GEEC: Simple, Efficient, and Consistent Consensus for Public Blockchains

Paper #25

Abstract

A public blockchain 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 (i.e., appended blocks are always confirmed) with a high efficiency comparable to Visa's.

We present GEEC, a new public blockchain protocol and its runtime system by leveraging the strong confidentiality and integrity of the Intel Software Guard eXtentions (SGX) hardware. GEEC 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 GEEC is strongly consistent. Extensive evaluation on both a public cloud and our own cluster shows that: (1) GEEC's block confirm latency was merely few seconds and scalable to the number of users; (2) it was robust on various network anomalies (e.g., partition attacks); and (3) its runtime system supports diverse decentralized applications. GEEC's source code and evaluation results are released on `github.com/ndsi19-p25/geec`.

1 Introduction

The emergence of public 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 [24], 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 public blockchain should be highly efficient: the throughput and energy consumption of processing transactions should be comparable to those of traditional centralized services (e.g., Visa processes about 2K transaction/s [4]). Moreover, this blockchain should be strongly consistent: appended blocks are always confirmed.

Unfortunately, despite much effort, no existing public blockchain consensus protocol can efficiently ensure strong consistency. Existing public blockchain consensus protocols belong to two main categories. First, Proof of Work (PoW) protocols [13, 48, 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 [48] consumes roughly the same electric power as Singapore but only has an average throughput of 7 transaction/s for worldwide users. Worse, previous work [31, 50] shows that PoW's nodes will confirm inconsistent (forked) chains when Internet incurs temporary partitions.

To improve efficiency, the Proof of Stake (PoS) protocols [10, 20, 29, 34, 45] 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 [29, 57] 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).

A key reason these existing protocols rely on either computation power or possessed coins is that there nodes have no trust base among each other. Therefore, these protocols need each node to *prove* its bets physically. Fortunately, the recent prevalence of Trusted Execution Environment (TEE) [32, 43]) 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 public blockchain consensus protocols use SGX in different aspects, including proving the identity of nodes (Scifer [3]), 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 [46]) so that the chain with the largest sum wins.

However, even with SGX, building an efficient and strongly consistent public blockchain protocol on the asynchronous Internet remains an open challenge. In a public 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 [31, 50] 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[1], an efficient and strongly consistent public blockchain consensus protocol, where an appended block is always confirmed. GEEC's key building block is a new consensus abstraction called *stealth acceptors*, which are designed for helping GEEC consistently append blocks and effectively hiding themselves against targeting attacks. The stealthness of an GEEC acceptor is ensured by two rules. First, a GEEC node's SGX code knows whether itself is an acceptor, but this secret is unknown to any other node or attackers. Second, non-acceptor nodes randomly issue covering votes to conceal the true ones.

This abstraction makes GEEC simple and efficient. GEEC first lets a node with a SGX-powered device join the blockchain in a decentralized attestation manner and maintains the list on the blockchain. When appending a block, GEEC selects a deterministic list of stealth acceptors from GEEC nodes and includes the encrypted list in the block. Only the true acceptors in the list know their identities. Then, GEEC runs a simple sub-protocol derived from Paxos [40], which seeks majority votes from these acceptors and consistently append blocks. Moreover, since SGX guarantees the integrity of GEEC's protocol, GEEC does not rely on the honesty of any node.

We implemented GEEC on the Ethereum [13,62] public blockchain platform. GEEC includes two promising applications: (1) GEEC-DB, a decentralized privacy-preserving database and (2) GEEC-ToR, a decentralized router directory service for the SGX-ToR [35] anonymous network. We evaluated GEEC with three blockchain systems: Ethereum [13, 62], EOS [25], and Intel-PoET [56]. 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.
- GEEC is easy to use for making applications decentralized.

Our major contribution is GEEC, the first efficient and strongly consistent consensus protocol for public 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 shows the efforts of building decentralized confidential application with GEEC's APIs. §7 gives implementation details. §8 shows our evaluation, §9 introduces related work and §10 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 to verify its correctness. Blockchains are divided into *private* and *public* blockchains. Private blockchains (e.g., Hyperledger Fabric [8, 14, 59], Hyperledger Sawtooth [56], and RSCoin [19]) know the identifies of all participating nodes and runs a consensus algorithm (e.g., Raft [51] or BFT [16, 59]) 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 [13, 62] provides a smart contract mechanism where users can run Turing-complete deterministic applications (e.g., lottery systnems). Moreover, digital voting, digital identity verification, and decentralized storage trading are being developed on public blockchains [11]. These promising applications often desire efficiency and strong consistency.

[1] GEEC stands for GEneral, Efficient and Consistent consensus for public blockchains.

Currently, most blockchain applications are on-chain applications: all their state transitions are always stored on the blockchains as transactions. A recent work [17] proposes off-chain executions, where applications do the smart contract execution with SGX-enabled computing node and upload their states periodically to the blockchain consensus nodes.

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) [32,43,44] 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 [7, 18, 33] (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 [12]). All SGX-based blockchain systems [1,3,46,56,64] 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 [32] which calls the hardware based pseudorandom generator(PRNG) through RDRAND on Intel CPUs [18]. Previous studies show that this random number generator is safe and cannot be altered from outside the enclave [9, 30, 47].

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 three 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. Third, liveness. Because of the FLP impossibility [28], a blockchain system cannot achieve both strong consistency and liveness in a network that may have partitions. GEEC makes the best effort to achieve liveness.

For SGX, GEEC has the same threat model as typical SGX-based systems [3, 46, 56, 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 [3]) 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

Figure 1 shows the architecture of GEEC. 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 want 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.

Referee module (§6) is invoked by GEEC for building various applications. To prove an off-chain application's SGX enclave is set up in a GEEC node, this module does a

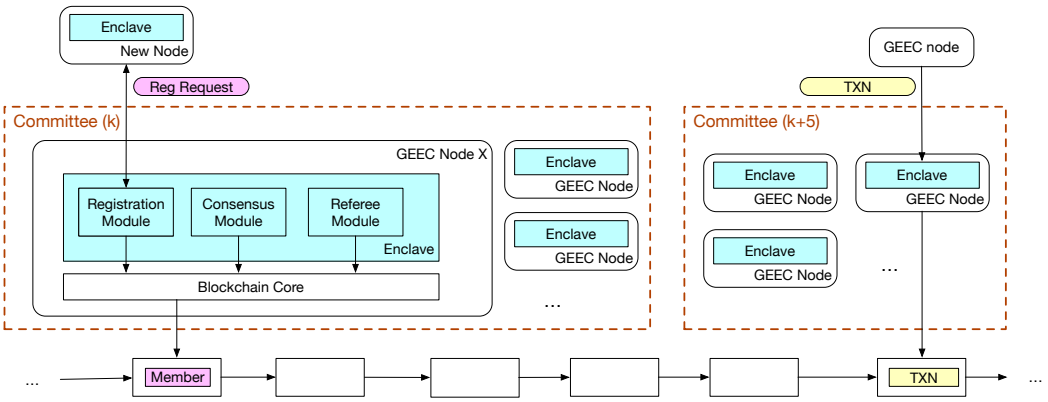


Figure 1: GEEC's architecture.

local attestation and proposes a transaction that confirms the setup. An on-chain application does not need to invoke this module.

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 (0_{th}) 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_c).

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 (§6). Then the creator generates the genesis block with $Block_0 = (pK_{shared}, member(pK_0, Q_0), H_c)$.

4.2 On-chain Node Join

In GEEC, the registration module on each registered node serves as a challenger for attesting a newly joining node i with three steps. First, the new node sets up the enclave code (with hash H_c). 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 (§3.2) $\langle Sign_{sk}(member(pK_i, Q_i)), pk\rangle$. 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 stealth acceptors are derived. The committee is explicit to all nodes, and it will be reformed if a block cannot be appended with a timeout. The acceptor group is stealth (only a true acceptor knows whether itself in the group) and will stay unchanged until the next block is confirmed. 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 ma-

licious 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).

A group of stealth acceptors is 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.

Algorithm 1 Proposer's algorithm

```
1:  function LEARN(C_v, N_acceptor):
2:      if C_v == 0 then
3:          blk ← GenerateBlock()
4:      else
5:          count ← 0
6:          BlkList ← [ ]
7:          Broadcast(Sign_sk(learn, C_v), pk)
8:          while count <= N_acceptor/2 do
9:              msg ← recv().verify()
10:             switch msg.type do:
11:                 case EMPTY:
12:                     count++
13:                 case NOTIFY:
14:                     count++
15:                     Append(BlkList, msg.Blk)
16:         if BlkList == [ ] then:
17:             blk ← GenerateBlock()
18:         else
19:             blk ← MaxVerBlock(BlkList)
20:     return blk
21:
22: function PROPOSE(blk, C_v, N_acceptor):
23:     count ← 0
24:     Broadcast(Sign_sk(propose, blk, C_v), pk)
25:     while msg ← recv().verify() do
26:         if VerifyVote(msg) then
27:             count++
28:         if count > N_acceptor/2 then
29:             break;
30:     Broadcast(Sign_sk(confirm, blk.header), pk)
```

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 [51]) 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 $\langle Sign_{sk}(elect, r, n, C_v), pk\rangle$ 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.

When a node receives a *elect* 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 *elect* message's r is larger than the node's own r, the node votes for who sends the message using UDP with $\langle Sign_{sk}(vote, n, C_v), pk\rangle$. If it is the representative of other nodes, it also transfer the votes to who sends the message.

Algorithm 2 An acceptor's algorithm

```
1:  function ACCEPTORVOTE:
2:      Cv_max ← 0
3:      Blk_pending ← null
4:      while msg ← recv().verify() do
5:          switch msg.type do:
6:              case PROPOSE:
7:                  if msg.Cv > Cv_max then
8:                      msg.Cv > Cv_max
9:                      hash ← msg.blk.header
10:                     ci ← Enc_msg.pk(vote, hash, pk)
11:                     Reply(Sign_sk(ci), pk)
12:                     Blk_pending ← msg.blk
13:             case LEARN:
14:                 if Blk_pending == null then
15:                     ci ← Enc_msg.pk(empty, pk)
16:                     Reply(Sign_sk(ci), pk)
17:                 else
18:                     B ← (Blk_pending, Cv_max)
19:                     ci ← Enc_msg.pk(notify, B, pk)
20:                     Reply(Sign_sk(ci), pk)
```

5.3 Confirming a Block

We derive GEEC's proposer and acceptor algorithms from "Paxos Made Simple" [40] 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.

GEEC's acceptors are stealth with three properties. First, only each GEEC node's code running in SGX knows whether itself is a true acceptor or not because of the aforementioned certificate mechanism. Second, the group of acceptors is derived from the last confirmed block with a deterministic group size. This approach can achieve a strong consistency (not probability based), so we do not use random function based approach (e.g., Algorand [29]). Third, to prevent attackers from finding true acceptors, GEEC non-acceptors also randomly generate covering messages that can only distinguished by the proposer. To achieve this, GEEC adds a special flag in all acceptors outbound messages, and only the true acceptors set the flag. The messages that an (true and covering) acceptor sends to a proposer is encrypted with the proposer's pk and thus the message can only be decrypted by the proposer. The message also contains the acceptor's pk to make them different.

5.4 Proof Sketch of Correctness

In this subsection, we provide a sketch of proof on the strong consistency (safety) guarantees of GEEC. Evaluation (§8.4) also shows that GEEC has reasonable liveness on network anomalies. We prove GEEC's safety by induction. Suppose GEEC guarantees safety from the 0_{th}

(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 based 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 learnt 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 Application

6.1 From On-chain to Off-chain

GEEC supports on-chain decentralized applications by the Ethereum Virtual Machine (EVM) [13]. Recent work [39, 59] reveals that confidentiality of the applications' transactions are important. To preserve confidentiality, GEEC provides a special transaction type called "secret" transaction. Secret transactions are encrypted by the key pair pK_{shared}, sK_{shared} shared by all GEEC nodes. Secret transactions can be only decrypted inside SGX enclaves so that these transactions' plaintext will not be exposed to the public. Also, access policies of these transactions can be clearly specified on GEEC's blockchain and enforced by SGX.

Another key feature brought by Intel SGX is the integrity of application code executions. GEEC introduces a Referee Module in each node (§3.2), which enables off-chain executions: on-chain smart contracts can trust the executions and results of off-chain application enclaves attested by the Referee Module.

Figure 2 shows a simple interaction protocol between the off-chain execution enclave and GEEC's blockchain presented by us. In this protocol, a transaction is modeled as an event that describes a transition of an application's on-chain state; an application's off-chain enclave code works as event handlers to process the events that has been registered by the application's on-chain smart

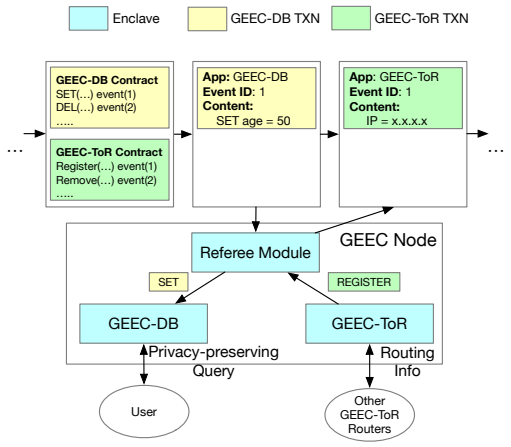


Figure 2: GEEC's off-chain interaction protocol.

contract; the Referee Module on each GEEC node is the event dispatcher that forwards the registered events to the off-chain enclaves on the local node.

An application can contain two parts of code: on-chain code (a smart contract that registers events) and off-chain code (arbitrary C/C++ code ran in SGX). For instance, to port a database in GEEC, the developer needs to write the on-chain code that includes the database's storage state and update functions (e.g., *set()*). Then, the developer needs to put these update functions as an event handler function in off-chain code's event listening loop. The loop sets up a secure channel with its local Referee Module.

To publish the database application, the database developer first registers the application's on-chain code on GEEC's blockchain and the off-chain code's hash so that all Referee Modules can verify the code. Second, if a GEEC node wants to run the application, it installs an off-chain enclave with the application code and asks the local Referee Module to do a local attestation. Third, the Referee Module proposes a transaction signed with its own private key to GEEC's blockchain to register the newly installed off-chain enclave. Then, the Referee Module starts to dispatch registered events from the chain down to the enclave.

In the opposite (bottom-up) interaction direction, an off-chain enclave can propose a transaction to the blockchain at any time by contacting the local referee module. For instance, the application can periodically checkpoint its execution states and put onto the blockchain using the transactions. In the next two subsections, we illustrates how GEEC can be used to design various decentralized applications and improve the applications' security and reliability.

Ekiden [17] is a SGX-powered system for blockchain applications. It offloads the execution of smart contracts

to a small group of SGX powered computing nodes, so that it can avoid the repetitive, redundant smart contract executions on all consensus nodes. ShadowEth [63] works similar as Ekiden. To deploy an application in Ekiden, the application has to be mostly re-written using the smart contract language [13]. To mitigate such re-write, GEEC's interaction protocol can easily port an application written in general languages (e.g., C/C++) by only registering on-chain events.

6.2 Decentralized Privacy-preserving DB

Recently, protection on data privacy has gained much attention. For instance, Facebook and Cambridge Analytic Ltd. leak personal information of over 50 million users [15]. EU issued the General Data Protection Regulation (GDPR) to enhance protection on personal data [21]. However, this regulation can be easily violated in a centralized data administration. To tackle this problem, we designed GEEC-DB, a GEEC powered database that enables decentralized privacy-preserving data sharing and analytics. We choose an open-source light SQL database [41] onto GEEC and made the database decentralized. We use this database because it was easy to include its dependent libraries into SGX.

GEEC-DB has three features. First, all data updates are committed to blockchain as secret transactions. Second, data can only be decrypted inside the Referee Module and transferred by secure channel to GEEC-DB's off-chain enclaves. Third, data can only be used by differentially-private queries (e.g., *sum()*) running on off-chain enclaves.

GEEC-DB invokes interaction protocol in §6.1: The GEEC-DB's local query engine runs in off-chain enclaves. An "update" event that defines how the data can be updated is registered on GEEC's blockchain. A data owner (e.g., Hospitals) manages her data by joining GEEC and updating data using GEEC's secret transactions. A data user (e.g., Research Institutions) joins GEEC and installs GEEC-DB's off-chain execution enclaves attested by GEEC. The encrypted data on blockchain will then be decrypted by GEEC referee module and transferred to these off-chain enclaves. The off-chain enclaves can then supply privacy-preserving query results to the data user.

6.3 Decentralized ToR Directory Service

ToR is a network protocol that enables anonymous communication. SGX-ToR [36] enhances the security of ToR by porting ToR routers into SGX enclaves. However, the ToR router directory service, which contains information (e.g., IP addresses) of SGX-ToR routers, is still centralized (run on a set of dedicated machines). This limits the availability of ToR on targeted attacks.

We present GEEC-ToR by porting SGX-ToR's directory service onto GEEC. In GEEC-ToR, each SGX-ToR router is modified to run in a GEEC's off-chain enclave. The information of these routers is encrypted and stored on GEEC's blockchain. Only registered off-chain application enclave can access the information and build a ToR network circuit.

7 Implementation Details

We implemented GEEC in the Golang implementation of Ethereum [26] (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 cgo 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 a 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.

8 Evaluation

Our evaluation was done on both the Tencent public cloud [60] 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 [6] 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. The scalability (§8.3) and robustness (§8.4) evaluation was done on the public cloud, and the rest in our cluster.

We compared GEEC's performance with four blockchain systems, including three public blockchains (Ethereum [26], EOS [25], and Snow-white [10]) and one SGX-based private blockchain system, Hyperledger-Sawtooth (Intel-PoET) [55]. These systems cover two PoW systems (Ethereum and Sawtooh) and two PoS systems (EOS, and Snow-white). We ran three of the systems (Ethereum, EOS, and Sawtooh) 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. We focus on the following questions:

- §8.1 : Is GEEC's blockchain consensus protocol efficient?
- §8.2 : How sensitive is GEEC's performance to its parameters?
- §8.3 : How does GEEC's performance scale with the number of nodes on Internet?
- §8.4 : How robust is GEEC on various network anomalies on Internet?
- §8.5 : How easy is GEEC to support applications?
- §8.6 : What did we learn from GEEC's limitations and future potential?

8.1 Efficiency

Figure 3 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 [25] 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

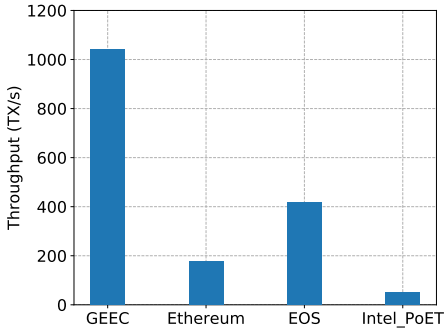


Figure 3: Throughput of blockchains in our cluster.

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 [25]. PoET achieved a low throughput because it uses PoW. Our result is similar to a recent study's [**?**]. Snow-White [10] 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.

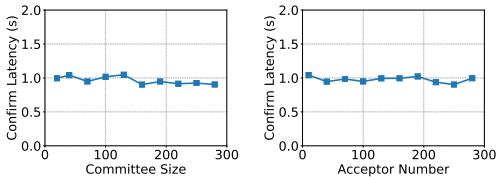
Recently, Hyperledger-Fabric [8, 14] reported a notable throughput of about 10K transaction/s using a byzantine ordering service. Hyperledger-Fabric is a private blockchain which is not designed to scale to nodes from the Internet, while GEEC is for public blockchains.

Table 2 shows that GEEC incurred only 0.93s confirm latency. GEEC's confirm latency is 88.5X faster than Ethereum, while GEEC's throughput is 6.1X higher than Ethereum. This is because in Ethereum, nodes should wait 6-10 appended blocks before having great confidence on confirming a block, while GEEC confirms whenever a block is appended (§5). Table 3 shows the latency breakdown of GEEC's protocol, which shows that time for learning and proposing a block (§5.3) is the major part. This is because GEEC broadcasts the proposed block on its P2P network in this phase. This P2P broadcast time is fundamental in a blockchain P2P network, and it is proportional to approximately the square root of the number of nodes. This implies that GEEC's block confirmation latency could be scalable on Internet.

System	GEEC	Ethereum	EOS	Intel_PoET
Latency (s)	0.93	82.3	43.1	45.2

Table 2: Confirm latency of blockchains in our cluster.

Table 4 shows the micro-events of GEEC. The ECall column shows the number of times that GEEC's proposer node entered its SGX enclaves on appending a block. Since each ECall only takes around 3us, and GEEC's proposer only did 97 ECalls on average for each block,



(a) (b)

Figure 4: Sensitivity of committee/acceptor size.

running in SGX hardware mode and simulation mode makes little difference for GEEC's performance. The other columns shows the average computing power consumption of each node, which is little.

Elect	Learn & Propose	Confirm
281.2 ms	513.2 ms	125.1 ms

Table 3: GEEC's latency breakdown: Elect (§5.2), Learn & Propose (§5.3), and Confirm (also §5.3).

Blk Size	TX/Blk	# ECall	CPU	Network
40 KB	1000	97	24.4%	2 Mbps

Table 4: Proposer's micro events for appending a block.

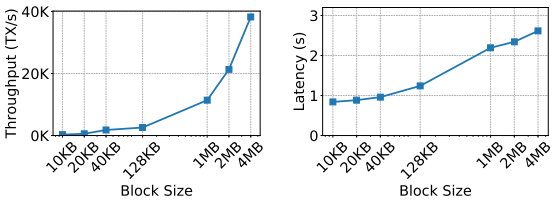
8.2 GEEC's Parameter Sensitivity

GEEC's throughput and confirm latency depend on three important protocol parameters, the size of a committee, the number of acceptors, and block size. Figure 4 shows the sensitivity on the committee size and the number of acceptors. These two parameters turns out to be in-sensitive because the latency is dominated by the time for broadcasting the new block on the P2P network (Table 3). Therefore, we set GEEC's default committee to be 30, same as Algorand's, and this setting works for all experiments in our evaluation. The number of acceptors affects the liveness of GEEC, and our evaluation shows that our default setting achieves good liveness on various network anomalies.

Figure 5 shows GEEC's performance sensitivity on block size. When the block size was larger (more transactions were included in each block), GEEC's throughput did increase, but its block confirm latency also increased. In our evaluation, to be close to real world usage, we conservatively set GEEC's block size to be 40KB, the same as Ethereum main chain's average block size [13].

8.3 Scalability

To evaluate GEEC's scalability, we booted up 100-10000 nodes on the Tencent public cloud, and evaluated its confirm latency. Figure 6 shows the scalability of GEEC's



(a) Throughput (b) Confirm Latency

Figure 5: Sensitivity of block size.

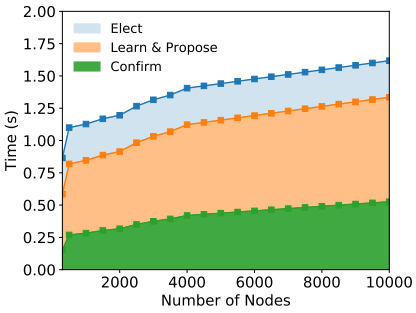
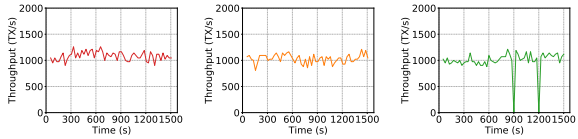


Figure 6: Scalability to the number of nodes on Internet.

latency. The latency is divided into three parts. This figure shows that the learn and propose phase of GEEC (§5.3) is the dominant factor because it broadcasts the proposed 40KB block on the P2P network. Since the broadcasting of new blocks is fundamental in all blockchain systems, GEEC's scalability is similar to Algorand's.



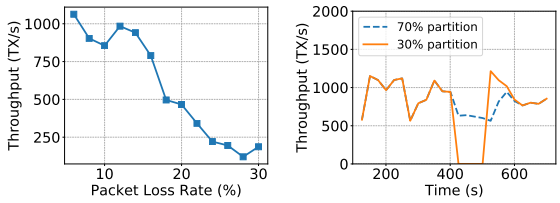
(a) offline rate = 1% (b) offline rate = 3% (c) offline rate = 4%

Figure 7: Throughputs with offline rates. We killed nodes of different offline rate every 300s for 5 times.

8.4 Robustness

We first made different portions of nodes offline by five times and measured GEEC's throughputs, shown in Figure 7. For instance, even when we made 4% of nodes go offline for five times (finally, 20% of nodes were offline) GEEC was still able to confirm blocks.

We also randomly dropped different portions of packets in the GEEC's network. GEEC's performance dropped significantly only when the portion of dropped packets were higher than 15%. Previous work shows that such a high packet loss rate already significantly downgrades the quality of most Internet applications [52].



(a) Packet Loss (b) Partition

Figure 8: Throughputs on packet loss and partitions.

We manually created a 75%-25% partition of our VM instances by discarding packets between the two partitions. Figure 8b shows that GEEC can still confirm blocks with slightly lower throughput in the major partition, because some committee failed to elect a proposer and a committee change was triggered (§5.1). When the partition recovered, the nodes in the minor partition got their missed blocks and the system worked as before.

8.5 Ease of Deploying Applications

As mentioned in §6, we introduces two decentralized applications which invoked off-chain executions enabled by GEEC. GEEC-DB needs to modify only 1045 LoC (623 LoC is for porting into SGX and 422 LoC is for porting into GEEC) in order to port an open-source light SQL database [41] onto GEEC and made the database decentralized. We use this database because it was easy to include its dependent libraries into SGX. We evaluated GEEC-DB's performance it by updating 100K shared data entries and it took 100K transactions to be committed and overall around 730s to confirm all transactions on GEEC.

SGX-ToR [35] uses SGX to improve the security and scalability of ToR. SGX-ToR's router directory service is centralized as it runs on a few machines. We modified 287 LoC modification to commit its router information on GEEC, making the directory service decentralized on GEEC's blockchain. We evaluated GEEC-Tor by setting up 10 SGX-ToR routers and updating its information on the GEEC based directory service, and it took 20 transactions and merely 32s for setting up 10 nodes sequentially.

8.6 Discussion

GEEC requires a joining node to have a TEE implementation with a remote attestation scheme in its device. Although GEEC's current implementation uses Intel SGX, GEEC's protocol is agnostic to TEE products, and it can support different TEE products. Although a compromised TEE device can affect the egalitarian of GEEC, REM [64] has shown that this can be mitigated by monitoring the distribution of block proposer and removing suspected devices from list.

To ensure good liveness, GEEC uses a reward mechanism (§5) to encourage a majority (e.g., 70%) of nodes to stay online. This is a reasonable online requirement because GEEC's online nodes consume little computation power. Moreover, GEEC's node join mechanism (§4.2) requires a node to rejoin after a number of blocks are confirmed, effectively eliminating zombie nodes. GEEC's consistency does not need this online requirement.

GEEC has demonstrated that it can easily support diverse applications, making the applications decentralized and highly available. GEEC can be further extended to support promising blockchain application (e.g., edge computing), and we leave them for future work.

9 Related Work

Public Blockchain System *Proof of Work.* BitCoin [48] is the first PoW based blockchain system that works as a fully functional cryptocurrency system, but it suffers from hugpropose toe energy consumption and bad performance. Bitcoin-NG [27] and Fruitchain [53] use multiple types of blocks to improve the throughput of Bitcoin. GHOST [58] proposes to select a subtree rather than the longest chain when the blockchain forks to reduce wasted blocks. Ethereum [13, 62] reduces the difficulty of hash computations in PoW and introduces the uncle block mechanism to reduce potential forks. Solidus [2], ByzCoin [37], PeerCensus [22], and Hybrid Consensus [54] use PoW to select a group of committee and runs a byzantine fault tolerant (BFT) protocols [16] to achieve consensus on a single block. Elastico [42] and OmniLedger [38] propose a sharding approach inspired by traditional database to linearly scale existing blockchains' throughput. These works improves 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. GEEC has good performance and does not assume the honesty of the nodes.

Proof of Stake. Algorand [29, 45] selects nodes to form a committee with the probability proportional to each nodes' number of coins (i.e., stake). It assumes an extremely powerful adversary that can compromise the committee members once they send out packets, so 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 only sends one messages in each round. GEEC differs from Algorand in three aspects. First, to preserve strong consis-

tency, Algorand assumes a vast majority (e.g., 80%) of the users to be honest because it works on arbitrarily devices, while GEEC does not need this assumption by leveraging SGX. As shown in our evaluation (§8.4), GEEC proceeds correctly even when more than 20% of nodes are offline. Second, Algorand strongly tolerates DoS attacks because each node only sends out one message in each round; GEEC uses non-acceptors to generate fake votes to cover the real stealth acceptors (§5.3), so GEEC tolerates DoS attack for these acceptors (slightly weaker than Algorand). Third, GEEC's protocol has few parameters (Table 1), and Algorand has 12 parameters.

Praos [20, 34] and Snow-White [10] are two PoS systems similar to Algorand that each node independently determines whether it is the proposer of next block. Praos uses a secure multi-party computation algorithm; Snowwhite uses a PoW similar hash computation mechanism and set each node's hash target according to their stakes. EOS [25] is a PoS system that uses off-band 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 GEEC, 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 [8, 14] trusts its committee nodes which are statically configured by consortium policy. These nodes achieve consensus by contacting a byzantine tolerant ordering service [59, 61] runs on another sets of partially trusted nodes. RSCoin [19] lets a set of trusted authorities to run a cryptocurrency and divides them into different levels to improve performance. BlockBench [23] is an evaluation framework for comparing different private blockchain systems. In general, private blockchains are efficient but are not designed to work in the Internet scale.

Blockchain Applications. Diverse applications have been developed on blockchains. Ethereum [13] introduces EVM, a deterministic runtime, to run smart contract applications on all consensus nodes of a blockchain. Hawk [39] and zkLedger [49] focus on enhancing confidentiality of smart contracts. Ekiden [17] offloads the execution of smart contracts to a small group of SGX powered computing nodes, so that Ekiden 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 application has to be mostly re-written using the smart contract language [13]. Unlike Ekiden and ShadowEth, GEEC provides a general event-driven interaction protocol (§6) to deploy an application with moderate code changes. BlockStack [5] builds decentralized DNS and storage services on blockchains. Overall, GEEC'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 [56] efficiently replaces the PoW puzzles with a trusted timer in SGX. GEEC is for public blockchains that any node can join, while PoET is a private blockchain with a known member list. Moreover, our evaluation (§8.1) shows that GEEC 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 [46] 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 [1] is an ongoing private blockchain project that can support diverse consensus protocol (e.g., Raft [51]) in SGX-powered nodes. Currently, CoCo has not described a detailed consensus protocol or evaluation results. SCIFER [3] 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.

10 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`.

References

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

[2] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman. Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus. https://arxiv.org/abs/1612.02916, Dec 2016. Accessed: 2017-02-06.

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

[4] 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.

[5] M. Ali, J. Nelson, R. Shea, and M. J. Freedman. Blockstack: Design and implementation of a global naming system with blockchains. http://www.the-blockchain.com/docs/BlockstackDesignandImplementationofaGlobalNamingSystem.pdf, 2016. Accessed: 2016-03-29.

[6] https://aws.amazon.com/ec2/instance-types/.

[7] 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.

[8] 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.

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

[10] 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.

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

[12] 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.

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

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

[15] C. Cadwalladr and E. Graham-Harrison. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. *The Guardian*, 17, 2018.

[16] 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.

[17] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution. *arXiv preprint arXiv:1804.05141*, 2018.

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

[19] G. Danezis and S. Meiklejohn. Centrally banked cryptocurrencies. In *Network and Distributed System Security*. The Internet Society, 2016.

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

[21] 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.

[22] 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.

[23] 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.

[24] J. R. Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

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

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

[27] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Security Symposium on Networked Systems Design and Implementation (NSDI'16)*. USENIX Association, Mar 2016.

[28] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. volume 32, pages 374–382. ACM, 1985.

[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] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.

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

[33] 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.

[34] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. https://pdfs.semanticscholar.org/1c14/549f7ba7d6a000d79a7d12255eb 2016. Accessed: 2017-02-20.

[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] 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.

[37] E. K. 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.

[38] 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.

[39] 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.

[40] L. Lamport. Paxos made simple. http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf.

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

[42] 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 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

[43] 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.

[44] 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.

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

[46] 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.

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

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

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

[50] C. Natoli and V. Gramoli. The balance attack against proof-of-work blockchains: The R3 testbed as an example. http://arxiv.org/abs/1612.09426, 2016. Accessed: 2017-02-15.

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

[52] `https://en.wikipedia.org/wiki/Packet_loss`.

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

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

[55] `https://www.hyperledger.org/projects/sawtooth`.

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

[57] M. Rosenfeld. Analysis of hashrate-based double spending. `http://arxiv.org/abs/1402.2009`, 2014. Accessed: 2016-03-09.

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

[59] 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.

[60] `https://cloud.tencent.com/?lang=en`.

[61] M. Vukolić. Rethinking permissioned blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, pages 3–7. ACM, 2017.

[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/2017/179`, 2017. Accessed: 2017-03-24.