_acceptor/2 do
      msg ← recv().verify()
    switch msg.type do:
      case EMPTY:
        count + +
      case NOTIFY:
        count + +
      Append(BlkList, msg.Blk)
    if BlkList == [] then:
      blk ← GenerateBlock()
    else
      blk ← MaxVerBlock(BlkList)
  return blk

function PROPOSE(blk, C_v, N_acceptor):
  count ← 0
  Broadcast(Sign_sk(propose, blk, C_v, pk))
  while msg ← recv().verify() do
    if VerifyVote(msg) then
      count + +
    if count > N_acceptor/2 then
      break;
  Broadcast(Sign_sk(confirm, blk.header, pk))
```

Algorithm 2: An acceptor’s algorithm

```plaintext
function ACCEPTORVOTE:
  C_vmax ← 0
  Blk_pending ← null
  while msg ← recv().verify() do
    switch msg.type do:
      case PROPOSE:
        if msg.C_v > C_vmax then
          msg.C_v > C_vmax
          hash ← msg.blk.header
          ci ← Encmsg.pk(vote, hash, pk)
          Reply(Sign_sk(ci), pk)
          Blk_pending ← msg.blk
        case LEARN:
        if Blk_pending == null then
          ci ← Encmsg.pk(notify, B, pk)
          Reply(Sign_sk(ci), pk)
        else
          B ← (Blk_pending, C_vmax)
          ci ← Encmsg.pk(notify, B, pk)
          Reply(Sign_sk(ci), pk)
```

When a node receives a `learn` message, it first checks the signature and whether they have the same \(C_v\). If the node has not generated the random number for block \(n\), it generates it first. If the `learn` message’s \(r\) is larger than the node’s own \(r\), the node votes for who sends the message using UDP with \((Sign_{sk}(vote, n, C_v), pk)\). If it is the representative of other nodes, it also transfer the votes to who sends the message.

5.3 Confirming a Block

We derive GEEC’s proposer and acceptor algorithms from “Paxos Made Simple” due to its proven safety and simplicity. We elect a single proposer from each committee because allowing multiple proposers will make the consensus harder to converge.

Algorithm 1 shows the proposer’s algorithm for confirming a block. It first invokes `Learn` function whether there is a potentially confirmed block. If not (normal case), it can propose any block it wants; if yes, it just proposes potentially confirmed block returned by `Learn`. The function must wait for a majority of acceptors’ responses. If `Learn` successfully returns a block, the proposer calls a `Propose` function, proposes this block, and waits a majority of votes from the acceptors, and then broadcasts a `confirm` message. The proposer keeps retrying these functions until it receives a confirm message and knows it failed.

Algorithm 2 shows the acceptor’s algorithm. An acceptor maintains the highest committee version is has voted for. On receiving a request for votes, if it has not voted for any block with a higher committee version, it sends its vote. On receiving `learn` message with version \(C_v\), if it already has sent a vote for a (pending) block, it replies the block to the proposer using UDP; otherwise, it replies an `empty` message.

5.4 Proof Sketch of Correctness

In this subsection, we provide a sketch of proof on the strong consistency (safety) guarantees of GEEC. We prove GEEC’s safety by induction. Suppose GEEC guarantees safety from the 0th (i.e., genesis) block to the \((n-1)_{th}\) block, and we prove that there is only one unique block \(B\) can be confirmed as the \(n_{th}\) block in the blockchain. The base case is trivial because all nodes start from the same genesis block.

From the induction hypothesis, since block \((n-1)\) is unique, the \(N\) acceptors list included in block \((n-1)\) is unique and unchanged. If there are two different blocks \(B_1\) and \(B_2\) confirmed as the \(n_{th}\) block, we try to prove
contradiction. Since each version of committee only has one proposer, the two blocks must have different version \( C_{v1} \) and \( C_{v2} \). Without losing generality, we assume \( C_{v1} < C_{v2} \). Since \( C_{v1} \) is confirmed, there must be a majority of acceptors voted for it. Then, if the proposer for \( C_{v1} + 1 \) proposed a block, it must learn \( B_1 \) when calling the Learn function in Algorithm 1. This is because this function only returns after it hears from a majority of acceptors and two majorities must overlap, it must hear \( B_1 \) with the max version \( C_{v1} \) from an acceptor (according to Algorithm 2). By recursion, we can find that for any version \( C_v > C_{v1} \), the proposed block can only be \( B_1 \) and thus \( B_2 = B_1 \) and the strong consistency holds.

6 Implementation Details

We implemented GEEC in the Golang implementation of Ethereum [28] (i.e., Geth), which is the official and most stable implementation version of Ethereum. Since Intel only provides SGX SDKs in C/C++ language, so we adopt cgio in Golang to invoke the SGX ECalls in GEEC.

Geth has an interface for implementing new consensus engines, which needs to implement mainly the functions for sealing blocks. GEEC checks whether it is a committee member according to the committee formation protocol (5.1) on receiving new blocks. When a node tries to seal a block, it first invoke an ECall to do the proposer election (5.2). The proposer invokes the two algorithms (5.3) implemented in enclaves via as an SGX ECall. If the election succeeds, the ECall returns a signed proof for the proposer to seal in the block.

We modified 2073 lines of Golang code for Geth, and implemented the election protocol (5.2) and the acceptor mechanism (5.3) for 1043 lines of C code to run in Intel SGX. GEEC’s consensus protocol is general for all blockchain platforms and the two enclaves for election protocol and partition detection protocol work as standalone libraries and can be directly ported.

| Config         | Cluster | Cloud       |
|----------------|---------|-------------|
| # Nodes        | 300     | up to 10K   |
| Committee Size | 30      | 30          |
| Acceptor Size  | 100     | 300         |

Table 1: GEEC’s evaluation parameters.

7 Evaluation

Our evaluation was done on both the Tencent public cloud and our own cluster consisting of 32 machines. In our cluster, each machine has Linux 3.13.0, 40Gbps NIC, 2.60GHz Intel E3-1280 V6 CPU with SGX, 64GB memory, and 1TB SSD. On the public cloud, we started 100 instances (VMs) running in the same city, each of which has 32 cores, 128GB memory, and up to 100 Mbps NIC. We did not choose the EC2 cloud because she was only able to rent each user 3 instances. While running GEEC on both our cluster and public cloud, we used the Linux TC command to limit the network latency between each two GEEC nodes to 200ms (same as Algorand’s evaluation setting). Because this cloud does not provide SGX hardware, we ran GEEC in the SGX simulation mode.

We compared GEEC’s performance with four blockchain systems, including three public blockchains (Ethereum [28], EOS [27], and Snow-white [14]) and one SGX-based private blockchain system, Hyperledger-Sawtooth (Intel-PoET) [57]. These systems cover two PoW systems (Ethereum and Sawtooth) and two PoS systems (EOS, and Snow-white). We ran three of the systems (Ethereum, EOS, and Sawtooth) in our cluster because they were open-source, while Snow-White’s results were from their papers. For Intel PoET, we used its own benchmark tools; for other three systems, we let each node to generate cryptocurrency transfer transactions. Table 1 shows the parameters we used in evaluation.

7.1 Efficiency

Our evaluation shows that GEEC’s throughput is at least 1.7X higher than the evaluated public blockchains because GEEC’s consensus protocol (5) only needs 1.5 network round-trip and two P2P broadcasts to elect a unique proposer. Ethereum is a fast PoW protocol which has a low block mining time compared to other PoW blockchains. However, its throughput is still much lower than GEEC’s. Although EOS whitepaper [27] estimates a throughput of 100K transaction/s, we found its throughput 420 transaction/s in our cluster. The reason is that EOS’s latest open-source implementation was under development and it lacked several crucial features (e.g., parallel chains and parallel signature verification) reported in their whitepaper. After all, EOS relies on pre-determined super nodes [27], PoET achieved a low throughput because it uses PoW. Snow-White [14] presents in their paper that it can achieve up to 150 transaction/s with 40 nodes on Amazon EC2. Overall, we found GEEC the fastest in evaluation.

Recently, Hyperledger-Fabric [11, 18] reported a notable throughput of about 10K transaction/s using a byzantine ordering service. Hyperledger-Fabric is a pri-
vate blockchain which is not designed to scale to nodes from the Internet, while GECC is for public blockchains.

8 Related Work

Public Blockchain System Proof of Work. BitCoin [51] is the first PoW based blockchain system that works as a fully functional cryptocurrency system, but it suffers from high propose for energy consumption and bad performance. Bitcoin-NG and Fruitchain [55] use multiple types of blocks to improve the throughput of Bitcoin. GHOST [59] proposes to select a subtree rather than the longest chain when the blockchain forks to reduce wasted blocks. Ethereum [17, 62] reduces the difficulty of hash computations in PoW and introduces the uncle block mechanism to reduce potential forks. Solidus, ByzCoin [56], PeerCensus [25], and Hybrid Consensus [56] use PoW to select a group of committee and runs a Byzantine fault tolerant (BFT) protocols [19] to achieve consensus on a single block. Elastico [41] and OmniLedger [57] propose a sharding approach inspired by traditional database to linearly scale existing blockchains’ throughput. These works improve the throughput and latency of PoW based blockchain but still consume large amount of energy. Moreover, the BFT protocols assume more than two thirds of the selected committee members are honest. GECC has high performance and does not assume the honesty of the nodes.

Proof of Stake. Algorand [29] selects nodes to form a committee that secures the blockchain system based on the ratio of the number of coins (i.e., stake). It assumes an extremely powerful adversary that can compromise the committee members when they send out packets, but it lets each committee member to determine whether it is a committee member independently, and proposes a customized Byzantine agreement protocol to let each committee member confirm that the messages are not sent twice in each round. GECC differs from Algorand. To preserve strong consistency, Algorand assumes a vast majority (e.g., 80%) of the users to be honest because it works on arbitrary devices, while GECC does not need this assumption by leveraging SGX. PRao [23, 33] and Snow-White [14] are two PoS systems similar to Algorand that each node independently determines whether it is the proposer of next block. PRao uses a secure multi-party computation algorithm; Snowwhite uses a PoW similar hash algorithm and set each node’s hash target according to their stakes. EOS [27] is a PoS system that uses off-chain methods (i.e., selling tokens) to elect 49 super nodes and runs a BFT protocol in a 21 node committee with this 49 super nodes to append new blocks. Unlike GECC, PoS systems often require a vast majority of nodes to be honest.

Private Blockchain. Private blockchain systems require a centralized company (or a consortium of them) to control the joining and consensus of nodes. Hyperledger [11, 18] is a private blockchain project that which are statically configured by consortium policy. GECC is designed for private blockchain. It achieves consensus by running the Byzantine fault tolerant (BFT) protocol in a 21 node committee with this 49 super nodes. RSCoin lets a set of trusted authorities to run a cryptocurrency and divides them into different levels to improve performance. BlockBench [26] is an evaluation framework for comparing different private blockchain systems. In general, private blockchains are efficient but not designed to work in the Internet scale.

Blockchain Applications. Diverse applications have been developed on blockchains. Ethereum [17] introduces EVM, a deterministic runtime, to run smart contract applications on all consensus nodes of a blockchain. Hawk [38] and zkLedger [52] focus on enhancing confidentiality of smart contracts. Ekiden [20] offloads the execution of smart contracts to a small group of SGX powered computing nodes, so that the mining can avoid the repetitive, redundant smart contract executions on all consensus nodes. ShadowEth [63] works similar as Ekiden. To deploy an application in Ekiden or ShadowEth, these applications have to be mostly re-written using the smart contract language [17]. Overall, GECC’s consensus protocol is complementary to these application systems and can be integrated in their underlying consensus layer.

TEE-powered Blockchain Consensus. Recently, TEE has been leveraged to improve diverse aspects of blockchain systems. Intel’s Proof of Elapsed Time [58] efficiently replaces the PoW puzzles with a trusted timer in SGX. GECC is for public blockchain services that any node can join, while PoET is a private blockchain with a known member list. Moreover, our evaluation [71] shows that GECC achieves better performance than PoET. Resource efficient mining [64] uses “useful” computation (e.g., big data computation) to replace the “useless” PoW puzzles and uses SGX to count the amount of the useful computation. Proof of Luck [49] presents a protocol that lets each miner seal a random number generated from SGX into the block as the “luck” of the block. The protocol selects the chain with the largest accumulative luck as the winner. This protocol will have inconsistency (forks) when the network is temporarily partitioned because the partitions will confirm different largest accumulative luck. CoCo [11] is an ongoing private blockchain project that can support diverse consensus protocol (e.g., Raft [53]) in SGX-powered nodes. Currently, CoCo has not described a detailed consensus protocol or evaluation re-
results. SCIFER [6] uses SGX’s remote attestation feature to establish a reliable identity for each user, records the identities on the blockchain and select the oldest active user as the proposer of the each block. SCIFER uses nodes’ “ages” on the blockchain to run a voting algorithm to select a block proposer, so the safety of this protocol has to assume a vast majority of nodes to be honest; GEEC’s safety does not rely on any node’s honesty.

9 Conclusion

We have presented GEEC, the first efficient, strongly consistent and general consensus protocol and its runtime system for public blockchains. The stealth acceptor abstraction takes the first step to enable the strong safety of Paxos to be integrated in GEEC’s consensus protocol. Extensive evaluation shows that GEEC is efficient, scalable, robust and has the potential to support general applications, greatly improving their security and reliability on Internet. GEEC’s source code and evaluation results are released on [github.com/ndsi19-p25/geec](http://github.com/ndsi19-p25/geec).

References

[1] GitHub - Azure/coco-framework. [https://github.com/Azure/coco-framework](https://github.com/Azure/coco-framework).

[2] TPC-C. [http://www.tpc.org/tpcc/](http://www.tpc.org/tpcc/) 2014.

[3] Medical Chain. [http://www.medicalchain.org/](http://www.medicalchain.org/) 2017.

[4] Satoshi Client Node Discovery. [http://www.the-blockchain.com/docs/BlockstackDesignandImplementationofaGlobalNamingSystem.pdf](http://www.the-blockchain.com/docs/BlockstackDesignandImplementationofaGlobalNamingSystem.pdf), 2016. Accessed: 2016-03-29.

[5] Supply Chain. [http://www.supplychain.org/](http://www.supplychain.org/) 2017.

[6] M. Ahmed and K. Kostiainen. Identity aging: Efficient blockchain consensus. arXiv preprint arXiv:1804.07391, 2018.

[7] S. Albrecht, S. Reichert, J. Schmid, J. Strüker, D. Neumann, and G. Fridgen. Dynamics of blockchain implementation-a case study from the energy sector. In Proceedings of the 51st Hawaii International Conference on System Sciences, 2018.

[8] M. Ali, J. Nelson, R. Shea, and M. J. Freedman. Blockstack: Design and implementation of a global naming system with blockchains. [https://aws.amazon.com/ec2/instance-types/](https://aws.amazon.com/ec2/instance-types/).

[9] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative technology for cpu based attestation and sealing. In Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy, volume 13. ACM New York, NY, USA, 2013.

[10] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, page 30. ACM, 2018.

[11] J. Aumasson and L. Merino. Sgx secure enclaves in practice–security and crypto review. Black Hat, 2016.

[12] J. Behl, T. Distler, and R. Kapitza. Hybrids on steroids: Sgx-based high performance bft. In Proceedings of the Twelfth European Conference on Computer Systems, pages 222–237. ACM, 2017.

[13] I. Bentov, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. https://eprint.iacr.org/2016/919.pdf, 2016. Accessed: 2016-11-08.

[14] https://www.quora.com/What-are-non-Bitcoin-applications-of-blockchain-technology.

[15] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably-authenticated group diffie-hellman key exchange. In Proceedings of the 8th ACM conference on Computer and Communications Security, pages 255–264. ACM, 2001.

[16] V. Buterin. Ethereum: A next-generation smart contract and decentralized application platform. [https://github.com/ethereum/wiki/wiki/White-Paper](https://github.com/ethereum/wiki/wiki/White-Paper), 2014. Accessed: 2016-08-22.

[17] C. Cachin. Architecture of the hyperledger blockchain fabric. [https://www.zurich.ibm.com/dccl/papers/cachin_dcc17.pdf](https://www.zurich.ibm.com/dccl/papers/cachin_dcc17.pdf), 2016. Accessed: 2016-08-10.

[18] M. Castro and B. Liskov. Practical byzantine fault tolerance. In Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI ’99), Oct. 1999.
[20] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Eki-den: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution. arXiv preprint arXiv:1804.05141, 2018.

[21] M. Conti, S. K. E, C. Lal, and S. Ruj. A survey on security and privacy issues of bitcoin. arXiv:1706.00916, 2017. Accessed: 2017-06-29.

[22] V. Costan and S. Devadas. Intel sgx explained. IACR Cryptology ePrint Archive, 2016:86, 2016.

[23] B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros: An adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573, 2017. Accessed: 2017-06-29.

[24] P. De Hert and V. Papakonstantinou. The proposed data protection regulation replacing directive 95/46/ec: A sound system for the protection of individuals. Computer Law & Security Review, 28(2):130–142, 2012.

[25] C. Decker, J. Seidel, and R. Wattenhofer. Bitcoin meets strong consistency. In Proceedings of the 17th International Conference on Distributed Computing and Networking, page 13. ACM, 2016.

[26] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan. Blockbench: A framework for analyzing private blockchains. https://arxiv.org/pdf/1703.04057.pdf, 2017. Accessed: 2017-03-22.

[27] EOSIO. Eos.io technical white paper v2. https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md, 2018. Accessed: 2018-05-09.

[28] Ethereum community. Ethereum: A secure decentralised generalised transaction ledger. https://github.com/ethereum/yellowpaper, 2016. Accessed: 2016-03-30.

[29] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017. Accessed: 2017-06-29.

[30] M. Hamburg, P. Kocher, and M. E. Marson. Analysis of intels ivy bridge digital random number generator. Online: http://www.cryptography.com/public/pdf/Intel_TRN_G_Report_20120312. pdf, 2012.

[31] Intel. Software guard extensions programming reference. https://software.intel.com/sites/default/files/329298-001.pdf.

[32] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. McKeen. Intel® software guard extensions: Epid provisioning and attestation services. White Paper, 1:1–10, 2016.

[33] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. https://pdfs.semanticscholar.org/1c14/549f7ba7d6a000d79a7d12255ebf0151.pdf, 2016. Accessed: 2017-02-20.

[34] S. M. Kim, J. Han, J. Ha, T. Kim, and D. Han. Enhancing security and privacy of tor’s ecosystem by using trusted execution environments. In NSDI, pages 145–161, 2017.

[35] S. M. Kim, J. Han, J. Ha, T. Kim, and D. Han. Enhancing security and privacy of tor’s ecosystem by using trusted execution environments. In NSDI, pages 145–161, 2017.

[36] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, Aug. 2016. USENIX Association.

[37] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger. Cryptology ePrint Archive, Report 2017/406, 2017. Accessed: 2017-06-29.

[38] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In Symposium on Security & Privacy. IEEE, 2016.

[39] https://github.com/cstack/db_tutorial/blob/master/db.c.

[40] L. Luu, V. Narayanan, K. Baweja, C. Zheng, S. Gilbert, and P. Saxena. Scp: a computationally-scalable byzantine consensus protocol for blockchains. https://www.weusecoins.com/assets/pdf/library/SCP-A-Computationally-Scalable-Byzantine.pdf, 2015. Accessed: 2016-08-10.

[41] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In Proceedings of the 2016
[42] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun. Rote: Rollback protection for trusted execution. IACR Cryptology ePrint Archive, 2017:48, 2017.

[43] D. Mazieres. Paxos made practical. Technical report, Technical report, 2007. http://www.scs.stanford.edu/dm/home/papers, 2007.

[44] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas. Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave. In Proceedings of the Hardware and Architectural Support for Security and Privacy 2016, page 10. ACM, 2016.

[45] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. HASP @ ISCA, 10, 2013.

[46] S. Micali. Algorand: The efficient and democratic ledger. http://arxiv.org/abs/1607.01341, 2016. Accessed: 2017-02-09.

[47] A. Miller, M. Moeser, T. Lee, and A. Narayanan. Compounding of wealth in proof-of-stake cryptocurrencies. https://arxiv.org/abs/1704.04299, 2017. Accessed: 2017-04-24.

[48] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of bft protocols. https://eprint.iacr.org/2016/199.pdf, 2016. Accessed: 2017-01-10.

[49] M. Milutinovic, W. He, H. Wu, and M. Kanwal. Proof of luck: An efficient blockchain consensus protocol. In SysTEX ‘16 Proceedings of the 1st Workshop on System Software for Trusted Execution, pages 2:1–2:6. ACM, 2016.

[50] M. H. Mofrad, A. Lee, and S. L. Gray. Leveraging intel sgx to create a nondisclosure cryptographic library. arXiv preprint arXiv:1705.04706, 2017.

[51] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf Dec 2008. Accessed: 2015-07-01.

[52] N. Narula, W. Vasquez, and M. Virza. zkledger: Privacy-preserving auditing for distributed ledgers. auditing, 17(34):42.

[53] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In Proceedings of the USENIX Annual Technical Conference (USENIX ’14), June 2014.

[54] https://en.wikipedia.org/wiki/Packet_loss

[55] R. Pass and E. Shi. Fruitchains: A fair blockchain. http://eprint.iacr.org/2016/916.pdf, 2016. Accessed: 2016-11-08.

[56] R. Pass and E. Shi. Hybrid consensus: Scalable permissionless consensus. https://eprint.iacr.org/2016/917.pdf, Sep 2016. Accessed: 2016-10-17.

[57] https://www.hyperledger.org/projects/sawtooth

[58] G. Prisco. Intel develops sawtooth lakedistributed ledger technology for the hyperledger project. Bitcoin Magazine, 2016.

[59] Y. Sompolinsky and A. Zohar. Accelerating bitcoin’s transaction processing. fast money grows on trees, not chains, 2013.

[60] J. Sousa, A. Bessani, and M. Vukolić. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. arXiv:1709.06921, 2017. Accessed:2017-09-25.

[61] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM Comput. Commun. Rev., 31(4):149–160, 2001.

[62] G. Wood. Ethereum: A secure decentralised generalised transaction ledger eip-150 revision (759dccd - 2017-08-07), 2017. Accessed: 2018-01-03.

[63] R. Yuan, Y.-B. Xia, H.-B. Chen, B.-Y. Zang, and J. Xie. Shadoweth: Private smart contract on public blockchain. Journal of Computer Science and Technology, 33(3):542–556, 2018.

[64] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. van Renesse. Rem: Resource-efficient mining for blockchains. http://eprint.iacr.org/2017/175, 2017. Accessed: 2017-03-24.