PoQ: A Consensus Protocol for Private Blockchains Using Intel SGX

Golam Dastoger Bashar, Alejandro Anzola Avila, and Gaby G. Dagher

CS, Boise State University, Boise ID 83725, USA
{golambashar,alejandroanzolaa}@u.boisestate.edu,
gabydagher@boisestate.edu

Abstract. In blockchain technology, consensus protocols serve as mechanisms to reach agreements among a distributed network of nodes. Using a centralized party or consortium, private blockchains achieve high transaction throughput and scalability, Hyperledger Sawtooth is a prominent example of private blockchains that uses Proof of Elapsed Time (PoET) (SGX-based) to achieve consensus. In this paper, we propose a novel protocol, called Proof of Queue (PoQ), for private (permissioned) blockchains, that combines the lottery strategy of PoET with a specialized round-robin algorithm where each node has an equal chance to become a leader with equal access. PoQ is relatively scalable without any collision. Similar to PoET, our protocol uses Intel SGX, a Trusted Execution Environment, to generate a secure random waiting time to choose a leader, and fairly distribute the leadership role to everyone on the network. PoQ scales fairness linearly with SGX machines: the more the SGX in the network, the higher the number of chances to be selected as a leader per unit time. Our analysis and experiments show that PoQ provides significant performance improvements over PoET.

Keywords: Blockchain · Consensus · Permissioned · SGX · Fairness.

1 Introduction

At the core of any blockchain platform, there is a ledger that is maintained by a trustless P2P network. Due to the untrustworthy nature of the network, there needs to be a way for the nodes in the network to reach an agreement among them on the valid transactions that can be appended to the ledger. Consensus protocols serve as mechanisms to reach such agreements in the network. While existing protocols solve the consensus problem fairly well, they also have their own shortcomings. A consensus protocol in a blockchain system is typically required to support three properties: *liveness* (transactions are added to the ledger in a reasonable time), *consistency* (all parties have the same view), and *fairness* (all nodes are equally likely to mine the next block) [4].

Public blockchain networks (e.g. Bitcoin [18] and Ethereum [7]) that use proof of work (*PoW*) [18], proof of capacity [15] or proof of activity [6], require a large amount of computational power, which is a misuse of resources and limits transaction throughput (usually expressed as transactions per second). On the other hand, proof of stake [19] and proof of burn[1] [4] are environment-friendly consensus protocols due to the insignificant computation requirements; however, they suffer from the "rich get richer" problem [5].

Private blockchains, on the other hand, require a centralized party (or a consortium of them) to control who joins the system and at what capacity (mine, view, transact, etc). Such reliance on the centralized party leads to a reduced cost for reaching consensus, high transaction throughput, improved scalability to support new nodes and services, and higher efficiency. In private blockchains, the level of access, visibility, and execution can be controlled. Private blockchains are more appropriate to a consortium of organizations, like the banking sector or the insurance industry, where participation is selective with known identity and may operate under a shared governance model [14]. Examples of private blockchains include Ripple (XRP) [20] and Hyperledger [3]. Hyperledger Sawtooth project was introduced by Intel as a modular blockchain that uses Proof of Elapsed Time ($PoET$) consensus protocol to implement a leader election lottery system [13][8]. In $PoET$, each miner node is randomly assigned a `waitTime`, and as soon as this `waitTime` expires, the specific node creates and publishes the next block on the network [8]. The protocol acts as a mix of first-come-first-serve ($FCFS$) and random lottery [21].

Contribution. In this paper, we propose a variant of $PoET$, we call it PoQ, that regulates how nodes compete to finish their `waitTime` such that the average wait time and number of leadership each node is assigned is approximately the same across all nodes. Our goal is to optimize the performance of the network concerning *throughput* and *scalability.* PoQ determines which node should execute its `waitTime` when there are multiple run-able nodes in the queue. To achieve this, we introduce the concept of *dynamic Quantum Time* (QT) indicating the amount of time a node will get the chance to execute for a single pass, which has a major impact on resource utilization and overall performance of the network. Some of the extended characteristics that differentiate our work over others include fast transaction processing, low energy consumption, fair distribution, and easy block verification (deterministic). PoQ avoids high resource utilization and replaces it with a true randomized system. Similar to $PoET$, PoQ uses execution environments in trusted hardware, more specifically Intel SGX [10], to achieve consensus while preventing tampering. Through the use of Intel SGX enabled CPUs, we enforce correct execution of code and guarantee the "one node one machine" policy (to prevent Sybil attacks) for all nodes in the network. We implemented PoQ in a distributed SGX environment, and our analysis and experiment results show that PoQ provides significant performance improvement over PoET.

[1] https://en.bitcoin.it/wiki/Proof_of_burn

2 Related Work

In recent years separate approaches are used to extend the performance of $PoET$. Research has been conducted to achieve a good overall performance in a private environment. In addition, there are existing related works that spotlight the perfections behind the intention of Trusted Execution Environment (TEE) design. Hardware-based TEE like ARM TrustZone (available on smartphones) or Trusted Platform Module and Intel SGX (for x86-based computer) are generally obtainable in commodity computing platforms. In the paper [8], authors provide remarks on the design of Sawtooth. In order to reduce potential collision, they discussed waiting times need to be longer. While [12] focusing on reducing the *stale block rate* by restricting the number of nodes which they call $PoET+$. A stale block is an accurate, previously announced block that does not belong as part of the longest chain. A stale block appears at any time when more than one node announces a valid block within a short duration. Proof of Luck (PoL) [17] is another consensus protocol based on TEE and similar to $PoET$ because it also generates random numbers from SGX into the block referred to as '*luck*' of the block. The protocol selects the chain with the highest accumulative luck as the winner and is determined the luckiest. The luckiest block is then added to the chain. It will generate forks when the network is periodically portioned because the partitions will ensure various largest accumulative luck. In [2], the authors proposed Proof of TEE-Stake ($PoTS$), that leveraging functionality from TEE for public blockchain, where each node in PoTS ensures the same structure to bootstrap a TEE program. In [11], the authors explore the response of throughput of $PoET$ and propose a simple adjustment to it (they termed as "$S - PoET$") which leads to a higher throughput as the network becomes larger. According to the authors, if the shortest `waitTime` and another `waitTime` are conflicted by fewer than the propagation delay then that will result in a stale block.

At the beginning, $PoET$ was preferred to replace the PoW with the exception of the longest-chain rule [22]. Due to its access control nature, it is most appropriate for permissioned blockchains, where certain works will be executed on TEE by certain authenticated nodes. Unlike permissionless, most permissioned blockchains don't require any rewards mechanism. Each consensus protocol is unique based on the way it creates a block, discloses the evidence (block propagation), the procedure of validation inside the network, and the rewards system for an honest effort. Table 1 shows an assessment among some of the protocols designed for TEE based or not [5]. The throughput measurements derive from the complimentary white paper or formal demonstration of the implementation of that protocol and indicate the scales of fastness, i.e. high or low.

Due to dynamic QT in PoQ, all nodes are approximately given the same priority to execute depending on its tier (A relationship engaging level between a set of nodes. PoQ is a multi-tier approach to the early recognition of nodes and provides necessary data to them), thus no nodes are left behind which leads to more speed in the system. PoQ progresses in a round-robin way, wherein each round, a selected node within all the nodes in the network will get a chance to reduce its `waitTime` and if it is successful to reduce all its `waitTime`, then that particular node proposes the potential block (a successive set of transactions). Thus, a node cannot get a total allocation of time above its assigned time. We can say that *PoET* is suitable for a limited network with small `waitTime` while in PoQ arrival time of a node puts great importance on becoming the leader quickly as it maintains a dynamic queue that pushes the nodes based on their arrival time. That means when a node appears in the network it starts to compete with other nodes and the leadership cannot be predetermined. It is suitable for a large number of participants, easy to implement, and also offers similar average `waitTime` and average elapsed time for all its nodes. Due to no hashing is required in PoQ, we can say it also saves energy too. To the very best of our knowledge, no work has been done to make the lottery election system fairer for *PoET*.

Table 1: Assessment of blockchain consensus protocols. Symbols for binary values: ✓ means yes, ✗ means no. Symbols for non-binary values: ● means high, ◗ means medium, ○ means low and ⊘ indicates undefined in the protocol white paper.

Concensus Protocols	Type		Block propagation	Block Validation	TEE based	Resource consuming	Rewards	Nodes execution	Throughput (tps)	Fairness
	Public	Private								
Nakamoto (BTC, Litecoin)	✓	✗	PoW	PoW (longest chain)	✗	●	✓	●	○	●
Nakamoto (Etheruem)	✓	✗	PoW (Ethash)	PoW (GHOST)	✗	●	✓	●	○	●
PoET (Hyperledger)	✗	✓	PoET within TEE	TEE certificate	✓	○	⊘	◗	●	●
PoTS	✓	✗	PoTS-based committee election	TEE, eligibility	✓	○	⊘	⊘	⊘	⊘
Chain-based PoS (Nxt)	✓	✗	PoS	PoS (longest chain)	✗	◗	✓	⊘	○	○
PoI (XEM)	✓	✗	PoI Harvesting	Importance score	✗	○	✓	◗	●	●
PoL	✗	✓	PoL	Longest total value of luck	✓	○	⊘	◗	⊘	●
PoQ	✗	✓	PoQ within TEE	Completion of wait time	✓	○	✗	●	●	●

3 Background

3.1 SGX

Intel Software Guard Extension (*SGX*) is fully implemented in the CPU hardware and yields a partial element to execute within an isolated environment, referred to as an enclave [10]. Generally, SGX breaks down an application into two logical segments: enclave and untrusted part (conventional application). Furthermore, an SGX application can handle 5-20 enclaves. The code in the enclave is used to handle the secret data. On the other hand, the remaining portion of the code, along with all its modules, keeps in the untrusted part. Interaction within these two parts happens via the call gate. A function call that enters the enclave from the untrusted portion is called an Enclave CALL (ECALL). A call

Table 2: Comparison of SGX based system. Symbols: ● means fully-provided, ✗ means unsupported and ⊘ means unspecified.

	Energy efficient	Higher throughput	Scalability
PoET	●	⊘	✗
PoL [17]	●	⊘	✗
REM [23]	✗	✗	✗
PoQ	●	●	●

within the enclave to an untrusted portion is called an Outside Call (OCALL). Figure 1 provides a high-level view of ECALL and OCALL communication. By definition, an OCALL is made from within an ECALL because an ECALL needs to enter the trusted portion. Figure 2 shows the execution of an SGX application and the way SGX safeguards an enclave from any envious program, including OS, BIOS, drivers, and firmware which pretends to steal application secrets[2].

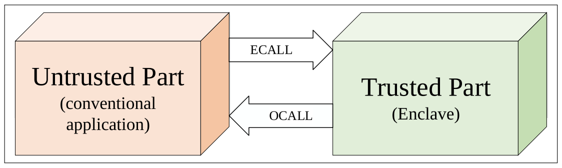


Fig. 1: Interaction between enclave and untrusted part in an SGX application.

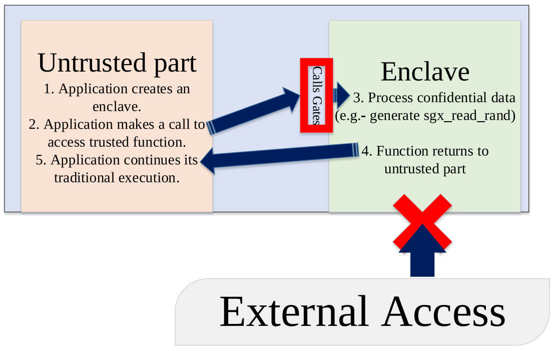

Untrusted part
1. Application creates an enclave.
2. Application makes a call to access trusted function.
5. Application continues its traditional execution.
Calls Gates
Enclave
3. Process confidential data (e.g.- generate sgx_read_rand)
4. Function returns to untrusted part
External Access

Fig. 2: Flow of execution in SGX application.

[2] https://software.intel.com/en-us/articles/intel-software-guard-extensions-tutorial-part-1-foundation

SGX depends on remote attestation to prove to remote users that the particular portion of code is executing in a genuine SGX-enabled CPU [10]. It also presents a reliable source of random number via its `sgx_read_rand` API which calls the hardware-based pseudorandom generator (PRNG) over RDRAND on Intel CPUs. Many researchers have already established that this random number generator is secure and cannot be modified from outside the enclave [9]. Presently, C and C++ are supported by Intel's Software Development Kit (SDK) which is available for both Windows and Linux.

3.2 Abstract model of PoET

PoET is usually used on the permissioned blockchain networks to determine the leaders of the block on a specific network. After joining the network, each node must ask for a `waitTime` from an enclave and then wait for that randomly chosen `waitTime`. A node who finishes the `waitTime` first – that is, the node with the shortest wait time for an appropriate transaction block, establishes a remote attestation that provides information for verification about its honesty, carries out a new block to the blockchain, and announces the mandatory data to the network. To find the next block, an identical procedure is required. By remote attestation, *PoET* ensures that the nodes select a random `waitTime` (not purposely chosen a curtailed `waitTime`) and the leader has actually waited the allocated `waitTime`.

3.3 Remote Attestation Architecture

Remote attestation (RA) is an exceptional property of Intel SGX, to establish a secure environment between the server and the node (client) [16]. Simply in computing, the term attestation means, a procedure to verify the identity of a software and/or hardware. More specifically, RA is a medium to verify the interaction between the software and the hardware that has been founded on a trustworthy platform. By following remote attestation flow, a client enclave ensures three things: its identity, its pureness (has not been altered), and a certain piece of code executing in a genuine SGX-enabled CPU. A server sends a remote attestation request to a node and it responds to the request by announcing information about the platform configuration. Node executes the client code while the server runs the server's side code. Both parties are interacting over a network, which is not recognized to be part of any side or secured. The whole operation contains fifteen steps with the server (also called challenger) and the node. Figure 3 shows the interactions between the entities engaging in RA. It is worth mentioning that RA adopts a modified version of the sigma protocol to support Diffie-Hellmann key exchange (DHKE) among the node and the server. The sigma protocol is proof that consists of commitment, challenge, and response. SCIFER [1] uses RA to verify the identity of users. Finally, we trust Intel to execute SGX RA service correctly (similar to [1,23]).

Remote Attestation Protocol

1. In reply to the challenge request from the server, the node will do the following:
  1*a*) Initialize the enclave by `sgx_create_enclave (.., enclave_id, ..)` and perform ECALL to go into the encalve.
  1*b*) Initialize the RA flow by calling `enclave_init_ra(enclave_id,..,b_pse, context)`. Here, *pse* means platform service.

2. If *b_pse* is true then call `sgx_create_pse_session()` before establishing the RA and key session.

3. Call `sgx_ra_init(&sp_pub_key,b_pse,context)` by passing the server's public key. Key is in little-endian byte order and must be hardcoded into enclave.

4. Close PSE session by `sgx_close_pse_session()`.

5. Return *context* to the untrusted part from the enclave.

6. The untrusted part of the node call `sgx_get_extended_epid_group_id()` to get active extended group ID (GID) of enhanced privacy group ID. EPID is an anonymous signature scheme for attestation.

7. This is send to the server as a body of msg0.
  7*a*) Verify by the server. If it is not valid, server terminate the attestation flow.

8. The untrusted part of the node calls `sgx_ra_get_msg1(...,enclave-id,g_a)` where *g_a* is a public key of a node enclave and this *enclave_id* is going to be attested.

9. The untrusted Key Exchange (uKE) part of the node builds a message, msg1 that contains g_a || GID.

10. Send msg1 to the server. All elements of msg1 are in little-endian byte order.
  10*a*) Server translate all elements into little-endian order to check.

11. Server replies with msg2 that contains *g_b*, *spid*, *quote-type*, *kdf-id*, *sigRL*, etc. The public key of the server, also known as g_b is based on NIST-256. Signature Revocation List (sigRL), is a list of unfaithful signatures, signed by the revocation authority.

12. After receiving msg2, the untrusted part calls the function `sgx_ra_proc_msg2(context,enclave_id,sgx_ra_proc_msg2_trusted_t, sgx_ra_proc_msg3_trusted_t, msg2, msg2_size, ...)`
  12*a*) By calling `sgx_ra_proc_msg2`, node builds msg3.

13. `sgx_ra_proc_msg2()` builds msg3 that contains mac, g_a, and platform security property.

14. The node sends msg3 to the server and expect to get the attestation result.

15. Upon receiving msg3 from the node, the server will do the following:
  15*a*) The server verifies the msg3 by calling `sgx_ra_proc_msg3_req(msg2,msg3_size,att_result_msg)`, to compare g_a w.r.t. g_a of msg1 and verify the msg mac using sigma protocol (SMK).
  15*b*) Send attestation result message to the node.
  15*c*) The node will receive the result and checks the MAC using MK.

Protocol 1.1: Remote attestation

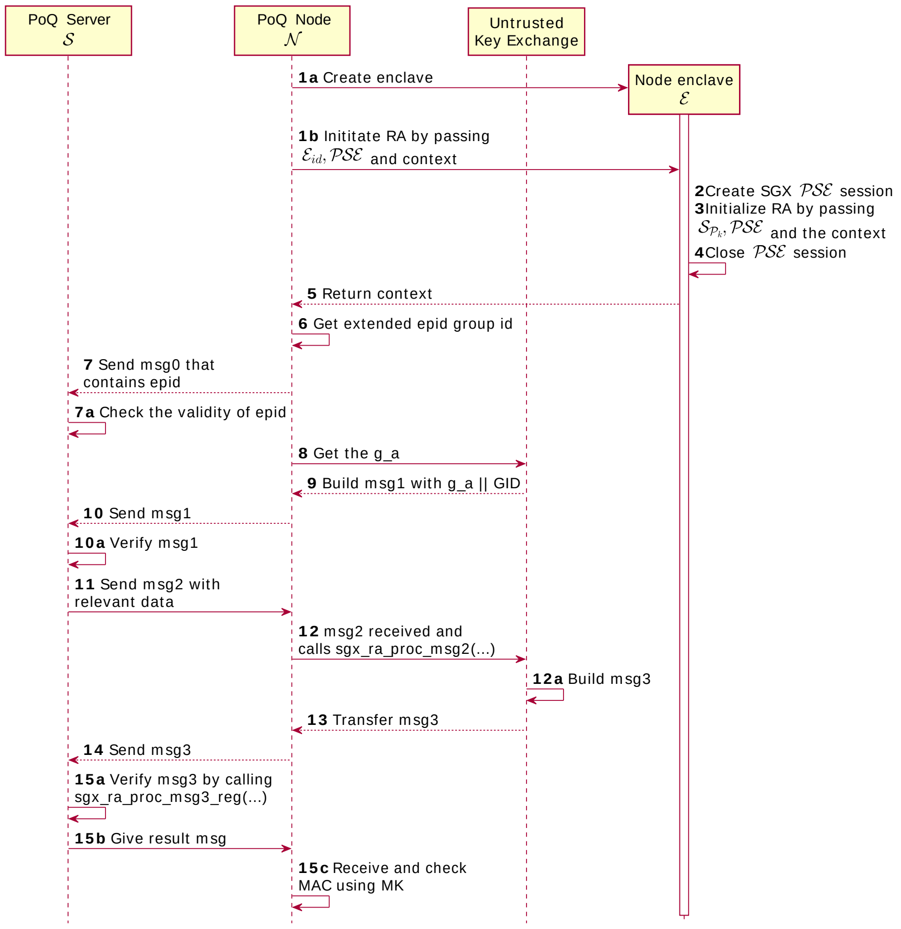

PoQ Server $\mathcal{S}$
PoQ Node $\mathcal{N}$
Untrusted Key Exchange
Node enclave $\mathcal{E}$
1a Create enclave
1b Inititate RA by passing $\mathcal{E}_{id}, \mathcal{PSE}$ and context
2 Create SGX $\mathcal{PSE}$ session
3 Initialize RA by passing $\mathcal{S}_{\mathcal{P}_k}, \mathcal{PSE}$ and the context
4 Close $\mathcal{PSE}$ session
5 Return context
6 Get extended epid group id
7 Send msg0 that contains epid
7a Check the validity of epid
8 Get the g_a
9 Build msg1 with g_a || GID
10 Send msg1
10a Verify msg1
11 Send msg2 with relevant data
12 msg2 received and calls sgx_ra_proc_msg2(...)
12a Build msg3
13 Transfer msg3
14 Send msg3
15a Verify msg3 by calling sgx_ra_proc_msg3_reg(...)
15b Give result msg
15c Receive and check MAC using MK

Fig. 3: SGX Remote Attestation.

4 Consensus Protocol: PoQ

4.1 Overview

This paper introduces a modified version of *PoET* consensus protocol called PoQ based on *SGX*. As part of this protocol, each participating node generates a random `waitTime` using the enclave $\mathcal{E}$, called SGX time $\mathcal{SGX}_t$ and waits for it to be expired. After $\mathcal{SGX}_t$ is finished, the node becomes the leader and is authorized to generate the next block. The `waitTime` and leadership for each node will be approximately the same after a certain period which achieves the equality issue of the consensus protocol.

Random $\mathcal{SGX}_t$. In our protocol, when a node joins the network, it gets a range from the server to generate a random `waitTime`, $\mathcal{SGX}_t$ from its $\mathcal{E}$. After having an $\mathcal{SGX}_t$, the node needs to submit it to the server for further verification.

PoQ Server

Initialization. The server $\mathcal{S}$ establishes a public key directory of permitted nodes, creates an empty Q (contains node id) and $\mathcal{SGX}_{\mathrm{T}}$, and then starts listening to requests from nodes interested to join the network.

Node Registration $Register(\mathcal{N}_{\mathcal{P}_k}, Sign(\mathcal{N}_{\mathcal{P}_k}))$. Upon receiving a registration request from a node, the server performs the following:

1. Check whether $\mathcal{N}_{\mathcal{P}_k}$ exists in the public key directory. Otherwise terminate the connection.
2. Check the validity of the signature. Otherwise terminate the connection.
3. Create an identification number, $\mathcal{N}_{id}$.
4. Send an acknowledgment back to the node, along with $\mathcal{N}_{id}$ and $\mathcal{SGX}_{\max}$.

Attestation $Remote\_Att(\mathcal{S}_{\mathcal{P}_k})$.The server and a node jointly execute the RA protocol (Protocol 1.1): Sends a RA request to the client to establish a secure channel.

– The server gets some information from the node which helps to decide whether the program functioning on the node is malicious or fair.

SGX Verification. $Verify(\mathcal{SGX}_t)$. Upon receiving $\mathcal{SGX}_t$, which is a randomly generated time by the node's $\mathcal{E}$ from $\mathcal{N}_{id}$, the server performs the following:

– Check whether $\mathcal{SGX}_t$ is within the range $[\mathcal{SGX}_{\min}, \mathcal{SGX}_{\max}]$. Otherwise terminate the connection.
– Add $\mathcal{N}_{id}$ to Q and build $\mathcal{SGX}_{\mathrm{T}}$ that has $\mathcal{Q}_t$ and $\mathcal{ST}$ for all nodes based on available information.
– Send meta-data ($\mathcal{N}_n$, $\mathcal{N}_{\mathcal{A}_t}$, $\mathcal{N}_{\mathcal{Q}_T}$, $\mathcal{T}_r$, $\mathcal{N}_{\mathcal{RT}}$) to $\mathcal{N}_{id}$.

Status. Server will perform the following operation in meta-data:

– Continuously updates the $\mathcal{SGX}_{\mathrm{T}}$, queue and dequeue the winner nodes to keep a track.
– Broadcast the result to the network.

Protocol 1.2: PoQ Server side protocol.

To ensure TEE platforms exists, nodes generally require to register with the hardware manufacturer to set up RA services. For instance, Intel SGX RA service needs registration with Intel Attestation Service (IAS)[3]. During manufacturing, each processor of SGX is equipped with a key that is certified by Intel [1]. After successful verification, the server adds the node id, $\mathcal{N}_{id}$ of that node to the queue, Q as it arrives. Subsequently, the node determines which tier, $\mathcal{T}_i$ it belongs to and then calculates its $\mathcal{Q}_t$ for that particular $\mathcal{T}_i$ for that time being which is equal to the amount of time it can be executed for its first pass. If it remains in the waiting part of the Q and any node joins which belong to its $\mathcal{T}_i$ then it needs to recalculate the $\mathcal{Q}_t$ again based on available data. A new node can also be added at the end of the Q. While a node is executing and a new node joins

[3] Intel, "Software sealing policies – intel® software guard extensions developer guide," 2017. [Online]. Available: https://software.intel.com/en-us/documentation/sgx-developer-guide

PoQ Node

1. The node signs its public key and then sends a node registration request $Register(\mathcal{N}_{\mathcal{P}_k}, Sign(\mathcal{N}_{\mathcal{P}_k}))$ to $\mathcal{S}$. Upon successful registration, the node receives an acknowledgment with its $\mathcal{N}_{id}$ and $\mathcal{SGX}_{\max}$.
2. Initialized by the server, the node jointly executes the *Remote_Att*$(\mathcal{S}_{\mathcal{P}_k})$ (Protocol 1.1) with the server to ensures its identity and it is running on an Intel SGX enabled platform without tampering.
3. The node performs the following steps in order to participate into the PoQ protocol:
   (a) Generate a random $\mathcal{SGX}_t$ from $\mathcal{E}$ within a range of $[\mathcal{SGX}_{\min}, \mathcal{SGX}_{\max}]$.
   (b) Broadcast and request $Verify(\mathcal{SGX}_t)$ to $\mathcal{S}$, hence gets meta-data that states information about the existing nodes in the network.
   (c) Determine tier $\mathcal{T}_i$ it belongs to according to the following:
   $$\mathcal{T}_i = \left\lceil \mathcal{T}_r \frac{\mathcal{SGX}_t}{\mathcal{SGX}_{\max}} \right\rceil \tag{1}$$
   (d) Calculate the local $\mathcal{Q}_t$ using the following formula:
   $$\mathcal{Q}_t = \left\lceil \frac{\sum_{i=1}^{\mathcal{N}_n} \mathcal{RT}}{\mathcal{N}_n{}^2} \right\rceil \tag{2}$$
   where $\mathcal{N}_n$ is the number of active nodes in the specific $\mathcal{T}_i$.
   (e) Obtain $\mathcal{ST}$ using the information from $\mathcal{SGX}_{\mathrm{T}}$.
4. Once it gets $\mathcal{ST}$, $\mathcal{SGX}_t$ will be reduced for the calculated $\mathcal{Q}_t$.
   While Remaining Time $\mathcal{RT} \neq 0$, then:
   (a) Generate a new $\mathcal{Q}_t$ and determine the next $\mathcal{ST}$.
   otherwise:
      i. A new block is propagated and the local leadership count is incremented by one.
      ii. Broadcast the winning result to $\mathcal{S}$ and announce new block.
5. To rejoin the network, steps 2 to 4 are repeated.

Protocol 1.3: Individual node side protocol.

that belongs to the same $\mathcal{T}_i$ it won't affect the $\mathcal{Q}_t$ of the executing node at that moment. However, if the node is unable to finish the entire $\mathcal{SGX}_t$ during that pass of $\mathcal{Q}_t$, it will be popped up from the Q and added again at the end of it without changing its $\mathcal{T}_i$ but this time it needs to recalculate the $\mathcal{Q}_t$ for its next pass. Then, the node who is in the starting point (starting time, $\mathcal{ST}$ = current time) of the Q will get the chance to reduce its $\mathcal{SGX}_t$. After completion of each node's $\mathcal{Q}_t$, the remaining $\mathcal{SGX}_t$ of the currently executing node is checked. A function "Time Left" keeps track of the *Remaining Time* ($\mathcal{RT}$) over $\mathcal{SGX}_t$ after each pass and once it has zero as its value, it will broadcast the result for claiming the leadership. A participating node is required to finish all its $\mathcal{SGX}_t$ to become

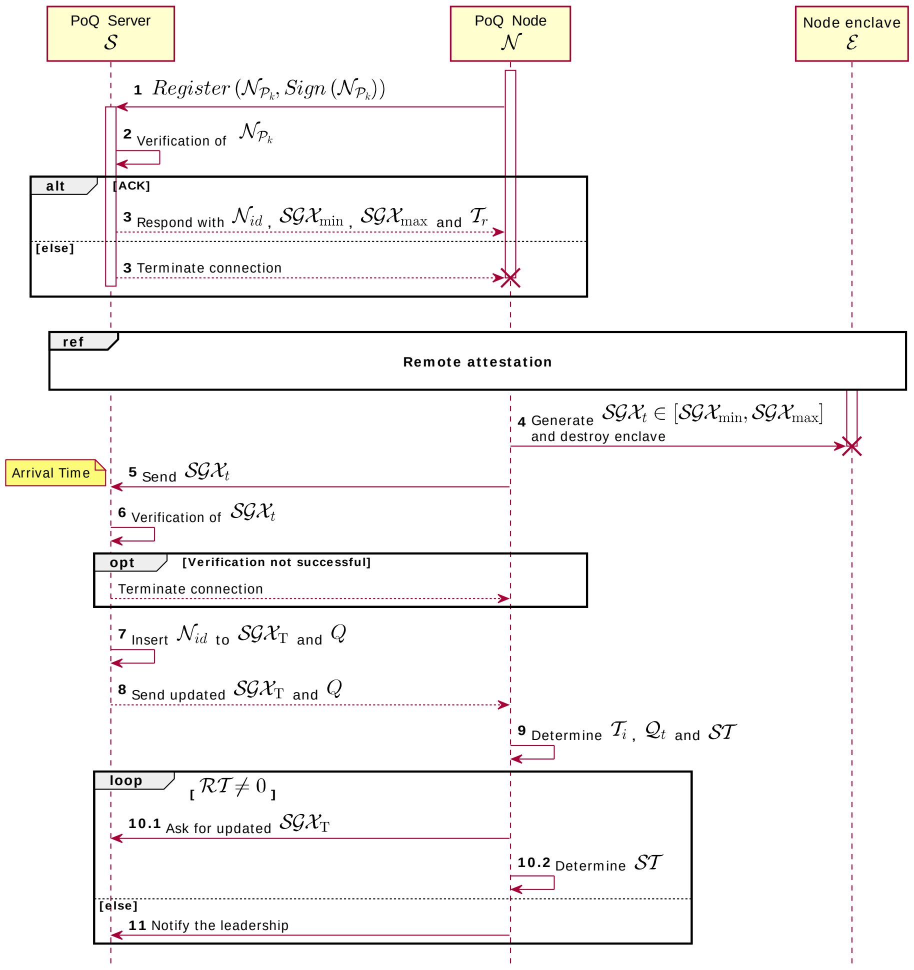

PoQ Server $\mathcal{S}$
PoQ Node $\mathcal{N}$
Node enclave $\mathcal{E}$
1 $Register\,(\mathcal{N}_{\mathcal{P}_k}, Sign\,(\mathcal{N}_{\mathcal{P}_k}))$
2 Verification of $\mathcal{N}_{\mathcal{P}_k}$
alt [ACK]
3 Respond with $\mathcal{N}_{id}$, $\mathcal{SGX}_{\min}$, $\mathcal{SGX}_{\max}$ and $\mathcal{T}_r$
[else]
3 Terminate connection
ref
Remote attestation
4 Generate $\mathcal{SGX}_t \in [\mathcal{SGX}_{\min}, \mathcal{SGX}_{\max}]$ and destroy enclave
Arrival Time
5 Send $\mathcal{SGX}_t$
6 Verification of $\mathcal{SGX}_t$
opt [Verification not successful]
Terminate connection
7 Insert $\mathcal{N}_{id}$ to $\mathcal{SGX}_{\mathrm{T}}$ and $\mathcal{Q}$
8 Send updated $\mathcal{SGX}_{\mathrm{T}}$ and $\mathcal{Q}$
9 Determine $\mathcal{T}_i$, $\mathcal{Q}_t$ and $\mathcal{ST}$
loop [$\mathcal{RT} \neq 0$]
10.1 Ask for updated $\mathcal{SGX}_{\mathrm{T}}$
10.2 Determine $\mathcal{ST}$
[else]
11 Notify the leadership

Fig. 4: Interactions corresponding to client-server communication in PoQ.

the leader and propagates a new block. It is worth mentioning that, the total waiting time of a node is not equal to its $\mathcal{SGX}_t$.

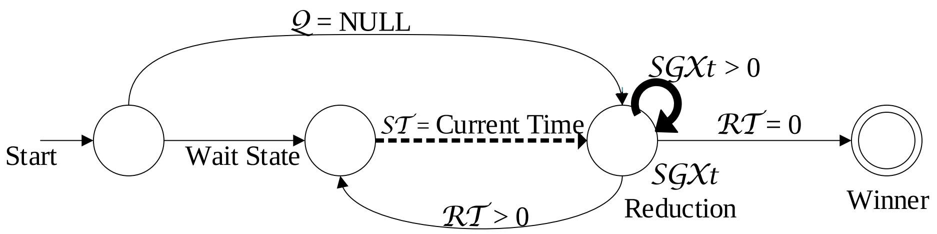


Fig. 5: The various state of a node before becoming a leader.

4.2 Principals:

Our protocol consists of two phases: server and client. The initial step is registration. In this phase, nodes need to join the network for authentication. Nodes are the major principals in the PoQ consensus protocol and rely on TEE. TEE can generate an independent identical random digit, which cannot be controlled by an advisory. RDRAND command is available in Intel SGX. Our design uses minimal energy consumption and exploits the Intel SGX floor.

4.3 Protocols:

In the initial phase, Intel SGX plays a crucial part. Our protocol develops an information flow between a local SGX and the server proposed in protocol 1 and 2.

Protocol 1.2 - Server side protocol. The server will always wait for a join request from nodes, $\mathcal{N}$. Whenever it gets a request, it calls $Register(\mathcal{N}_{\mathcal{P}_k}, Sign(\mathcal{N}_{\mathcal{P}_k}))$ method where it verifies the authentication with the directory of permitted public keys, $\mathcal{N}_{\mathcal{P}_k}$. If it is valid, the server generates an identification number, $\mathcal{N}_{id}$, inserts it into the Q and sends an acknowledgment to the newly joined node with the range of the $\mathcal{SGX}_t$ and $\mathcal{N}_{id}$. Immediately after receiving $\mathcal{SGX}_t$ from an node, $\mathcal{N}_1'$, it calls the $Verify(\mathcal{SGX}_t)$ method where it checks the $\mathcal{SGX}_{t1}'$. If it does not fit in the range, it aborts the connection. Thus, no node can go after the smallest number that is beyond the range to generate too many blocks. After successful verification of $\mathcal{SGX}_{t1}'$, the server stores the time when it's submitted and treats it as its arrival time, $\mathcal{A}_{t1}'$ of $\mathcal{N}_1'$. Then calculates $\mathcal{Q}_{t1}'$ according to its tier $\mathcal{T}_i$ and obtains the first starting time, $\mathcal{ST}_1'$, of that particular node $\mathcal{N}_1'$. Then the server builds an SGX Table, $\mathcal{SGX}_\mathrm{T}$ based on available data that contains the number of nodes, $\mathcal{N}_n$, arrival time $\mathcal{N}_{\mathcal{A}_t}$, quantum time, $\mathcal{N}_{\mathfrak{Q}_T}$, and remaining $\mathcal{SGX}_t$ for all active nodes $\mathcal{N}_{\mathcal{RT}}$ with numbers of tiers.

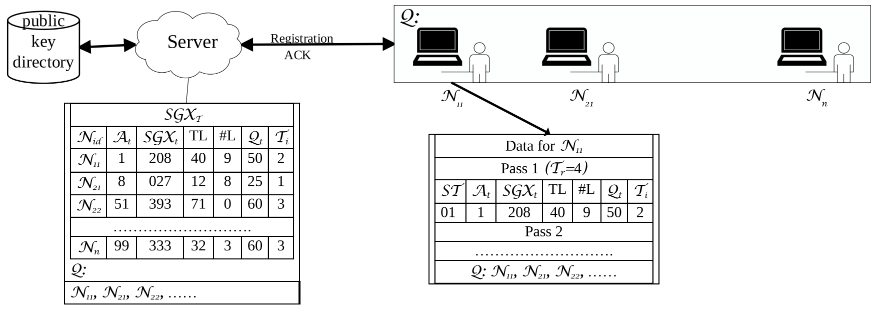
The server sends this meta-data to all active nodes. Thus, all $\mathcal{N}_{id}$ have access to a similar database concurrently that server has, and almost every data will replicate to all $\mathcal{N}$ which accelerates the speed of the network. The server always monitors the scheme of finding a new leader. When a new leadership has been claimed, a node $\mathcal{N}_1'$, is supposed to announce that it has completed its $\mathcal{SGX}_{t1}'$, so that it is considered as a leader and get appended to the blockchain and dequeued from the Q. If that particular node wants to rejoin in the network its $\mathcal{N}_{id}$ will remain the same with different credentials. Whenever a new node, $\mathcal{N}_2'$, join or a node $\mathcal{N}_1'$ leaves, the server continuously updates the Q based on all available data. However, the server can compute Average Elapsed Time, $\mathcal{AET}$ and Average Waiting Time, $\mathcal{AWT}$ by the following formulas:

$$\mathcal{AET} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{ET}_i \tag{3}$$

$$\mathcal{AWT} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{WT}_i \tag{4}$$

Definition 41 *Waiting Time.* The inactive time of a $\mathcal{N}_i$ after consider its $\mathcal{A}_{ti}$. Simply, the amount of idle time $\mathcal{WT}_i$ spent by a $\mathcal{N}_i$ in the Q before the last pass to finishes its $\mathcal{SGX}_{ti}$ for a single round can be calculated by Eq (6); where $\mathcal{A}_{ti}$ refers to $\mathcal{A}_t$ of $\mathcal{N}_i$ and $\mathcal{SGX}_{ti}$ refers to the time generated from $\mathcal{E}$ of $\mathcal{N}_i$. $\mathcal{AWT}$ is the average value of wait time of N nodes and can be calculated by Eq.(4).

Definition 42 *Elapsed Time.* The entire time requires a $\mathcal{N}_i$ to become a leader. That means the time elapsed between $\mathcal{A}_{ti}$ of a $\mathcal{N}_i$ and its termination. Elapsed time for a $\mathcal{N}_i$ can be calculated by Eq. (5); where $\mathcal{WT}_i$ refers to $\mathcal{WT}$ of $\mathcal{N}_i$ and $\mathcal{SGX}_{ti}$ refers to the time generated from $\mathcal{E}$ of that $\mathcal{N}_i$. The average elapsed time of N nodes $\mathcal{AET}$ can be calculated by using Eq. (3).



$\mathcal{SGX}_{\mathcal{T}}$						
$\mathcal{N}_{id}$	$\mathcal{A}_t$	$\mathcal{SGX}_t$	TL	#L	$\mathcal{Q}_t$	$\mathcal{T}_i$
$\mathcal{N}_{11}$	1	208	40	9	50	2
$\mathcal{N}_{21}$	8	027	12	8	25	1
$\mathcal{N}_{22}$	51	393	71	0	60	3
....						
$\mathcal{N}_n$	99	333	32	3	60	3

$\mathcal{Q}$:

$\mathcal{N}_{11}$, $\mathcal{N}_{21}$, $\mathcal{N}_{22}$, ……

Data for $\mathcal{N}_{11}$						
Pass 1 ($\mathcal{T}_r$=4)						
$\mathcal{ST}$	$\mathcal{A}_t$	$\mathcal{SGX}_t$	TL	#L	$\mathcal{Q}_t$	$\mathcal{T}_i$
01	1	208	40	9	50	2
Pass 2						
....						
$\mathcal{Q}$: $\mathcal{N}_{11}$, $\mathcal{N}_{21}$, $\mathcal{N}_{22}$, ……						

Fig. 6: Top level architectural diagram of the system.

Protocol 1.3 - Individual node side protocol. At first, a participant node needs to register for joining the network. After joining, a local SGX acts as a client node that requires to download the PoQ code and execute it. When a local SGX connects to the server, it gets $\mathcal{N}_{id}$ and the range to generate a random $\mathcal{SGX}_t$ (which is subject to change subsequently after each round) from the trusted code inside $\mathcal{E}$ and needs to submit it to the server for verification, which is done on the same platform. If there is more than one node ($\mathcal{N}_2'$, $\mathcal{N}_{21}''$) who produces $\mathcal{SGX}_t$ at the same time then they will be added into the Q as ascending order of $\mathcal{N}_{id}$. If Q is null then $\mathcal{N}_{id}$ is added at the front of Q. If Q is not null but the $\mathcal{RT}$ of the current executing node is zero then $\mathcal{N}_{id}$ is added in $FIFO$ manner, otherwise, $\mathcal{N}_{id}$ will insert into the Q after the $\mathcal{N}_{id}$ of the executing node. Later, the node gets the $\mathcal{SGX}_{\mathrm{T}}$ from the server that consists of its $\mathcal{Q}_t$, $\mathcal{ST}$ and $\mathcal{A}_t$ along with some meta-data ($\mathcal{N}_n$, $\mathcal{N}_{\mathcal{A}_t}$, $\mathcal{N}_{\mathcal{Q}_T}$, $\mathcal{T}_r$, $\mathcal{N}_{\mathcal{RT}}$). If we consider there are only three nodes ($\mathcal{N}_2'$, $\mathcal{N}_{21}''$, $\mathcal{N}_3'$) in the network where $\mathcal{N}_2'$, $\mathcal{N}_{21}''$ has same $\mathcal{A}_t$ and $\mathcal{N}_3'$ join after two units of time, then the start time of $\mathcal{N}_2'$ which is $\mathcal{ST}'(\mathcal{N}_2')$ immediately when it arrives and $\mathcal{ST}''(\mathcal{N}_{21}'')$ is after the amount of quantum time of $\mathcal{N}_2'$. The $\mathcal{ST}'(\mathcal{N}_3')$ is the addition of all the nodes $\mathcal{Q}_t$ left to execute, who are in front of it in the Q at the moment it arrives. Based on all available information, a particular node, $\mathcal{N}'$, can also calculate which $\mathcal{T}_i$ it belongs to, $\mathcal{Q}_t{}'$ of that specific $\mathcal{T}_i$ and $\mathcal{ST}'$. For a specific tier whoever comes first will start first. When a new $\mathcal{N}$ is added to a particular $\mathcal{T}_i$, the $\mathcal{Q}_t{}'$ of that $\mathcal{T}_i$ is recalculated according to the available updated information based on $\mathcal{RT}$ of all nodes in that $\mathcal{T}_i$. When $\mathcal{ST}$ is equivalent to the current time, nodes will execute to reduce its $\mathcal{RT}$. When a node joins, the amount of its $\mathcal{RT}$ is equivalent to its $\mathcal{SGX}_t$.

After a single pass, if a particular node, $\mathcal{N}_1'$, is not able to finish its $\mathcal{SGX}_{t1}'$ as a whole, $\mathcal{RT}_1'$ will be updated by deducting the time spent on that pass and it will put itself at the end of the updated queue, then calculates its next $\mathcal{ST}_1''$ and needs to wait for another pass. If there is no $\mathcal{N}$ in the overall Q than the current node may carry on. It is mentioned that, at any stage, for a particular $\mathcal{N}_1'$, if $\mathcal{SGX}_{t1}'$ or $\mathcal{RT}_n'$ is less than the $\mathcal{Q}_{t}{}'_{\mathcal{T}_i}$ then the $\mathcal{Q}_{t1}'$ will be updated and assigned to the equal portion of that specific $\mathcal{RT}_n'$. If any node finishes its $\mathcal{SGX}_t$, then it will be withdrawn from the Q and becomes the leader and the number of leadership is assigned to it will be increased by one. Thus, a new block is propagated. Figure 6 elucidates the top-level architecture of PoQ and figure 4 shows the inter-process communication between the nodes and server. However, the node can compute its Elapsed Time and Wait Time by the following formulas:

$$\mathcal{ET}_i = \mathcal{SGX}_{ti} + \mathcal{WT}_i \tag{5}$$

$$\mathcal{WT}_i = \text{EndTime}_i - (\mathcal{SGX}_{ti} + \mathcal{A}_{ti}) \tag{6}$$

5 Experimental Evaluation

5.1 Goals

The design of a good consensus protocol must satisfy the following goals: (i) backing up a large-scale network (ii) obtain a higher throughput, and (iii) achieve fairness. To better motivate and illustrate our design, we performs these experiments to achieve those goals throughout the experiments.

5.2 Setup

We built a prototype of PoQ to evaluate its performance. All practical experiments performed below were done using a system equipped with SGX PSW 2.X of version 2.5.100.2 and SGX SDK 2.X of version 2.5.100.2 which acts as a in-house client-server network. The system has Windows 10 OS with the latest updates, Intel® Core$^{\text{TM}}$ i7-7567U processor (3.5 GHz to 4.0 GHz Turbo, Dual Core 4 MB cache, 28W TDP), 32GB RAM, 64 Mb Flash EEPROM, and 34.1

GB/s Max Memory Bandwidth[4]. We assume that every full node is a potential validators.

5.3 Throughput

The purpose of this experiment is to measure and compare the throughput of PoQ and PoET. We ran multiple experiments with different parameters. We measure the number of leaderships per second for ten different nodes and three $\mathcal{SGX}_t$ ranges: [1,100], [1,500], and [1,1000]. In the baseline case, we assume that all nodes arrive approximately at the same time: within the first two (Figure 7.a), ten (Figure 7.c), and twenty (Figure 7.e) seconds . Then, we allow those ten nodes to join randomly at different times within a certain range of $\mathcal{A}_t$ with the same $\mathcal{T}_r$ which is 5. We ran each test 50 times where $\mathcal{SGX}_t$ and $\mathcal{A}_t$ were generated randomly. The time duration for each run for Figure 7.a and 7.b were 90 sec ($\mathcal{A}_t \in$ [0,300]s), 450 sec ($\mathcal{A}_t \in$[0,1500]s) for Figure 7.c and 7.d, and 900 sec ($\mathcal{A}_t \in$[0,3000]s) for Figure 7.e and 7.f. In Figure 7, the graph with different arrival times deals with the average result of tests where 20% of nodes leave the network randomly at any time after becoming a leader at least once. Note that the protocol was slightly modified when performed 20% nodes left. For baseline case, experiments were run with the same settings as discussed above and the final result is averaged where node does not leave the network. For comparison, we implement PoET and run the same experiments with the same attributes to evaluate the performance with respect to PoQ.

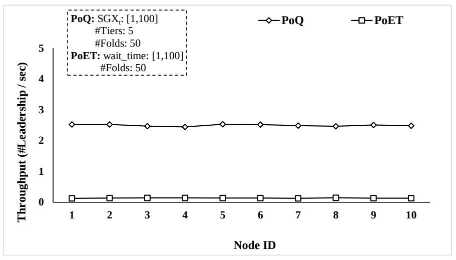
By comparing PoQ with PoET in Figure 7, we observe that the throughput of PoQ is higher in both cases: all nodes join approximately at the same time (baseline case), and when they join at different times. The difference between the two protocols' throughput could be as low as 0.3 (Figure 7.2) and as high as 3.5 (Figure 7.d).

5.4 Scalability

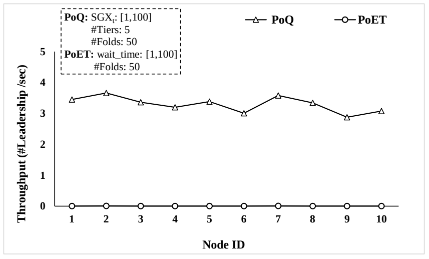
In this section, we evaluate the scalability of our protocol. We start with a network of 2000 nodes then double the network size 5 times, raising to 10,000 nodes in the last setting. In Figure 8 we keep the same parameters involved in Figure 7.a but with a larger number of nodes. We ran the experiment only once until all the nodes become exactly one leader. It should be noted that epoch time is longer (e.g., 100 seconds in 2000 nodes to 504 seconds in 10,000 nodes) since it requires relatively more times when the total number of nodes increases.

By observing the graph we conclude that $\mathcal{AWT}$, $\mathcal{AET}$ for different sizes of network increase linearly as the network size grows. We also measured RA which takes roughly 2ms and we did not consider it in result data.

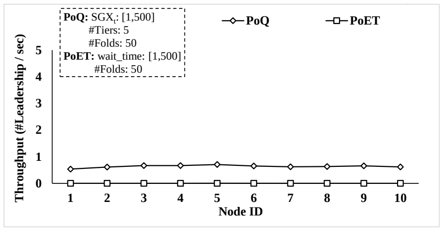
[4] Max Memory Bandwith is the maximum rate at which data can be read from or stored into a semiconductor memory by the processor (in GB/s).



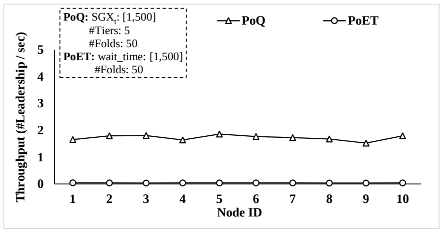
(a) Nodes start approximately at the same time, $\mathcal{A}_t \in \{0,2\}s$



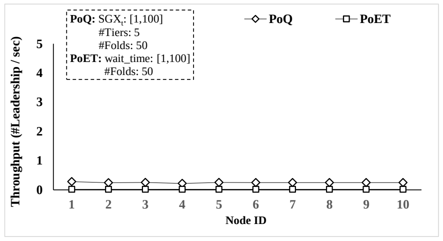
(b) Nodes start at different time, $\mathcal{A}_t \in \{0,300\}s$



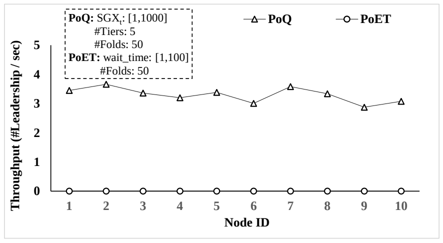
(c) Nodes start approximately at the same time, $\mathcal{A}_t \in \{0,10\}s$



(d) Nodes start at different time, $\mathcal{A}_t \in \{0,1500\}s$



(e) Nodes start approximately at the same time, $\mathcal{A}_t \in \{0,20\}s$



(f) Nodes start at different time, $\mathcal{A}_t \in \{0,3000\}s$

Fig. 7: Throughput evaluation results among ten nodes for PoQ and PoET. Each data point in our plots is averaged over 50 independents measurements.

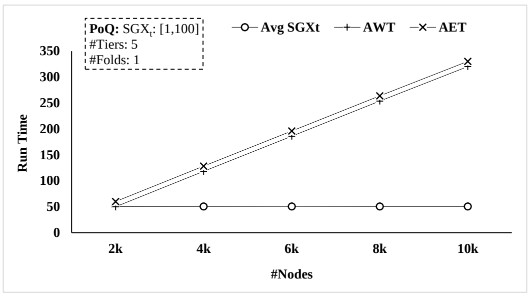


Fig. 8: Overview of bridging between $\mathcal{AWT}$ and $\mathcal{AET}$ in PoQ in response to scalability (up to ten thousands of nodes).

5.5 Fairness

A consensus protocol is fair if a miner/validator with p share of the overall resource ratio can produce a block with a probability p. In this section, we trying to measure the relation between the number of SGX machines a node has and the number of leadership it can reach. We conducted this experiment with the same parameter elaborated in Figure 7.a. We ran the experiment 50 times and the Figure 9 reported below are averaged over fifty independent runs. The graph shows the cumulative average leadership of a validator who has a certain number of SGX per node. The X-axis indicates the total number of SGX a node has and the Y-axis shows the average leadership.

After running our experiments, described above, we observe that the probability of being chosen to be a leader scaled linearly, in relation to the number of SGX machines per node.

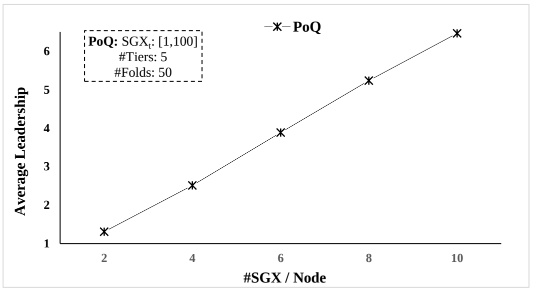


Fig. 9: A linear growth in experimentation over nodes.

6 Conclusions

In this paper, we proposed Proof of Queue PoQ, a consensus protocol for private (permissioned) blockchains based on Intel SGX's trusted execution environment. PoQ is specifically designed to achieve fairness when determining the leader to mine the next block. Our protocol is suitable for a large number of nodes with an enormous wait time. The design of PoQ shows that it maintains approximately similar wait times for all the nodes without any collision. Finally, based on the simulation and large-scale evaluation of PoQ, we showed that no nodes is left behind, and it achieves better scalability, throughput, and fairness over PoET. As a future work, we plan to modify the node-side protocol by adjusting the quantum time depending on the time left of executing nodes in order to optimize the overall fairness of the protocol.

References

1. Ahmed, M., Kostiainen, K.: Identity aging: Efficient blockchain consensus (2018)

2. Andreina, S., Bohli, J.M., Karame, G.O., Li, W., Marson, G.A.: Pots - a secure proof of tee-stake for permissionless blockchains. IACR Cryptology ePrint Archive **2018**, 1135 (2018)
3. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the thirteenth EuroSys conference. pp. 1–15 (2018)
4. Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., Danezis, G.: Sok: Consensus in the age of blockchains (2017)
5. Bashar, G.D., Hill, G., Singha, S., Marella, P.B., Dagher, G.G., Xiao, J.: Contextualizing consensus protocols in blockchain: A short survey. 2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA) pp. 190–195 (2019)
6. Bentov, I., Lee, C., Mizrahi, A., Rosenfeld, M.: Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract] y. ACM SIGMETRICS Performance Evaluation Review **42**(3), 34–37 (2014)
7. Buterin, V., et al.: Ethereum: A next-generation smart contract and decentralized application platform **7** (2014)
8. Chen, L., Xu, L., Shah, N., Gao, Z., Lu, Y., Shi, W.: On security analysis of proof-of-elapsed-time (poet). In: SSS (2017)
9. Chen, X., Zhao, S.: Scalable, efficient, and consistent consensus for blockchains (2018)
10. Costan, V., Devadas, S.: Intel sgx explained. IACR Cryptology ePrint Archive **2016**(086), 1–118 (2016)
11. Dang, H., Dinh, A., Chang, E.C., Ooi, B.C.: Chain of trust: Can trusted hardware help scaling blockchains? ArXiv **abs/1804.00399** (2018)
12. Dang, H., Dinh, T.T.A., Loghin, D., Chang, E.C., Lin, Q., Ooi, B.C.: Towards scaling blockchain systems via sharding. In: Proceedings of the 2019 International Conference on Management of Data. pp. 123–140. SIGMOD '19, ACM, New York, NY, USA (2019)
13. Dhillon, V., Metcalf, D., Hooper, M.: The hyperledger project. In: Blockchain enabled applications, pp. 139–149. Springer (2017)
14. Dib, O., Brousmiche, K.L., Durand, A., Thea, E., Hamida, E.B.: Consortium blockchains: Overview, applications and challenges. International Journal On Advances in Telecommunications **11**(1&2) (2018)
15. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Annual Cryptology Conference. pp. 585–605. Springer (2015)
16. Johnson, S., Scarlata, V., Rozas, C., Brickell, E., Mckeen, F.: Intel® software guard extensions: Epid provisioning and attestation services. White Paper **1**, 1–10 (2016)
17. Milutinovic, M., He, W., Wu, H., Kanwal, M.: Proof of luck: An efficient blockchain consensus protocol. In: Proceedings of the 1st Workshop on System Software for Trusted Execution. SysTEX '16, ACM (2016)
18. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
19. Saleh, F.: Blockchain without waste: Proof-of-stake (2018)
20. Schwartz, D., Youngs, N., Britto, A., et al.: The ripple protocol consensus algorithm. Ripple Labs Inc White Paper **5**(8) (2014)
21. Wahab, A., Mehmood, W.: Survey of consensus protocols (2018)
22. Xiao, Y., Zhang, N., Lou, W., Hou, Y.T.: A survey of distributed consensus protocols for blockchain networks (2019)

23. Zhang, F., Eyal, I., Escriva, R., Juels, A., Renesse, R.V.: REM: Resource-efficient mining for blockchains. In: 26th USENIX Security Symposium (USENIX Security 17). USENIX Association, Vancouver (Aug 2017)

Appendix

Abbreviations
The following abbreviations are used in this manuscript:

Symbol	Description
Q	Queue
$\mathcal{N}$	SGX Node
$\mathcal{E}$	SGX Node enclave
$\mathcal{S}$	SGX Server
$\mathcal{SGX}_t$	SGX time
$\mathcal{SGX}_{\min}$	Minimum value of $\mathcal{SGX}_t$
$\mathcal{SGX}_{\max}$	Maximum value of $\mathcal{SGX}_t$
$\mathcal{SGX}_{\mathrm{T}}$	SGX Table
$\mathcal{N}_n$	Number of active nodes in a specific tier
$\mathcal{N}_{id}$	Node id generated by the SGX Server
$\mathcal{T}_i$	Tier id for i-th node
$\mathcal{T}_r$	Total number of tiers available. This value is defined by the SGX Server and is uniformly distributed
${\mathcal{A}_t}^{(i)}$	Arrival Time of i-th node
$\boldsymbol{\mathcal{A}}_t$	Arrival Times from all nodes
$\mathcal{ET}$	Elapsed Time
EndTime	End Time
$\mathcal{ST}$	Starting Time
$\mathcal{RT}$	Remaining Time
$\mathcal{AET}$	Average Elapsed Time
$\mathcal{WT}$	Wait Time
$\mathcal{AWT}$	Average Wait Time
σ	Standard deviation
$\mathcal{Q}_t$	Quantum Time
$\boldsymbol{\mathcal{Q}}_T$	Quantum Time of all nodes
$\mathcal{P}_k$	Public key
$\mathcal{S}_k$	Private key
$\mathcal{N}_{\mathcal{P}_k}$	Node public key
$\mathcal{N}_{\mathcal{S}_k}$	Node private (secret) key

Table 3: Summary of notation used throughout this paper