On Elapsed Time Consensus Protocols

Mic Bowman
Intel Labs
mic.bowman@intel.com

Debajyoti Das
Purdue University
das48@purdue.edu

Avradip Mandal
Fujitsu Laboratories of America
amandal@us.fujitsu.com

Hart Montgomery
Fujitsu Laboratories of America
hhmontgomery@us.fujitsu.com

Abstract

Proof of Elapsed Time (PoET) is a Nakamoto-style consensus algorithm where proof of work is replaced by a wait time randomly generated by a trusted execution environment (TEE). PoET was originally developed by Intel engineers and contributed to Hyperledger Sawtooth, but has never been formally defined or analyzed. In particular, PoET enables consensus on a bitcoin-like scale without having to resort to mining. *Proof of Luck* (PoL), designed by Milutinovic et. al., is a similar (but not identical) protocol that also builds a Nakamoto-style consensus algorithm using a TEE. Like PoET, it also lacks a formal proof.

In this work, we formally define a simplified version of PoET and Proof of Luck, which we call *elapsed time (ET) consensus* with a trusted timer. We prove the security of our ET consensus protocol with a trusted gimer given an honest majority assumption in a model very similar to the bitcoin backbone model proposed by Garay et al. which we call the *elapsed time backbone model*. Our model and protocol aims to capture the essence of PoeT and PoL while ignoring some of the more practical difficulties associated with such protocols, such as bootstrapping and setting up the TEE.

The PoET protocol also contains a function called the z-test that limits the number of blocks a player can publish in any particular larger set of blocks. Surprisingly, by improving this z-test a little bit we can prove the security of our ET consensus protocol *without* any TEEs with a (slightly stronger) honest majority assumption. This implies that Nakamoto-style consensus with rate limiting and no proofs of work can be used to obtained scalable consensus in a permissioned setting: in other words, "bitcoin without proofs of work" can be made secure without a TEE for private blockchains!

1 Introduction

In today's interconnected world, it is important to be able to share data widely but in a selective manner. Efficient distributed databases have been known for quite some time and continue to get better and better [VO99]. However, basic distributed databases have a core problem: they have absolutely no protection from malicous users. Since we do not live in a perfect world, we cannot expect database users to be angels [Mad88], and this leads to many practical issues when using distributed databases: what happens when two people or entities that do not trust each other need to share data? What if these participants in the database are outright malicious?

Almost two decades ago, Castro and Liskov [CL+99] came up with a clever solution to this problem: the practical byzantine fault tolerant (PBFT) consensus algorithm. PBFT was a clever and practical invention: it allowed people to use what were essentially distributed databases that tolerated up to a

third of the users being malicious. This was a big improvement over basic distributed databases, but still did not allow for truly public databases. In addition, PBFT protocols require a large amount of communication between participants–$O\left(n^2\right)$, for n parties [DR85], which makes them very difficult to scale. So while PBFT protocols proved to be very useful for many applications of distributed computing, they did not fully solve the fundamental problem at hand.

In 2008, another new technology radically changed the state of distributed databases: bitcoin [Nak]. Someone using the pseudonym Satoshi Nakamoto designed what amounted to a new distributed database[1] with some pretty incredible properties: the database is fully public, so anyone can participate, and (probabilistic) consensus in the optimistic case only requires $O\left(n\right)$ communication, meaning that it is easy for tens of thousands of users to participate in bitcoin at any given time. The ideas behind bitcoin have been further generalized: blockchains enabling smart contracts such as Ethereum constitute even more powerful types of distributed database.

But bitcoin and other proof of work-based systems have one major drawback: energy consumption. One source [dig] reports that the power consumed for bitcoin proof-of-work in January 2019 was around 40TWh/year, comparable to the power use of a small country. In essence, the public trust that bitcoin guarantees is directly correlated to the energy consumption of the bitcoin miners. To put it differently, bitcoin's resiliency to attack is a direct result of consuming large amounts of power.

This brings us to a fundamental question in modern distributed databases: can we build systems with many of the good core properties of bitcoin–scalability and broadly decentralized trust—without the drawbacks associated with mining?

1.1 Proof of Elapsed Time (PoET) Consensus

In an attempt to offer a low power but scalable alternative, Intel included in the Hyperledger Sawtooth [saw] distributed ledger platform a form of Nakamoto consensus that replaced proof-of-work with an alternative called *proof of elapsed time*, or *PoET* for short [poe]. A validator participating in a PoET consensus network creates a random "lottery ticket" that indicates a future time when it may claim the right to publish the next block in the blockchain (assuming no other validator has a ticket with an earlier time). Instead of backing the resiliency of the network with massive power consumption, PoET utilitizes the security properties provided by a trusted execution environment (TEE).

A trusted execution environment is a hardware-assisted, secure execution environment that protects computation from common attacks caused by applications, system administrators and potentially even the operating system [EKA13]. TEEs provide features that may include confidential execution, application integrity protection, and attestation. Examples include Intel SGX [Int13] and ARM TrustZone [tru09].

In other words, PoET functions very similarly to bitcoin, except for the fact that it uses TEEs to replace mining. Intuitively, the analogy is as follows: in bitcoin, miners have to actually compute proofs of work, and the first miner to produce and publish a valid proof of work on the head of a chain claims the next block of the chain. In PoET, the "miners" query a TEE. The TEE "tells" the "miners" when they would have finished mining, and offers a "proof" of this fact that can be verified by other "miners." The first "miner" to publish such a proof extending the head of a chain after its "wait time" is up claims the next block in the chain. While this analogy is not exactly correct, it provides the right intuition for how PoET works relative to bitcoin.

Other Proposals. There have been other proposals for TEE-based "bitcoin without mining" consensus protocols. To our knowledge, proof of luck [MHWK16] is the only academically proposed system. Intuitively, proof of luck works in a very similar way to PoET, although there are some key differences we will discuss later.

[1] We take the view in this paper that bitcoin is, functionally speaking, a "trustless" distributed database for currency accounting.

Production Systems and Implementations. PoET has been used in many production systems that run on Hyperledger Sawtooth [saw], which is a permissioned blockchain platform. These include things as wide-ranging as Salesforce's marketing platform [sal], the Tel Aviv Stock Exchange blockchain platform [tas], and many startups, such as Blockchain Technology Partners' Sextant [sex]. These production systems use PoET due to its good perfomance and scalability, and in spite of perceived TEE insecurities and the lack of a formal security proof.

Despite its lack of formal analysis, there have been several academic works studying the efficiency of the PoET protocol [SZH+19, Cor19], as well as many casual analyses across the internet. Generally, all of these works point to the efficiency of PoET and its strong performance in large systems. We defer to the above works for precise details on PoET's practical performance.

When compared to other consensus algorithms, there are a number of compelling reasons to use PoET: it has absolutely minimal communication (like bitcoin), does not require any complicated cryptography or have a complicated leader set election (like Algorand [GHM+17] or Dfinity [AMNR18]), does not need a randomness beacon (like Ethereum 2.0 [But18]), and can be ideal for IoT scenarios in which low power and low communication are required. On the other hand, there are some clear drawbacks: consensus is not as fast as some proof-of-stake systems, the chain can have forks (no instant finality), and, perhaps most of all, trusted hardware is recommended. But, as networks become larger, communication becomes more of a bottleneck, and more and more devices have trusted hardware, we think PoET and similar protocols will see more widespread use.

1.2 Our Contributions

We generalize the PoET and Proof of Luck protocols into what we call *elapsed time consensus*, and we prove the security of our elapsed time consensus protocols in a model that generalizes the bitcoin backbone model [GKL15] which we call the *elapsed time backbone model*. Elapsed time consensus can be thought of as a relaxation of the PoET protocol where we focus on the critical protocol itself and ignore some of the practical difficulties faced in practical implementations, such as bootstrapping, onboarding parties, and dynamic membership. Many of our proofs and models are inspired by those from the works of Garay et al. on the bitcoin backbone [GKL15, GKL17] as well as more recent follow-up works [BMTZ17].

We focus on elapsed time consensus protocols with two main assumptions on the TEE: 1. The TEE has access to a trusted timer.[2] 2. No TEEs are present, or TEEs can be easily compromised.

Elapsed Time Consensus Model. Our first main contribution is the development of a simple security model that we believe effectively captures the guarantees desired for a blockchain implementation in the permissioned model (participation in consensus is not fully public[3]). We define blockchain security in the form of a security game played between a challenger and adversary, mirroring traditional cryptographic security games. The adversary wins if they can cause the blockchain to have certain undesirable properties: for instance, breaking properties that correspond to safety, liveness, or the ability of honest parties to complete transactions. In section 3, we define our elapsed time consensus model.

In the bitcoin backbone paper [GKL15], the authors rigorously prove security but do it in an ad-hoc manner. We generalize their security proof into a model that explicitly formalizes a game that rigorously defines a protocol structure and adversary, and thus is potentially much more extensible to other blockchain proofs of security. Our model allows us to more easily handle things like a public key infrastructure and a TEE formalization, which would be cumbersome using the [GKL15] approach. Furthermore, the [GKL15] framework explicitly requires that the chance of any party "winning" and generating a block in any given round is identical. For one of our protocols[4], this will explicitly not be

[2]We will explain precisely what this means later, but it is not a fully synchronized "wall clock."

[3]For a reader unfamiliar with permissioned blockchains, we explain the concept rigorously later in this paper.

[4]For those looking forward: our elapsed time consensus protocol with z-test will allow an adversary to win specific blocks with probability one, clearly violating this guarantee

the case, so we do in fact need a generalization of [GKL15] in order to prove our protocols secure.

While our model does not meet the requirements for universally composable (UC) security [Can01] (and thus does not give ideal security guarantees), it is considerably simpler and easier to work with than UC proofs of bitcoin, like that of [BMTZ17]. However, we do believe in the merits of UC-framework blockchain constructions and consider building a UC version of our protocol to be an interesting line of further research.

Elapsed Time Consensus with a Trusted Timer. We next define the simplest version of our protocol: elapsed time consensus with a trusted timer. In this version of the protocol, we assume that the TEE has access to a trusted timer.[5] We also run (essentially) all code inside the TEE, resulting in some inefficiencies since a "mining node" must have an active connection (i.e. constantly query) the TEE to figure out when blocks are ready.[6]

This protocol captures the essence of PoET [poe] and other related works like proof of luck (which we discuss more later). We define the protocol and discuss our design decisions in section 4. While it may, at first glance, seem like TEEs make this proof trivial, it turns out that there are still many possible attacks. While we assume that an adversary cannot compromise any TEE, they still may cause Byzantine faults in communications between nodes. For instance, selfish mining attacks [ES18] are still possible, and our proof must take these into account.

In our first main proof, we show that our elapsed time consensus protocol with a trusted timer is secure with an honest majority[7] when the protocol is run on nodes such that each node is run inside a trusted execution environment that cannot be compromised. Due to its similarities to how bitcoin works, the proof of this system follows in a similar manner as similar proofs for bitcoin [GKL15]. Our proof for this protocol can also be thought of as a proof for the proof of luck system, modulo some setup assumptions.

PoET Analysis. We show that, unfortunately, PoET does not fall into our above analysis, and we cannot prove that it is secure without a rate-limiting function on block production called a z-test (we explain this in more detail below). The fundamental issue is that PoET gives players a "WaitCertificate" which tells them how long they have to wait. This information is not available to either a bitcoin miner or a "miner" in our ET consensus with trusted timer protocol, and we do not currently know how to prove security of such a protocol without a z-test. However, even without the z-test, we do not know how to use this additional information to attack the PoET protocol. So, the status of the PoET protocol without a z-test remains uncertain.

Elapsed Time Consensus with a z-test. As we mentioned before, it is well known by this point that the security of TEEs may not be perfect [KHF+19, VMW+18, LSG+18, KKSAG18]. In deference to this possibility,[8] the Intel engineers who developed the PoET protocol cleverly allowed for the use of what is called a z-test in order to mitigate potential TEE compromises. The z-test works as a rate-limiter: if a particular participant in the blockchain protocol produces "too many" blocks, then all of the other participants refuse to accept their blocks for a certain amount of time. The PoET protocol as described in the specification [poe] currently uses a simple z-test that is based on the ratio of the number of blocks a particular player has produced over a period of time to the total number of blocks produced by the entire blockchain over that same time period.

[5] We note that many TEEs, including SGX [pla], have a trusted timer or equivalent functionality.

[6] We explain this more in the technical overview and proof sections.

[7] The exact proportion of honest users needed is dependent on a large variety of parameters, including things like network latency. For reasonable choices of parameters, this honest majority condition will still allow a constant fraction of participants to be adversarial.

[8] PoET was designed before these attacks were known.

We modify our basic ET consensus protocol to include a z-test. However, our z-test is quite different than the one proposed by the Intel engineers, as theirs is not sufficient for our proof.[9] We instead use a "time-based"[10] z-test, which we explain thoroughly in section 6. To summarize, rather than check the proportion of the total number of blocks a player produces, we (essentially) restrict how many blocks a player can produce over a sliding window of time.

Somewhat surprisingly (at least to us), we can show that elapsed time consensus with our z-test is secure in our (permissioned) model **without TEEs** and assuming that some constant fraction of the participants are Byzantine. In other words, we show that "bitcoin without mining"–where participants sample a wait time (which can be thought of as the time it "would have taken" them to mine a block) and (if they are behaving honestly) release a block if they have not yet received a block and their time is up — coupled with some clever rate-limiting (the z-test) it is secure in a permissioned network without any hardware security guarantees or other extra assumptions.

We only need to assume a (slightly higher) constant fraction of participants in the protocol are honest. Some of our other parameters, unfortunately, are substantially worse, including the amount of time needed to be confident of block confirmation (blockchain finality), but the protocol is certainly practical (and can be used in conjunction with a TEE to get "the best of all worlds"). We go through this in detail in section 6. This proves that the core protocol of PoET (with some modfications to the z-test) is secure, even if TEEs are compromised, although performance is worse in this case.

Implications. PoET, proof of luck, and other similar "elapsed time" based consensus protocols might have many practical applications, since they, like bitcoin, have (almost) minimal communcation cost: if there are no (or few) forks, communication is $O(n)$ where n is the number of participants. Traditional BFT-based solutions typically require $O(n^2)$ communication. While there are some recent and exciting BFT protocols that scale linearly in the number of participants [YMR+19], none of these seem to have the raw scalability of "bitcoin without mining" protocols. Thus, elapsed time consensus protocols seem quite useful for any application of a blockchain where consensus is permissioned and communication complexity is the limiting factor.

Until now, PoET, proof of luck, and other elapsed time based consensus systems have never (to our knowledge) been formally proved secure. Modifying these protocols so that they are admissible under our security proof (including the z-test) would mean that they are provably resilient to TEE compromise (although security guarantees wouldn't be as strong in the case of TEE compromise).

Perhaps the most exciting implication, though, is that we can ignore TEEs completely in these protocols and still maintain relatively good security with our z-test. This notion of "permissioned bitcoin without mining" might be very useful for blockchains in the future.

1.3 Related Work

The only current work of which we are aware on the security of PoET is [CXS+17]. This work shows how an adversary that is capable of compromising SGX can attack PoET up to the bounds of the z-test that PoET currently uses. Unfortunately, this paper does not offer any formal analysis in the other direction: it does not include a rigorous security proof that PoET is secure outside of these bounds. As we have mentioned before, we note that Milutinovic et al. [MHWK16] define a consensus protocol called *proof of luck* that functions very similarly to PoET, but do not offer a security analysis to their protocol or even a comparison to PoET. Additionally, the authors of [ABK+18] show a construction of a proof of stake protocol using TEEs, but this protocol also lacks a formal security proof.

[9] Using the z-test given by the PoET specification would require much stronger parameter settings than for which we achieve provable security

[10] For a reader familiar with distributed systems literature, our protocol is actually round-based rather than time based, but, as usual, thinking about round-based restrictions as time-based restrictions is a useful abstraction.

Improved consensus algorithms and models with exciting new properties have proliferated recently. Some examples include Thunderella [PS18], the sleepy model of consensus [PS17b], Snow White [DPS16], Fruitchain [PS17a], Ouroboros [KRDO17, DGKR18], Bitcoin-NG [EGSVR16], Casper [BG17], Stellar [Maz15], and ByzCoin [KJG+16]. Exciting new work in the space includes things like proofs of space and storage [BG18, CP19, DFKP15] and verifiable delay functions [BBBF18]. However, most of these protocols focus on public blockchains.

Comparatively, the adacemically-focused work on permissioned blockchains has been substantially less, but has notably included things like an analysis of Hyperledger Fabric [Cac16, ABB+18] and the Tendermint consensus protocol [Kwo14]. Work on Byzantine fault tolerant consensus has also been done [BSA14], including the recent an exciting development of [YMR+19].

There have also been a number of very useful papers that have analyzed these consensus protocols and their properties, including [CV17], [BGM16], [Vuk15], [GKW+16], [NKMS16].

Several consensus protocols in past have leveraged different forms of trusted hardware. For example, MinBFT [SVCNB+13] proposes a trusted counter to reduce the number of nodes required from $3F + 1$ to $2F + 1$ for F faulty nodes. More recently, FastBFT [LLKA19] uses a trusted execution environment to aggregate messages for latency and throughput improvements. Other protocols use TEEs for the purposes of sharding on blockchains [DDL+19].

1.4 Paper Outline

The rest of the paper proceeds as follows: In Section 2, we discuss background material that are relevant to this paper. In particular, we discuss some properties of PoET that give intuition on why it is a nice consensus algorithm. In section 3, we define our model and security notions that we use in the rest of the paper and justify our choices behind our model. In section 4, we define our elapsed time consensus protocol using a TEE with a trusted timer, which is our abstraction of PoET. We discuss its security in our ET backbone model in section 5. In section 6, we add the z-test functionality to our protocol; and in Section 7 we discuss a proof showing that our ET consensus with z-test is secure in our model even without TEEs. Finally, since the parameter choices are very complicated, in section 8 we discuss our parameter choices for the ET consensus protocol, and implications of the security. In Section 9, we present a version of our elapsed time consensus protocol with z-test that provides some performance improvement. We offer concluding remarks in section 10.

2 Background

In this section, we start by explaining how the PoET protocol works, since it is the inspiration for our constructions. We then explain proof of luck and compare it to PoET. Afterwards, we discuss the bitcoin backbone model of [GKL15], which we base our elapsed time backbone model on.

2.1 PoET

We begin by describing a leader election algorithm designed for Nakamoto consensus, called "Proof of Elapsed Time" or PoET, that elects leaders through a simple computation in a trusted execution environment provided by commodity processors. PoET is based on the observation that proof-of-work, at its core, is a method for generating and enforcing random wait times — proof-of-work elects as leader the first participant to solve a cryptographic puzzle, which induces a time delay that is a random interval drawn from (roughly) an exponential distribution.

With PoET, participants use a trusted execution environment — such as Intel's Software Guard Extensions (SGX) [poe]–to generate and then enforce a random wait time without a cryptographic puzzle.

Where proof-of-work can verify correctness by checking the solution to the puzzle, PoET uses an attestation of correctness that is provided by the trusted execution environment; any participant can verify that the wait time was generated correctly and enforced. In this way PoET preserves the fairness and integrity of leader election provided by proof-of-work without high cost of electricity and custom hardware.

As with proof-of-work algorithms, at the beginning of each PoET consensus round, a validator builds a block that extends the chain it believes to be the correct one (note that the selection of a chain to extend is an implicit "vote" for the chain). Next, the validator performs a computation that determines a time duration it must wait before it may claim leadership and extend the chain with the block it constructed. If no other validator claims leadership before the computed time expires, the validator broadcasts its proposed block and a proof that it computed the time correctly. The chain is extended with the new block and the process begins again. Effectively, the future time that is computed represents a form of "lottery" where every validator is given a ticket and the validator with the smallest ticket wins the lottery.

Just like bitcoin, players in PoET construct chains of blocks, and the longest chain is considered to be the correct chain. However, instead of mining, players query a TEE, which gives a "wait time." This wait time indicates how many rounds a player must wait until they can produce a blog. So players basically do the following:

- When a new block is received, check to see if:
  - The block results in the creation of a longer valid chain.
  - The "wait time" specified in the block has elapsed–i.e., that enough time has passed since the block's creation for it to be valid.
- If yes to both of the above:
  - Replace the currently stored chain with the received chain (implied by the block).
  - Query the TEE with the relevant parameters for a new block and receive a wait time, a in the form of a $WaitCertificate$.
- If no: Do nothing.
- When a player's wait time is up:
  - Query the TEE again and reeive the new block with the appropriate parameters.
  - Broadcast the block.

Otherwise, the player just waits.

Intuitively, PoET meets the basic requirements for decentralized consensus:

• **Censorship Resistant:** The computation should distribute leader election across the broadest possible population of participants. Fairness ensures that no single validator can persistently control the transactions that may be committed; that is, fairness ensures censorship resistence. To achieve fairness, PoET requires every validator to generate a random number across the same distribution. The properties of the distribution used are captured in the leadership claim so that every other validator can verify that the leader "played fairly."

• **Simple to Verify Correctness:** Third-party attestation is one of the most valuable characteristics of modern trusted execution environments. The basic idea is that the trusted execution environment can create an attestation about the characteristics of a computation that anyone can validate. The current implementation of PoET that is included in the Hyperledger Sawtooth distributed ledger platform [poe] uses Intel SGX as the trusted execution environment and the Intel Attestation Service (IAS) [AGJS13] as the root of trust for third party attestation. The PoET enclave creates an attestation that is signed by IAS. Anyone can verify the signature. The attestation binds an ECDSA key to the PoET enclave that can be

used to sign leadership claims. As a result, any validator can, by verifying the ECDSA signature is bound to a valid PoET enclave, verify that the leadership claim came from a trusted execution environment.

• **Distributed Wait Times:** Nakamoto consensus is efficient when all validators accept a leader with a single broadcast. To achieve this, wait times must be distributed such that a validator that believes it has been elected leader can broadcast its claim across the network before any other validator believes it has been elected leader. Similar to other proof-of-work algorithms, the current implementation of PoET pulls wait times from an exponential distribution. One of the unique benefits of PoET is that we can easily choose alternative distributions with potentially more efficient distribution of wait times. We do not discuss this in this work, but it is an interesting topic for future research.

• **Adapt to Changing Populations**[11]**:** Adaptation to changing populations of validators is one of the main benefits of Nakamoto consensus over traditional Byzantine Fault Tolerance. Proof-of-work algorithms handle adaptation through a variable difficulty level. If blocks are published more frequently than a target interval (an indication that the population of validators has increased), then the difficulty level increases enough to slow down block publication. Similarly, PoET samples the random wait times generated for winning blocks and determines if the intervals are appropriate. If blocks are being published too quickly, the distribution from which each validator chooses its wait time is increases. This is done deterministically and the result of the computation is captured in the leadership claim and signed by the enclave.

(Lack of a) Formal Description of PoET. We do not present a formal description of PoET here because, to our knowledge, there is not one. The most extensive description of the algorithm is given in the form of code in the Hyperledger Sawtooth repository [poe] and lacks the formal vigor necessary for a rigorous security proof. Instead, we define our own simplified versions of PoET, called *elapsed time consensus*, in section 4. Our elapsed time consensus protocols ignore some of the more difficult aspects of PoET to incorporate into a security proof, including dynamic membership, bootstrapping, and many of the practical difficulties associated with using enclaves. However, our protocols do exactly model the "core" functionality of PoET and thus we feel captures the theoretical essence of the protocol[12].

2.2 Proof of Luck

Proof of luck [MHWK16] (PoL) is a consensus protocol that is very similar to PoET. It was developed independently from PoET (the paper doesn't refer to PoET at all, although it appeared after Hyperledger Sawtooth) but uses the same core principle of building a "bitcoin without mining" consensus protocol using a TEE. The proof of luck protocol is well written in [MHWK16], but the paper lacks a formal security proof.

The main difference between PoET and PoL is the extent to which the TEE is used. PoL utilizes a strategy of running essentially all of the protocol execution inside of the TEE, whereas PoET is designed to use the TEE as little as possible. Players in PoL do more or less the following:

- When a new block is received, send it to the TEE.
- The TEE:
  - Checks if the block results in the creation of a longer valid chain.
  - The "wait time" specified in the block has elapsed–i.e., that enough time has passed since the block's creation for it to be valid.
  - If yes to both of the above:

[11] Dynamic adaption will not be included in the proofs of correctness in this paper.
[12] In [GKL15] and [GKL17], the authors make similar assumptions for bitcoin.

    - The TEE replaces the currently stored chain with the received chain (implied by the block).
    - The TEE samples a new wait time.
  - If no: Just wait.

- When a player's wait time is up (the player doesn't know the wait time):
  - The TEE generates a new block with the appropriate parameters and sends it to the player.
  - The player broadcasts the block.

It's not exactly clear how the "wait" functionality in PoL would be implemented efficiently in practice given the fact that, at least in SGX, the trusted timer is located outside of the core TEE and must be queried by the TEE [pla]. An implementation of PoL would probably have to have the TEE query the trusted timer frequently or use some other clever way of approximating time. In any case, it likely that this kind of solution would be substantially less efficient in terms of CPU and TEE operations than the PoET solution, where the wait time is made public.[13]

Looking ahead, the first protocol we present–elapsed time consensus with a trusted timer–is very similar to, PoL. Thus, our proofs of security for it can be reasonably applied to PoL, modulo practical details (i.e. setup, bootstrapping the blockchain and public key interface, etc.).

2.3 Bitcoin Backbone Model

In a series of very influential works [GKL15, GKL17], Garay, Kiayias, and Leonardos developed what they termed the *bitcoin backbone model* and used it to argue that bitcoin (as a distributed consensus protocol and public ledger) was secure. Their model is based on the principles of the UC-model [Can01] and thus captures a very general notion of adversarial behavior. It is our opinion that this bitcoin backbone model (and its derivatives, e.g. [BMTZ17]) to this point is the best and most comprehensive model for blockchain security. The full justification of the model is out of scope for this work, but we encourage readers to refer to the excellent works [GKL15] and [GKL17] for in-depth explanation and justifcation of the model.

At a high level, the model focuses on three core properties: *common prefix*, *chain quality*, and *chain growth*. We summarize these below:

- **Common Prefix Property**: the common prefix property, at a high level, says that, while the chains of all honest parties might be different, they share some common subchain $\mathcal{C}'$ that is a prefix of their main chain $\mathcal{C}$ (in our notation, $\mathcal{C}' \preceq \mathcal{C}$. We require that this common prefix chain $\mathcal{C}'$ be no more than λ blocks shorter than $\mathcal{C}$ with all but negligible probability, for some λ that is essentially a security parameter. More informally, the common prefix property says that no forks above a certain length can occur between honest parties on the chain. This property corresponds to "safety" in traditional consensus protocols.

- **Chain Quality Property**: the chain quality property requires that, in any subchain $\mathcal{C}$ of the main chain $\mathcal{C}$ of some honest party, at least some fraction of the blocks in $\mathcal{C}'$ have been produced by honest parties. This provides censorship resistance and "fairness" in practice–i.e., any honest party will be able to have a transaction included in the blockchain with some small amount of waiting.

- **Chain Growth Property**: the chain growth property simply requires that blocks are produced and the chain is extended at some minimal rate over time. It is roughly analagous to liveness in the traditional distributed consensus setting.

[13] But keeping the wait time hidden will have serious and positive implications on the security proof!

In [GKL15], the authors show that a bitcoin-like protocol with these three properties allows for secure applications like byzantine fault tolerant consensus and a "public ledger"–the core application of bitcoin– to be built. The proof follows from what the authors refer to as the **honest majority assumption**. For bitcoin, this has colloquially meant that honest parties control a majority of the hashing power on the chain. However, in [GKL15], the authors recognize that this is not enough for security–for instance, selfish mining attacks [NKMS16] are a counterexample to this–and the honest majority assumption requires that the number of participants/"hash power" of the honest parties be at least some fraction of the total number of parties which is dependent on parameters including the proof of work difficulty and network latency.

2.4 About TEE

We state the following security theorem about TEEs, which we will use in our proofs in order to assume that TEEs cannot be compromised.

Theorem 1. *Let T be a trusted execution environment, or TEE. Let $P_1, ..., P_\ell$ be a sequence of programs with input and output pairs $(\mathcal{I}_1, \mathcal{O}_1)$. Any adversary that can observe $T.Run(P_1)$, ... , $T.Run(P_\ell)$ and an arbitrary number of calls to the other provided TEE functions cannot distinguish the behavior of the TEE T from a PPT simulator given $(\mathcal{I}_1, \mathcal{O}_1)$ and the results of the other TEE function calls.*

This is a standard TEE security theorem mimicking the complexity-theoretic statements of [BGI+01]. We can also use a weaker theorem that models the guarantees from the *sealed glass model*, as given in [TZL+16]. In this case, we allow an adversary to see a computation but not to change it. We note that this model allows many of the most practical known attacks on TEEs.

Theorem 2. *Let T be a trusted execution environement, or TEE. Let P_1 be a program that may or may not include calls to other TEE functions. No adversary with access to T may induce an error in the computation of $T.Run(P_1)$. Moreover, no adversary may learn T's secret values used for certification (signing) or pseudorandom number generation.*

Multi enclave TEEs:

In Section 3.3 we introduced a TEE model that does not support parallel execution of multiple secure enclaves. Even though it simplified our TEE abstraction and protocol description, real TEEs such as Intel SGX do support parallel execution of secure enclaves. This opens the possibility of probability amplification attacks, where an adversarial player can potentially run multiple enclaves in parallel. However, such attacks are thwarted by the use of monotonic counter. The *WaitForBlock* function in Figure 4.1 remembers the chain length of the last call in the monotonic counter and aborts unless the current chain length is strictly larger than the chain length in the previous call.

3 Elapsed Time Backbone Model

In this section, we explain the notation and model that we use in our paper and why we chose the model we ended up using. We base most of our notation on that of [GKL15], although we have made some changes for the sake of clarity. Several follow-up works [GKL17, BMTZ17] have also influenced our description here.

3.1 An Overview on Our Elapsed Time Backbone Model

In this paper, we create a new elapsed time consensus backbone model that is inspired by the bitcoin backbone model. Our model has a few notable differences from the bitcoin backbone model [GKL15],

but is similar in its overall approach and can be thought of as a general formalization of that model. We outline our backbone model and explain some intuition below. In section 4, we formally define our model.

Permissioned vs. Permissionless Blockchains. Perhaps the largest deviation between our elapsed time backbone and the bitcoin backbone protocols is related to the notion of a *permissioned* versus a *permissionless* blockchain. Our constructions assume a permissioned model. This means that all parties know in advance who is allowed to participate in the blockchain (or, at least in the consensus aspect of the blockchain), and that we assume a pre-existing public key infrastructure. In other words, we assume that all participants in a blockchain have the public keys of all other participants when the genesis block is formed.

As an example, note that Byzantine fault-tolerant algorithms require knowledge of the participants in advance in order to be secure, and thus can only be used for *permissioned* blockchains. On the other hand, proof of work consensus does not require participants to know the identities of other participants, and thus can be used in the *permissionless* setting.

Unlike our protocols which assume some level of knowledge of the legal participants on the chain (even if it comes from the TEE), bitcoin (and the backbone protocol) are permissionless. It is not required that everyone know everyone else's public key in advance (or even ever). This is an important quality for truly public blockchains, but it makes communication more complicated since it is not possible for participants to know who they are talking to during the protocol. Since our elapsed time consensus protocols assume a PKI in some form (including through a TEE), we can dramatically simplify the communication protocol present in the bitcoin backbone paper and assume reliable secure channels[14]. For instance, we can assume all messages are signed by the sender, and we have a protocol abstraction that simulates this. This is not a possible assumption when all parties in the protocol do not know each other's public keys.

Players. We will consider a fixed number of N players in our protocol. Ideally, we would like to be able to handle a dynamic amount of players in our protocol (and even dynamic corruptions), and there isn't seemingly any reason we cannot do this other than proof complexity. However, we leave this topic to future work.

Discretizing Time. As in [GKL15], we consider a round-based protocol. Our protocol proceeds as a series of rounds. Each round is essentially started and ended by the adversary, but with the restriction that all honest parties have performed all of the actions that they intended to do in that particular round.

The main reason for this is that it is very difficult to analyze time-based protocols like this and prove concrete facts. Minor clock synchronicity incidents can be difficult or impossible for analysis, as are cases where things happen right around a time cutoff. Thus, we abstract these things out with a model, and assume that, in practice, these temporal events cannot be efficiently exploited by an adversary. Moving our model more towards reality with regards to our round structure is excellent future work.

Adversarial Corruptions. We allow the adversary to adaptively corrupt some fraction of the N players. The exact fraction allowed will vary depending on the protocol and depends on the protocol we are analyzing and many other parameters of the system. We assume that once a player is corrupted, it is permanently corrupted and always counts towards the adversary's number of allowed corrupted players.

Public Key Interface. We assume a PKI of some sort in our model. In other words, we assume that all users know the public key of all of the other players. This would not be a reasonable assumption if we

[14]This isn't always a good assumption, but we think it is reasonable for a permissioned blockchain on today's networks and internet.

were analyzing, say, bitcoin. But since we are focused on permissioned blockchains, we think that this is an eminently reasonable assumption to make.

Reliable Messaging. In our protocol, all of the parties will need to send messages to other parties. As we mentioned earlier, we require that messages from honest parties always be delivered, although we allow an adversary to delay these messages up to some amount (or, more precisely, reorder messages within a given round). Since we are assuming a PKI, we may also assume that these messages are signed.

We think that this correctly models the reality of the protocol: if an adversary can completely cut off a party from communication, then we count that as an adaptive corruption. Otherwise, in the real world, if it is possible for a party to communicate with at least a little part of the world at large, then it is likely that they can reach their desired recipients. The Internet is powerful! However, an adversary may have much faster messaging than an honest user, which is reflected in our model.

Due to our reliable messaging and PKI assumptions, we may also assume that players know exactly who sent what message and that messages have not been tampered with by an adversary. This follows trivially from adding abstractions of signatures that we call *certifications* to each message.

Formal Security Game. In the bitcoin backbone model, the authors define an adversary with certain properties and structure their proof around this adversary, but do not have a generalizable formal security game that can be easily used to prove the security of other, related protocols. In this work, we formally define an elapsed time consensus security game that can, at least in theory, be used to show the security of many different types of blockchain consensus algorithms. We hope that our formalisms around this security game can be used by other protocols in the future.

Our model requires that protocols have two core functionalities: *Genesis* and *RunHonestPlayer*. *Genesis* is the blockchain setup protocol that creates the genesis block and any other common infrastructure needed by the blockchain. It is run once at the beginning of the protocol. *RunHonestPlayer* is the heart of the protocol: what players are supposed to do during every round. It is run once, per (honest) player, per round.

Round Structure. As we have previously mentioned, our protocol is round-based. Each round proceeds roughly as follows:

- The round starts.
- The players receive messages sent in the previous round and can do any necessary "precomputation" activity.
- The players can "mine" or query a TEE if it is part of the protocol.
- The players can *Broadcast* messages, sending them to all other players. The messages are not received until the next round though.

Network Latency and Adversarial Reordering. The round-based protocol also allows us to model (at least to a degree) network latency. We allow the adversary to control the order that the messages sent in round i are received in round $i + 1$. This models the adversary having a "network latency" advantage over other parties. We note that [GKL15] and virtually all other works based upon it make the same or a very similar assumption. This is incorporated into our round structure, but we omit it above to simplify our description.

Adversarial Corruptions. We also allow the adversary to adaptively corrupt up to a certain threshold of parties (this threshold is stated in the individual security guarantees for that particular protocol). The corruptions are declared in round i and then the nodes are corrupted in round $i+1$. This is to ensure that an adversary cannot corrupt "winning" nodes after they have won but before they have broadcast their blocks.

3.2 Notations

We start by defining some basic primitives. We use $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\kappa$ to represent a cryptographic hash function (modelled as a random oracle), where $\kappa \in \mathbb{N}$ is the security parameter. We assume that a blockchain has a fixed number n of players which participate in the protocol.

A *block* is any tuple of the form $B = \langle s, x, \pi \rangle$, where $s \in \{0,1\}^\kappa$ is the hash of the previous block, $x \in \{0,1\}^*$ is the `content` of the block and $\pi \in \{0,1\}^*$ is the proof of block validity. `validblock` is the predicate that takes a block B as input, checks validity of the content. We formally define this in Figure 1. We will use this Block.XXX notation for querying the properties of blocks in our protocol.

Block:
Block.s // Hash of previous block.
Block.x // Block content.
Block.π // The block proof of validity will depend on the scheme.

Chain $\mathcal{C} = (B_1, \cdots, B_\ell)$:
Chain.head // The head of the chain B_ℓ.

ValidBlock(Block b, Chain $\mathcal{C}$):
//Verify the content of b, return 1 if valid and 0 otherwise.

Figure 1: Block & Chain Structure

With this in mind, we can move to defining chains. A *blockchain* or *chain* is a sequence of *blocks* (refer to Figure 1). The last block of a blockchain $\mathcal{C}$ is called the *head* of $\mathcal{C}$ and is denoted by $\mathcal{C}.head$. For an empty string or empty blockchain ε, we have $\varepsilon.head = \varepsilon$. The *length* of a chain is its number of blocks. A chain $\mathcal{C}$ can be extended by a block $B = \langle s, x, \pi \rangle$ if $s = \mathcal{H}(\mathcal{C}.head)$, $\texttt{ValidBlock}(B, \mathcal{C})$ is `true` and $B.\pi$ is a valid proof for the extended chain $\mathcal{C}' = \mathcal{C} \| B$. For the new chain $\mathcal{C}'$, we have $\mathcal{C}'.head = B$. For any chain $\mathcal{C} = (B_1, \cdots, B_\ell)$, $Length(\mathcal{C}) = \ell$ denotes the length of the chain. For any pair of integers $1 \leq i \leq j \leq \ell$, the chain $\tilde{\mathcal{C}} = (B_i, \cdots, B_j)$ is called a subchain of $\mathcal{C}$. $\mathcal{C}^{\lceil k}$ denotes the chain resulting from removing the k rightmost blocks, for a given non-negative integer k. If $k \geq \ell$, then $\mathcal{C}^{\lceil k} = \epsilon$. For two chains $\mathcal{C}_1, \mathcal{C}_2$ the notation $\mathcal{C}_1 \preceq \mathcal{C}_2$ denotes that $\mathcal{C}_1$ is a prefix of $\mathcal{C}_2$.

3.3 Abstractions

In this work we use the following abstractions to simplify our protocol and proof. These are generally slight modifications of standard abstractions that have been used repeatedly in the blockchain literature.

Certifications. Certifications are attestations of the statement being certified from the entity performing the certification. In other words, they act as completely unforgeable, perfect digital signatures. We use the *Cert* functionality instead of explicitly using digital signatures because it dramatically simplifies our presentation and proof while still capturing the essence of digital signatures.

$\mathbf{Cert_u}$ (Statement m):
return $Cert_{u,m}$ //certificate issued by u attesting validity of statement m.

$\mathbf{IsCertValid}_u$ ($Cert$ $c_{u,m}$, Statement m):
Return 1 if $c_{u,m}$ certifies statement m on behalf of user u and 0 otherwise.

Figure 2: Certification Functionality:

Trusted Execution Environment(TEE). In our model, a TEE is an unbreakable black box (a VBB obfuscator [BGI+01]) that, given an input, runs some code in a way that completely hides the internals of the code and then produces an output. An adversary can only see the input and output values of the program being run by the TEE. We note that this is an idealistic model for a TEE since it presumes perfect security and no side channel attacks. We use the $TEE.$ prefix to signify that a computation is done inside a TEE. In other words, everything that happens within a function called with the $TEE.$ prefix is hidden from all other users, who only see the final output.

In some protocols, we will give our TEE a trusted timer functionality. Since time is approximated by round number in our protocol, we will have the TEE return the current round number for this function.

Additionally, we endow our TEE with a monotonic counter $TEE.Counter$ which can never be decreased in value, *even if the TEE is reset*. This is the only non-permanent state that must be preserved in the TEE. We also allow our TEE to run programs securely, but these programs cannot preserve state outside of their runs with the exception of the monotonic counter. We note that many modern TEEs like Intel's SGX have both trusted timer and monotonic counter functionalities [pla].

Queues. A *queue* contains an ordered sets of certified (i.e. authenticated) chains. We allow, whenever a player or the adversary is running:

- *All parties* to add a certified chain to any other party's queue.
- *Players* may read from and remove items from *their own* queues.
- *The adversary* may examine all players' queues.

Later in our protocol, we allow the adversary to implicitly modify the queue (by reordering). But this is done in a controlled fashion by the protocol, and, in general, the adversary is not allowed to modify queues on their own.

Broadcast Communication. We give all of the participants in our protocol and the adversary the ability to broadcast messages to all other parties participating in the protocol. We assume reliable broadcast: a message that is sent always arrives at its destination — the adversary cannot interfere with message delivery.

Our broadcast protocol involves an array of n *queues* (recall that n is the total number of players) called $PLAYER\_QUEUE$, where $PLAYER\_QUEUE[i]$ corresponds to the incoming message queue for player i.

Corruptions. In our model, we allow the adversary to adaptively corrupt honest parties. When an adversary corrupts an honest party, we give them full access to all of the internal state of the honest party and access to all of the subroutines that the honest party could call.

3.4 Model and Structure

We now formally define our model. We include the presence of an adversary, who is given substantial power in the system. In particular, we allow the adversary to see all of the *global variables* in the system.

TEE.$\alpha := null$**;** // Can be set to any program code.
TEE.$args := null$**;** // Arguments for $\alpha()$.
TEE.$Counter := 0$**;** // monotonic counter inside TEE.

TEE.GetTime():
return current-round. // round number is our proxy for time

TEE.CounterSet(x)
// Verify that the value is higher than the current value
if $TEE.Counter < x$ **then** $TEE.Counter \leftarrow x$ **end if**

TEE.GetCounterValue():
Return TEE.Counter // Just return the counter value.

TEE.Cert$_{TEE}(m)$**:** return **Cert**$_{TEE}(m)$
// Just a digital signature abstraction with TEE guarantees.

TEE.Run$(Prog, arguments)$**:**
// Run the program *Prog* inside the TEE.
//If there is any currently running program it aborts the current program[a]
Abort(α), $\alpha \leftarrow Prog$, $args \leftarrow arguments$, Run $\alpha(args)$

TEE.Poll():
// Checks to see if a program is still running in the TEE
// Can be called only after calling TEE.Run()
if $\alpha = null$ **then** return $\perp$ **end if**
if $\alpha()$ has completed running **then**
// Return a CERT of the program and output
$O \leftarrow$ output of $\alpha(args)$, **Return** $(O, \textbf{Cert}_{TEE}(O||\alpha||args))$
else return *Incomplete* **end if**

[a]Our model of TEE handles only single secure enclave running at a time. Please refer to Section 2.4 for a discussion related to multi enclave TEEs.

Figure 3: TEE Functionality:

PLAYER_QUEUE [i] .ADD (Chain $\mathcal{C}$**):**
$PLAYER\_QUEUE[i].ADD(\mathcal{C})$ //Add chain $\mathcal{C}$ to player i's queue of chains

Figure 4: Queue Functionality

In our model we have three top level parameters: the total number of players n, the security parameter λ and honest parties' advantage δ. If t is the number of corrupt players then we require that $\frac{t}{n-t} \leq 1 - \delta$. We note that the adversary also has access to the description of all of the algorithms in the protocol, which includes things like the distribution from which wait times are sampled.

Model Rules. We let each honest party send to other players' queues, but only read from their own queue. All honest parties are allowed access to the round number as well. Honest parties are not allowed to see the $PLAYER\_CORRUPT$ array: only the adversary is allowed to view the contents of this. In

Broadcast (**Chain** $\mathcal{C}$)**:**
// Send the chain $\mathcal{C}$ to all parties.
for $i = 1$; $i \leq n$; $i++$ **do**
$PLAYER\_QUEUE[i].ADD(\mathcal{C})$
end for

Figure 5: Broadcast Functionality:

addition, the adversary is not allowed to see the internal state of honest parties.

Main Game Structure. We start by presenting the structure of our game in figure 6. It calls several subroutines, which we describe below.

RunProtocol(Number of players n**, Security Parameter** λ**, Honest Advantage** δ**):**
Run $Blockchain.Genesis(n, \lambda)$ // protocol's initialization algorithm.
$CORRUPTION\_THRESHOLD \leftarrow \frac{(1-\delta)n}{(2-\delta)}$
// Set up our initial variables.
integer $ROUND \leftarrow 0$, Allocate queue $PLAYER\_QUEUE[n]$
Allocate boolean $PLAYER\_CORRUPT[n]$
for $i = 1$ to n; i++ **do**
$PLAYER\_QUEUE[i] \leftarrow null$, $PLAYER\_CORRUPT[i] \leftarrow 0$
end for
// Start the core part of the protocol.
while true **do**
ROUND++
$RunAllPlayers(n)$ // Run players in an order chosen by the adversary.
$AdversarialReordering()$ //Adversary can reorder $PLAYER\_QUEUE$
$AdversarialCorruption()$ // Allow adversary to corrupt players.
end while

Figure 6: Model and Structure: *Elapsed Time Framework*

We next enumerate our algorithm for running all of the players in figure 7. The point of this algorithm is to allow the adversary to adaptively run all of the players, both honest and corrupt, in an order of their choosing. If the adversary tries to run a player twice, we abort the protocol, as this is not allowed.

In figure 8, we define the functionalities of our adversary. To allow the adversary to run an adversarial player we give access to the player's internal state and the player's subroutine queries (like querying a TEE or "mining," for instance). We also allow the adversary to reorder the messages sent by all parties before they are delivered. This is formally done by allowing the adversary to permute the queues (including the queues of the honest players) in $PLAYER\_QUEUE$.

3.5 Blockchain properties and security game

We next define three core blockchain properties: common prefix property, chain quality property, and chain growth property. As argued in [GKL15], these properties together essentially define what it means to be a functional and useful blockchain.

```
PLAYERS[n]; // Array containing instances of all the players
RunAllPlayers(Number of Players n):
  boolean HAS_RAN[n] := all elements set to 0;
  for i = 1 to n; i++ do
    x ← Adversary(·)   //Adversary picks a player to run.
    if HAS_RAN[x] = 0 then
      if PLAYER_CORRUPT[x] = 0 then
        PLAYERS[x].RunPlayer()  // RunPlayer() is specified by protocol
      else RunAdversarialPlayer(x) end if
      HAS_RAN[x] ← 1.
    else
      // The adversary is trying to run a player twice, so run all honest players that have not
      run and exit
      for j = 0; j ≤ n; j++ do
        if HAS_RAN[j] = 0 AND PLAYER_CORRUPT[j] = 0 then
        PLAYERS[j].RunPlayer() end if
      end for
      Return
    end if
  end for
```

Figure 7: Adversarially Ordered Player Running: $ET_{Allplayers}$

Definition 1. ***Common Prefix Property****: Suppose $\mathcal{C}_1$ is a chain which has been accepted by an honest party at round r_1 and $\mathcal{C}_2$ is another chain which has been accepted by some honest party at round r_2. The following holds for all integers $k \geq \ell_{cf}$, where ℓ_{cf} is the common prefix parameter.*

- *if $r_1 < r_2$ then $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$*
- *if $r_2 < r_1$ then $\mathcal{C}_2^{\lceil k} \preceq \mathcal{C}_1$*
- *if $r_1 = r_2$ then $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$ and $\mathcal{C}_2^{\lceil k} \preceq \mathcal{C}_1$*

Definition 2. ***Chain Quality Property****: Suppose $\mathcal{C}$ is a chain that got accepted by some honest party. Any subchain $\tilde{\mathcal{C}}$ of $\mathcal{C}$ of length $\tilde{\ell} \geq \ell_q$ must contain at least $\mu\tilde{\ell}$ many honest blocks. Here ℓ_q, μ are chain quality parameters.*

A block is an honest block, if it is created by an honest party.

Definition 3. ***Chain Growth Property****: Suppose $\mathcal{C}_1$ is a chain of length ℓ_1 which has been accepted by an honest party at round r_1 and $\mathcal{C}_2$ is another chain of length ℓ_2 which has been accepted by some honest party at round r_2. If $r_2 - r_1 > r_g$, then $\ell_2 - \ell_1 \in [\tau(r_2 - r_1), \sigma(r_2 - r_1)]$. Here r_g, τ, σ are chain growth parameters.*

The security of a blockchain system can be stated as the following game between an adversary and a challenger. The game can be characterized by the number of players n, security parameter λ, common prefix parameter ℓ_{cf}, chain quality parameters ℓ_q, μ, chain growth parameters r_g, τ, σ and honest parties advantage δ .

Definition 4. ***Blockchain Security Game****: We consider a security game parameterized by $(n, \lambda, \ell_{cf}$, $\ell_q, \mu, r_g, \tau, \sigma, \delta)$. The game starts with the challenger running **RunProtocol**(n, λ, δ). After some number of rounds $r_{end} = poly(\lambda)$ the adversary chooses to end the game. We say that the adversary wins if at any*

RunAdversarialPlayer (index i):

// Give the adversary control over the player.
$Adversary\,(PLAYERS[i], PLAYER\_QUEUE[i])$.

AdversarialReordering:

// Permute the player queues one by one.
for $i = 1; i \leq n; i++$ **do**
// Adversary picks a permutation to reorder the queue.
$\mathcal{P}_i \leftarrow Adversary\,(\cdot)$; PLAYER_QUEUE[i] $\leftarrow \mathcal{P}_i\,(PLAYER\_QUEUE[i])$
end for

AdversarialCorruption:

// Get the desired corruptions from the adversary.
Allocate boolean $NEW\_CORRUPT[n]$
$NEW\_CORRUPT \leftarrow \{Adversary\,(\cdot) \rightarrow$ boolean $[n]\}$
// Make sure there aren't more than the allowed number of corruptions.
if $\sum_{i=1}^{N} PLAYER\_CORRUPT[i] + \sum_{i=1}^{N} NEW\_CORRUPT[i] \geq CORRUPTION\_THRESHOLD$ **then** Return **end if**
// Set the bits for corruption.
for $i = 1$; $i \leq n$; i++ **do**
if $NEW\_CORRUPT[i] = 1$ **then** $PLAYER\_CORRUPT[i] \leftarrow 1$ **end if**
end for

Figure 8: Adversarial Functionalities: ET_{ADV}

time during the protocol execution at least one of the three core blockchain security properties (common prefix, chain quality or chain growth) does not hold.

4 Elapsed Time Consensus Protocol

In this section, we first construct an elapsed time consensus protocol with a trusted timer. This protocol very closely mirrors the description of proof of luck that we mentioned in the introduction.

4.1 Elapsed Time (ET) consensus with Trusted Timer

We start by defining our block structure for elapsed time consensus. Our proof of block validity π (see Fig 9) contains the following things: a *RoundGenerated* parameter that indicates the round in which the block was initially "mined," a $WaitTime$ that indicates how many rounds until the block can be issued, and a $WaitCert$ which certifies that a TEE generated the $WaitTime$ properly.

Block.π:

π.RoundGenerated // Round of "mining."
π.WaitTime // How many rounds until the block can be issued.
π.WaitCert // Proof that a TEE generated the WaitTime properly.

Figure 9: Block Proof Structure

The ideal version of our ET with trusted timer protocol, denoted by ET_{timer}, works in two phases: (1) Initialization phase, (2) Leader election phase.

Initialization Phase. In this phase, the challenger initializes the blockchain by calling $Genesis()$ (refer to Figure 10), which in turn creates the genesis block $B_{genesis}$ and initializes all the players. Since we consider a setting where all the players are active from the beginning, initialization for all the players happen together (at round 0), and happens before the "leader election" phase starts. It is worth repeating here, since we are in a round based model, all the time values in $PoET_{honest}$ are measured in number of rounds.

Leader Election Phase. In this phase, all the players compete with each other to be elected to generate the next block. Once a block is generated, they start a fresh competition and repeat the process [poe]. We define the leader election protocol in the $RunPlayer()$ routine defined in Figure 11. For an honest user, the challenger runs $RunPlayer()$ exactly once per round.

Genesis(n, λ)**:**
generate the genesis block $B_{genesis}$
$P :=$ Geometric Distribution with parameter p
$\mathcal{H} :=$ hash function $\mathcal{H}$ based on security parameter λ
for $i = 1$ to n; i++ **do**
$PLAYERS[i].Initialize(i, B_{genesis})$
end for

Figure 10: Initialization of the $Blockchain$ for ET_{timer}

For each round, $RunPlayer()$ checks if there are new chains in the input buffer of the player. These chains are generated in the previous round by other players. Our current player picks the best chain according to $PickChain()$ (defined in figure 12) to replace (if applicable) its own local chain. If $PickChain()$ updates the local chain, our player stops competing for the last election and calls $TEE.Run(WaitForBlock, \mathcal{C}, \cdots)$ to start a new competition.

Leader Election Mechanism. The TEE runs the leader election mechanism for a given player by running $WaitForBlock(\mathcal{C})$ (defined in figure 13). Given an existing chain $\mathcal{C}$, the leader election mechanism works very much like a lottery algorithm: a player wins an election if their $WaitTime$ (picked from a pre-defined probability distribution P) is smallest among all players.

The TEE waits for $WaitTime$ rounds before generating the wait certificate $waitCert$ for a block which needs to be added after $\mathcal{C}$, unless the player interrupts by calling a new $TEE.Run(WaitForBlock, \cdots)$. The TEE calls $CreateBlock$ after $WaitTime$ rounds to generate a block which is populated with transactions provided by the player, with a valid $WaitCert$ that proves validity of the $WaitTime$ for that generated block. The player then adds the new block to his local chain and broadcasts the chain. Without a valid $WaitCert$ in the block, other players will not accept the new block as a valid block.

Collision Resolution: It is possible that two players get the same $WaitTime$. In that case, both the players broadcast their chains in the same round, each chain with a newly added block – it is considered a collision.

Each player handles collision locally in the following way (refer to Figure 12 for the pseudocode representation): If two chains $C1$ and $C2$ are of same length, and the last blocks in $C1$ and $C2$ were generated at rounds t_1 and t_2 respectively. Among t_1 and t_2, whichever is smaller the corresponding

chain $\mathcal{C} := empty$; // A chain - an ordered list of blocks.
$playerID$; // denotes the index of the player provided by the challenger
TEE; // denotes the trusted execution environment specific to the player

Initialize(integer i, the genesis block $B_{genesis}$):
  $playerID \leftarrow i$; Add $B_{genesis}$ to $\mathcal{C}$; $TEE.Initialize(i)$;
  $x \leftarrow$ Collected transactions from users; $TEE.Run(WaitForBlock, \mathcal{C}, x)$

RunPlayer():
  Check $PLAYER\_QUEUE[playerID]$ for all new received chains.
  // The player has received a chain from some other player
  if QUEUE is not empty **then**
    $\mathcal{C}_{new} \leftarrow PickChain(playerID)$
    if $\mathcal{C}_{new} \neq \mathcal{C}$ **then**
      $\mathcal{C} \leftarrow \mathcal{C}_{new}$; $x \leftarrow$ Collected transactions from users
      $TEE.Run(WaitForBlock, \mathcal{C}, x)$ // Wait on TEE for the next block
    end if
  else
    // The player has NOT received a chain from some other player
    if $TEE.Poll() \neq Incomplete \wedge TEE.Poll() \neq \perp$ **then**
      // It seems the player is a leader
      $(B, WaitCert) \leftarrow TEE.Poll()$; $B.\pi.WaitCert \leftarrow WaitCert$
      Add B to $\mathcal{C}$; $Broadcast(\mathcal{C})$
      // Again, repeat for the next leader election
      $x \leftarrow$ Collected transactions; $TEE.Run(WaitForBlock, \mathcal{C}, x)$
    end if
  end if

Figure 11: Ideal functionality of ET with timer protocol : ET_{timer}

chain is chosen. However, if $t_1 = t_2$, the collision is not resolved; the player can choose any one of the chains as the current chain and keeps mining for that chain. If C_1 and C_2 are of different lengths, the longer chain is chosen by the player.

Note that, since we have a synchronized round based model and the players have access to a reliable broadcast mechanism, it might seem unlikely to have $t_1 \neq t_2$. However, a compromised player might choose not to broadcast his chain, or selectively send the chain to some players.

Probability Distribution for $WaitTime$. In our protocol, the TEE generates the $WaitTime$ by sampling from a probability distribution specified by the protocol. In the real world, the PoET protocol uses an exponential distribution. However, since we are quantifying time in terms of number of rounds, we must consider a discrete probability distribution. In our case, we use a geometric distribution where the probability of success in each round p.[15] Lemma 13 in Appendix A shows the validity of approximating an exponential distribution with a geometric distribution.

The parameters of the probability distribution are defined by the protocol, and are globally known. Each $WaitTime$ is independent of all previous ones and all other players. We denote the probability distribution with P.

We emphasize that the sampled wait times are never leaked outside the TEE. Due to the memory-

[15]Here we are considering the version of geometric distribution where trial number $WaitTime$ is the first successful trial.

PickChain(integer i denoting the player index):

for each $\mathcal{C}^*$ in $PLAYER\_QUEUE[i]$ **do**
// Check if the chain is valid and longer than the current chain
// In case of a tie, keep the current chain
if $(Length(\mathcal{C}^*) > Length(\mathcal{C})) \wedge IsChainValid(\mathcal{C}^*)$ **then** $\mathcal{C} \leftarrow \mathcal{C}^*$ **end if**
end for
return $\mathcal{C}$

IsChainValid(chain $\mathcal{C}^* = \{B_1, B_2, \ldots B_\ell\}$):

if B_1 is not the genesis block **then** return $false$ **end if**
for each block B_i in $\mathcal{C}^*$ except B_1 **do**
$\mathcal{C}_{i-1} \leftarrow (B_1, \cdots, B_{i-1})$
// Verify that the current block is pointing to the previous block
if ($B_i.s \neq \mathcal{H}(B_{i-1})$) **then** return $false$
// Verify that the block is not created ahead of its previous block
else if ($B_i.\pi.timestamp < B_i.\pi.WaitTime + B_{i-1}.\pi.timestamp$)
then return $false$
// Verify the $WaitCert$ and content for the block
else if $!ValidBlock(B_i, \mathcal{C}_{i-1}) \vee$ **!IsCertValid**$_{TEE}\big(B_i.\pi.WaitCert,$
$B_i.\pi.WaitTime \| WaitForBlock \| (\mathcal{C}_{i-1}, B_i.x)\big)$
then return $false$
else continue
end if
end for
// Verify that the last block is not created ahead of time
if ($B_\ell.\pi.timestamp \geq$ current-round) **then** return $false$ **end if**

Figure 12: Chain Selection mechanism for $PoET_{honest}$

WaitForBlock(chain $\mathcal{C}$, transactions x)

// Verify that the new chain is longer than the local chain
if (TEE.GetCounterValue() $\geq Length(\mathcal{C})$) **then** return $\perp$
else TEE.CounterSet($Length(\mathcal{C})$) **end if**
$waitTime \leftarrow$ draw an element from P
$sleep(waitTime)$ // Wait for the required amount of time
$B \leftarrow CreateBlock(waitTime, \mathcal{C}, x)$; **return** B

CreateBlock(time delay t, chain $\mathcal{C}$, transactions x):

create an empty block B; $B.x \leftarrow x$ // Add transactions
$B.s \leftarrow \mathcal{H}(\mathcal{C}.head)$ // Point to the last block of the existing chain
// Add details that will help others to verify the block
$B.\pi.timestamp \leftarrow$ current-round; $B.\pi.WaitTime \leftarrow t$
$B.\pi.playerID \leftarrow$ current player ID
return B

Figure 13: Function to be run under Trusted Execution Environment (TEE)

lessness of the exponential/geometric distribution[16] adversarial parties gain no information about the sampled wait times until the wait time actually expires and TEE releases the block. The following Lemma captures this fact, and we use it in our security proofs. Without loss of generality we assume, the adversarial nodes always generate a block, when TEE time expires. Even though they are free to do anything with the generated block.

Lemma 1. *The event that any party generates a block in a given round happens with probability p over the random coins of that party's TEE. This event is independent of all other random coins in the protocol as well as adversarial choices.*

Proof. The probability of block generation by an honest party can only be influenced if a new block arrives. However, because the adversary has no knowledge about the current wait times of honest or adversarial parties, drawing a new wait time is exactly same as drawing a new element from the probability distribution with replacement, and hence, is independent of the current wait times or any past wait times.

The memorylessness property of geometric distribution ensures that the probability of generating a block by an honest party is p in a given round, independent of if the party has queried a new wait time in that round or not. An adversarial party can draw a new wait time in two ways:

1. if the party decides to discard the current wait time and draw a new wait time,
2. or, after the completion of the current wait time.

In the first case, the probability for the adversarial party to generate a block in a given round still remains p since the wait time is stored inside the TEE and the party does not see it. In the second case, the probability is trivially p. Note here, we assume that the adversarial party always generates a block, when the TEE time expires. □

5 Security of ET Consensus with Trusted Timer

In our first proof, we prove the security of our ET consensus protocol with a trusted timer. As with the bitcoin backbone papers [GKL15] [GKL17], this involves proving the common prefix property, the chain quality property, and the chain growth property given an honest majority assumption. These properties imply that our ET consensus protocol can be securely used and suggest that the core principles of real-life proof of luck consensus are sound.

It might seem that our proof would be trivial given the presence of (uncompromisable) TEEs. However, this is far from the case: while an adversary is not allowed to interfere with computations in the TEE, they can still block or cause Byzantine faults in communication, causing victim TEEs to fall behind in the chain or receive conflicting information. Additionally, in proof of work protocols blocks are difficult to create since they require a valid proof of work; however, In PoET or our elapsed time consensus with TEE model, blocks only require valid signatures, which means they are easy to create. Moreover, attacks like selfish and stubborn mining [NKMS16] are still possible, and we end up needing essentially the full power of the proof in [GKL15].

Before we begin the proof in earnest, we recall that the hash function and the certification (signature) scheme are assumed to be ideal primitives and thus cryptographically secure. Additionally we assume integrity of the TEE as defined in Theorem 1 in Section 2.4.

We say an honest party adopts a chain $\mathcal{C}'$ at round r if, after completion of the execution of the honest party in round r, the honest party's internal best chain $\tilde{\mathcal{C}}$ becomes $\mathcal{C}'$. We say a chain $\mathcal{C}'$ got adopted by an honest party if that honest party adopted the chain $\mathcal{C}'$ at some round r. We define some additional Boolean random variables:

[16] By memoryless, we mean that the distribution of winning a block is independent of the history of block wins or losses.

Table 1: Table of all parameters

n	:	Total number of players
t	:	Number of corrupted players
δ	:	Advantage of honest parties, ($\frac{t}{n-t} \leq 1-\delta$)
p	:	Probability that an honest player creates a block in a given round
f	:	Probability at least one honest player creates a block in a given round
r_{end}	:	Total number of rounds in the security game
ϵ	:	A security parameter used to bound the "luckiness" of the adversary See the "typical execution" definition in Def. 6
ϵ'	:	Quality of concentration for z-test (Definition 7)
λ	:	Security parameter
ℓ_{cf}	:	Minimum number of blocks in common prefix property
μ	:	Parameter in chain quality property
ℓ_q	:	Minimum number of blocks for which the chain quality property holds.
τ	:	Minimum number of blocks in chain growth property.
σ	:	Maximum number of blocks in chain growth property.
r_g	:	Minimum number of rounds for which the chain growth property holds

- $HON_i^{\geq 1}$ is defined to be 1 if *at least* one honest party creates a block at round i and 0 otherwise.
- HON_i^1 is defined to be 1 if exactly one *single* honest party creates a block at round i and 0 otherwise.
- $ADV_{i,j}$ is defined to be 1 if the jth dishonest party creates a block at round i and 0 otherwise[17].

We also define, $ADV_i := \sum_j ADV_{i,j}$.
In addition, we define the following terms over sets of rounds S:

$$HON^{\geq 1}(S) := \sum_{r \in S} HON_r^{\geq 1}, \quad HON^1(S) := \sum_{r \in S} HON_r^1,$$

$$ADV(S) := \sum_{r \in S} ADV_r.$$

A round r is *successful*, iff $HON_r^{\geq 1} = 1$, it is *uniquely successful* iff $HON_r^1 = 1$.

The integrity of the trusted environment guarantees for all participants (whether honest or adversarial) that the probability of block creation is the protocol parameter p. We note that this also assumes we are using a "memoryless" distribution like an exponential or geometric distribution. Suppose f is the probability that at least one honest party creates a block at a given round. In other words, for any round r, $\Pr[HON_r^{\geq 1} = 1] = f$. Suppose there are n honest players and t corrupted players. Then we have $f = 1 - (1-p)^{n-t}$. Below, we mention an inequality that we will use often.

$$f < p(n-t) < \frac{f}{1-f} \tag{1}$$

Note, the first inequality is a straight forward application of Bernoulli's inequality which says for real $x > -1$ and integer $r \geq 0$ we have, $(1+x)^r > 1 + rx$. The second inequality is another application of Bernoulli's inequality after applying the following inequality $(1-p)^{-(n-t)} > (1+p)^{n-t}$.

[17]The adversary, actually have a choice try to mine in a specific round or not. However, without loss of generality we can consider an adversary who always tries to do so. Even though, the adversary is always free to do whatever with a successfully mined block. This assumption helps us in defining random variables $ADV_{i,j}$ in terms of pure probabilistic events.

We can also easily calculate expectation of the random variables $HON^{\geq 1}(S)$, $HON^{1}(S)$ and $ADV(S)$ for any set of rounds S.

$$\begin{gathered}\mathbb{E}[HON^{\geq 1}(S)] = f|S|, \qquad \mathbb{E}[HON^{1}(S)] = (n-t)p(1-p)^{n-t-1}|S| \\ \mathbb{E}[ADV(S)] = pt|S|\end{gathered} \tag{2}$$

Furthermore, we know $x(1-x)$ is an increasing function when $x \in (0, \frac{1}{2})$. If we assume $f < 1/3$, this in turn implies $f < (n-t)p < \frac{f}{1-f} < \frac{1}{2}$ by inequality (1). Note, $\frac{x}{1-x}$ is also an increasing function for $x \in (0, \frac{1}{2})$, which shows $\frac{f}{1-f} < \frac{1}{2}$, for $f < \frac{1}{3}$. Now, assuming $f < \frac{1}{3}$ we can have the following lower bound on $\mathbb{E}[HON^{1}(S)]$.

$$\begin{aligned}\mathbb{E}[HON^{1}(S)] &= (n-t)p(1-p)^{n-t-1}|S| \\ &> (n-t)p(1-p)^{n-t}|S| \\ &> (n-t)p\left(1-(n-t)p\right)|S| && \text{Bernoulli's inequality} \\ &> f(1-f)|S| && \text{Preceding paragraph} \qquad (3) \\ &> \frac{2}{3}f|S| \qquad (4)\end{aligned}$$

5.1 Honest Majority Assumption and Typical Execution

All of our security guarantees hold if there are enough honest parties in the system, where the exact amount that is "enough" depends on other parameters of the system. For reasonable and real-world choices of parameters, our security proofs can handle a constant fraction of adversarial parties. Below, we formally state the honest majority assumption.

Definition 5 (Honest Majority Assumption)**.** *Suppose n is the total number of parties, and out of them t parties are corrupted. If δ is the advantage of honest parties, then we require that $t < (1-\delta)(n-t)$, where $3f + 3\epsilon < \delta \leq 1$, where ϵ is a positive fraction (used in various concentration bounds) and f is the probability that at least one honest party creates a block at a given round.*

In our security game (Definition 4), the adversary stops the game after r_{end} many rounds. We define such a protocol execution as 'typical' if for any set of consecutive rounds S of size more than λ, the quantities $HON^{\geq 1}(S), HON^{1}(S)$ and $ADV(S)$ do not deviate a lot from their expected values.

Definition 6 (Typical Execution)**.** *An execution of r_{end} rounds, is (ϵ, λ)-typical, for an $\epsilon \in (0,1)$, if for any set $S \subseteq [1, r_{end}]$ of at least λ consecutive rounds the following hold:*
(a) $(1-\epsilon)\mathbb{E}[HON^{\geq 1}(S)] < HON^{\geq 1}(S) < (1+\epsilon)\mathbb{E}[HON^{\geq 1}(S)]$
(b) $(1-\epsilon)\mathbb{E}[HON^{1}(S)] < HON^{1}(S)$
(c) $ADV(S) < (1+\epsilon)\mathbb{E}[ADV(S)]$

Theorem 3. *An execution of r_{end} rounds, is (ϵ, λ)-typical with probability at least $1 - r_{end}(e^{-\Omega(\epsilon^2 \lambda f)} + e^{-\Omega(\epsilon^2 \lambda pt)})$.*

Proof. The first thing to observe here is that the value of p remains same over all rounds because of the memorylessness property of the exponential distribution. Therefore, the random variables $HON_i^{\geq 1}$ (and similarly HON_i^1 and ADV_{ij}) are independent but identically distributed random variables.

Now we use Chernoff bound on $HON^{\geq 1}(S) = \sum_{i \in S} HON_i^{\geq 1}$ to derive the following,

$$Pr\left[HON^{\geq 1}(S) \leq (1-\epsilon)\mathbb{E}[HON^{\geq 1}(S)]\right] \leq e^{-\epsilon^2 \mathbb{E}[HON^{\geq 1}(S)]/2}.$$

And,

$$Pr\left[HON^{\geq 1}(S) \geq (1+\epsilon)\mathbb{E}[HON^{\geq 1}(S)]\right] \leq e^{-\epsilon^2\mathbb{E}[HON^{\geq 1}(S)]/3}.$$

From Equation(2), we have $\mathbb{E}[HON^{\geq 1}(S)] = f|S| \geq f\lambda$. We also know there are at most r_{end} many possible choices of set S. Hence property (a) of Definition 6 holds with probability at least $1 - r_{\text{end}}e^{-\Omega(\epsilon^2\lambda f)}$. Using a similar argument and Inequality (4) we can show property (b) also holds with probability at least $1 - r_{\text{end}}e^{-\Omega(\epsilon^2\lambda f)}$. Similarly, for property (c), the probability is at least $1 - r_{\text{end}}e^{-\Omega(\epsilon^2\lambda pt)}$ because $\mathbb{E}[ADV(S)] = pt|S|$. □

Now we shall look at some more properties of typical execution through the following lemma.

Lemma 2. *For any set S of at least λ consecutive rounds, assuming integrity of the TEE the following properties hold in a typical execution:*

(a) $(1-\epsilon)f|S| < HON^{\geq 1}(S) < (1+\epsilon)f|S|$

(b) $HON^1(S) > (1-\epsilon)f(1-f)|S| > (1-\frac{\delta}{3})f|S|$

(c) $ADV(S) < \frac{t}{n-t}\frac{f}{1-f}|S| + \epsilon f|S| < (1-\frac{2\delta}{3})f|S|$

(d) $ADV(S) < (1+\frac{\delta}{2})\frac{t}{n-t}HON^{\geq 1}(S) + \epsilon f|S|$

(e) $ADV(S) < HON^1(S)$.

Proof. Recall, $\mathbb{E}[HON^{\geq 1}(S)] = f|S|$. Hence, part(a) follows from the definition of typical execution (Definition 6). For part (b) from definition of typical execution and equation (2) we have,

$$\begin{aligned} HON^1(S) &> (1-\epsilon)\mathbb{E}[HON^1(S)] \\ &> (1-\epsilon)f(1-f)|S| \qquad \text{By Inequality (3)} \end{aligned}$$

The last inequality uses the Honest majority assumption (Definition 5) which guarantees $f < 1/3$. Finally, from honest majority assumption we also have $f + \epsilon < \delta/3$, which implies $(1-\epsilon)(1-f) > (1-\frac{\delta}{3})$. This completes proof of part (b). For part (c) from definition of typical execution and equation (2) we have,

$$ADV(S) < (1+\epsilon)pt|S| < (1+\epsilon)\frac{t}{n-t}\frac{f}{1-f}|S| < \frac{t}{n-t}\frac{f}{1-f}|S| + \frac{1-\delta}{1-f}\epsilon f|S|$$

The second inequality uses equation (1). The last inequality uses honest majority assumption, which also implies $\frac{1-\delta}{1-f} < 1$. This proves first inequality of part (c). Second inequality of part (c) again follows from honest majority assumption, i.e. $\frac{t}{n-t} < 1-\delta$ and $3(\epsilon+f) < \delta$.

Part (d) is implied by first inequality of part (c), lower bound of $HON^{\geq 1}(S)$ on part (a), because the honest majority assumption which guarantees $(\epsilon + f) < \frac{\delta}{3} < \frac{1}{3}$ implies $(1+\frac{\delta}{2})(1-\epsilon) > \frac{1}{1-f}$. Second inequalities of part (b) and part (c) readily imply part (e). □

5.2 Chain Growth Property

Now we provide an upper bound as well as a lower bound on the number of blocks generated given a chain $\mathcal{C}$ and a sequence of rounds S with length $s > \lambda$. Here we prove that the number of blocks x added to the chain $\mathcal{C}$ in s rounds is upper bounded by σs and lower bounded by τs, for some constants τ and σ.

Lemma 3. *$\mathcal{C}_1$ and $\mathcal{C}_2$ be two chains which got adopted by some honest parties. Suppose B_1 and B_2 are the k-th blocks of chains $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. If $\texttt{id}(B_1)$ is an honest user and $\texttt{round}(B_1)$ is a uniquely successful round, then either $B_1 = B_2$ or $\texttt{id}(B_2)$ is corrupted.*

Proof. For contradiction, we assume $B_1 \neq B_2$ and $\texttt{id}(B_2)$ is honest. Security of the signature scheme implies the blocks B_1 and B_2 are actually created by $\texttt{id}(B_1)$ and $\texttt{id}(B_2)$. As, $\texttt{round}(B_1)$ is uniquely successful round and both $\texttt{id}(B_1)$ and $\texttt{id}(B_2)$ are honest, we have $\texttt{round}(B_1) \neq \texttt{round}(B_2)$. Suppose, $\texttt{round}(B_1) < \texttt{round}(B_2)$; as both of the players are honest $\texttt{id}(B_2)$ must have received the chain ending in B_1 with length k on or before $\texttt{round}(B_2)$. This implies position of the block B_2 must be greater than k, which is a contradiction. The cryptographic security of the hash function ensures an honest party creates a block at position k and that adversarial players cannot insert that block at a different position. A similar argument holds for the case $\texttt{round}(B_1) < \texttt{round}(B_2)$. □

Lemma 4 (Chain Growth Lemma)**.** *Suppose an honest party has adopted a chain of length ℓ at round r. Then, by round $h > r$, every honest party has adopted a chain of length at least $\ell + \sum_{i=r}^{h-1} HON_i^{\geq 1}$.*

Proof. We prove the above theorem using induction on $h \geq r+1$.

Induction base: The protocol has moved only one round after round r, hence $h = r+1$. If at round r, an honest party has a chain of length ℓ, every honest party will adopt a chain of length at least ℓ by round $r+1$. Additionally, if $HON_r^{\geq 1} = 0$, the statement follows directly. If $HON_r^{\geq 1} = 1$, the successful honest party will broadcast a chain of length $\ell + 1 = \ell + HON_r^{\geq 1}$, and all honest parties will adopt a chain of at least that length by round $h = r+1$.

Inductive step: Let us assume that every honest party has adopted a chain of length at least $\ell' = \ell + \sum_{i=r}^{h-2} HON_i^{\geq 1}$ by round $h-1$.
Now, two things could have happened on round $h-1$:

1. $HON_{h-1}^{\geq 1} = 0$, in which case $\sum_{i=r}^{h-1} HON_i^{\geq 1} = \sum_{i=r}^{h-2} HON_i^{\geq 1}$. Hence, the statement follows.
2. $HON_{h-1}^{\geq 1} = 1$, in that case a successful honest party will broadcast a chain of length at least $\ell' + 1$ in round $h-1$. By round s, all honest parties will adopt a chain of length at least $\ell' + 1 = \ell + \sum_{i=r}^{h-2} HON_i^{\geq 1} + 1 = \ell + \sum_{i=r}^{h-1} HON_i^{\geq 1}$.

□

Lemma 5. *Suppose $\mathcal{C}$ is a chain adopted by an honest party during a typical execution. For any $k \geq \max(2\lambda f, 4)$, let $B_m, B_{m+1}, \cdots, B_{m+k-1}$ be k consecutive blocks of the chain $\mathcal{C}$. Then, we have*

$$|[\texttt{round}(B_m), \texttt{round}(B_{m+k-1})]| \geq \frac{k}{2f}$$

.

Proof. Suppose $S' = [\texttt{round}(B_m), \texttt{round}(B_{m+k-1})]$. For contradiction let us assume $|S'| < \frac{k}{2f}$. Consider the set S of consecutive rounds such that $S \supseteq S'$ and $|S| = \lceil \frac{k}{2f} \rceil$. Integrity of the trusted execution environment, security of the signature scheme along with the fact chain $\mathcal{C}$ has been adopted by an honest party ensures $HON^{\geq 1}(S') + ADV(S') \geq k$. As $S' \subseteq S$, this in turn implies $HON^{\geq 1}(S) + ADV(S) \geq k$. As, $|S| \geq \lambda$, we can apply part (a) and part (c) of Lemma 2 and it imply the following.

$$
\begin{aligned}
& HON^{\geq 1}(S) + ADV(S) \\
& < (1+\epsilon)f|S| + (1 - \frac{2\delta}{3})f|S| \\
& < (2 + \epsilon - \frac{\delta}{3})f|S| \\
& \leq (2-f)f|S| && \text{Since } 3f + 3\epsilon \leq \delta \\
& < (2-f)f(\frac{k}{2f} + 1) && \text{Since } |S| = \lceil \frac{k}{2f} \rceil < \frac{k}{2f} + 1 \\
& \leq k - \frac{kf}{2} + 2f - f^2 < k + f(1 - k/4) < k. && \text{Since } k \geq 4
\end{aligned}
$$

This shows we have a contradiction. □

A direct corollary to Lemma 2, the chain growth lemma (Lemma 4), and Lemma 5 is the following theorem.

Theorem 4 (chain-growth)**.** *In a typical execution, the chain growth property holds with parameters $\tau = (1-\epsilon)f$, $\sigma = 2f$ and $r_g > \lambda$.*

5.3 Common Prefix Property.

We want the honest parties to eventually agree on a common chain. If the common prefix property holds, once a transaction is included in a block B, the transaction becomes irreversible once honest parties have mined enough blocks extending the chain that includes B.

Lemma 6 (Common Prefix Lemma)**.** *In a typical execution, for two chains $\mathcal{C}_1$ and $\mathcal{C}_2$ with $\texttt{len}(\mathcal{C}_2) \geq \texttt{len}(\mathcal{C}_1)$, if $\mathcal{C}_1$ is adopted by an honest party at round r, and $\mathcal{C}_2$ is either adopted by an honest party or broadcast at round r, then $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$ and $\mathcal{C}_2^{\lceil k} \preceq \mathcal{C}_1$, for all $k \geq \max(2\lambda f, 4)$.*

Proof. Let us assume, for the sake of a contradiction, that there exists a $k > 2\lambda f$ such that $\mathcal{C}_1^{\lceil k} \not\preceq \mathcal{C}_2$ or $\mathcal{C}_2^{\lceil k} \not\preceq \mathcal{C}_1$. Suppose B^* is the last block on the common prefix of $\mathcal{C}_1$ and $\mathcal{C}_2$ such that $\texttt{id}(B^*)$ is honest. Let us denote $\texttt{round}(B^*) = r^*$. Note that B^* can be the genesis block, in which case $r^* = 0$.

Now, we define $S = \{i : r^* < i < r\}$. Suppose $B_m, B_{m+1}, \cdots, B_{m+k'-1}$ are k consecutive blocks of the chain $\mathcal{C}_1$, where B_m is the next block after $B*$ and $B_{m+k'-1}$ is the last block of $\mathcal{C}_1$. Clearly, $k' \geq k \geq \max(2\lambda f, 4)$ and we can apply Lemma 5. This implies, $|[\texttt{round}(B_m), \texttt{round}(B_{m+k'-1})]| \geq \frac{k'}{2f} \geq \lambda$. We also know, $S \supseteq [\texttt{round}(B_m), \texttt{round}(B_{m+k'-1})]$. Hence, $|S| \geq \lambda$ and Lemma 2 applies for the set of rounds S.

For an uniquely successful round $u \in S$, let j_u be the position at which the uniquely successful honest party created the block. J be the set of positions at which honest parties created the blocks on uniquely successful rounds. $J = \{j_u : u \in S, HON_u^1 = 1\}$. Suppose the maximum value of the set J is $\max(J)$. Then, $\texttt{len}(\mathcal{C}_1) \geq \max(J)$, since $\mathcal{C}_1$ is adopted by an honest party at round r, by which the honest party has already received a chain of length $\max(J)$.

Since, $\texttt{len}(\mathcal{C}_2) \geq \texttt{len}(\mathcal{C}_1)$, j^{th} the block exists in both the chains $\mathcal{C}_1$ and $\mathcal{C}_2$ for all $j \in J$. We denote such blocks by $B_{1,j}$ and $B_{2,j}$ respectively. Now, we want to claim for all $j \in J$ at least one of the players between $\texttt{id}(B_{1,j})$ and $\texttt{id}(B_{1,j})$ is corrupted. By Lemma 3, if both $\texttt{id}(B_{1,j})$ and $\texttt{id}(B_{1,j})$ are honest then we must have $B_{1,j} = B_{2,j}$. Cryptographic strength (collision resistance) of the hash function implies $B_j = B_{1,j} = B_{2,j}$ belongs to the common prefix of chains $\mathcal{C}_1$ and $\mathcal{C}_2$. However, we also know $\texttt{round}(B_j) > \texttt{round}(B^*)$ and B^* is the last block in the common prefix such that $\texttt{id}(B^*)$ is honest. This implies a contradiction.

Now, we have established the fact that for all $j \in J$ at least one of the players between $\texttt{id}(B_{1,j})$ and $\texttt{id}(B_{2,j})$ is corrupted. Hence, the total number of blocks B such that $\texttt{id}(B)$ is corrupted, $B \in \mathcal{C}_1 \cup \mathcal{C}_2$ and $\texttt{round}(B) \in S$ must be more than or equal to size of set J. This along with the integrity of the trusted execution environment implies $ADV(S) \geq |J| = HON^1(S)$. However, for a typical execution with $|S| \geq \lambda$, by part (e) of Lemma 2 we have $ADV(S) < HON^1(S)$. Hence, we arrive at contradiction. Therefore, we can say that for all $k > 2\lambda f$, it holds that $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$ and $\mathcal{C}_2^{\lceil k} \preceq \mathcal{C}_1$. □

Theorem 5 (Common Prefix)**.** *In a typical execution the common prefix property holds with parameter* $\ell_{cf} \geq \max(2\lambda f, 4)$.

The proof follows directly from the Common Prefix Lemma.

5.4 Chain Quality Property.

We also want the property that at least a constant fraction of blocks are added by honest parties in a chain $\mathcal{C}$ that is adopted by an honest party.

Theorem 6 (Chain Quality)**.** *In a typical execution, the chain quality property holds with parameters* $\ell_q \geq \max(2\lambda f, 4)$ *and* $\mu = 1-(1+\frac{\delta}{2})\frac{t}{n-t} - \frac{\epsilon}{1-\epsilon} > 1-(1+\frac{\delta}{2})\frac{t}{n-t} - \frac{\delta}{2}$ *for any chain adopted by any honest party.*

Proof. Let us consider a chain $\mathcal{C}$, which has been adopted by an honest party P at round r, such that $\texttt{len}(\mathcal{C}) > \ell_q$. Suppose $\mathcal{C}$ consists of sequence of blocks $(B_1, B_2, \ldots, B_{\texttt{len}(\mathcal{C})})$ and $(B_u, B_{u+1}, \ldots, B_{u+\ell_q-1})$ is an arbitrary ℓ_q length subsequence of $\mathcal{C}$.

Suppose $(B_{u'}, B_{u'+1} \ldots, B_{u'+L-1})$ is the shortest subsequence of $\mathcal{C}$ containing $(B_u, B_{u+1}, \ldots, B_{u+\ell_q-1})$ (i.e. $u' \leq u$ and $L \geq \ell_q$) such that:

1. $\texttt{id}(B_{u'})$ is honest
2. there exists an honest party which adopted the chain $(B_1, B_2, \ldots, B_{u'+L-1})$

Observe that B_1 is the genesis block and $\texttt{id}(B_1)$ is honest by definition. An honest party P adopted the chain $\mathcal{C}$ and $\texttt{len}(\mathcal{C}) > \ell_q$ Hence, the whole chain $\mathcal{C}$ trivially satisfies the above properties, except it might be not the shortest one. This shows existence of the shortest subsequence $B_{u'}, B_{u'+1} \ldots, B_{u'+L-1}$.

Suppose, $r_1 = \texttt{round}(B_{u'})$ and the earliest round at which the chain $(B_1, B_2, \ldots, B_{u'+L-1})$ got adopted by an honest party is r_2. Let S be the sequence of rounds defined as $S = \{r : r_1 \leq r < r_2\}$. Observe that,

$$S \supseteq [\texttt{round}(B_{u'}), \texttt{round}(B_{u'+L-1})] \supseteq [\texttt{round}(B_u), \texttt{round}(B_{u+\ell_q-1})].$$

Hence, by Lemma 5, we have $|S| \geq \ell_q/2f \geq \lambda$ and properties of typical execution are applicable (Lemma 2) for the set of rounds S.

Let x be the number of honest blocks in the ℓ_q length sequence. In other words $x = |\{B \in (B_u, B_{u+1}, \ldots, B_{u+\ell_q-1}) | \texttt{id}(B) \text{ is honest}\}|$. For contradiction, we assume the chain quality property does not hold for this ℓ_q length sequence of blocks $(B_u, B_{u+1}, \ldots, B_{u+\ell_q-1})$. Hence,

$$x < \mu\ell_q \leq \mu L \tag{5}$$

As the chain $(B_1, B_2, \ldots, B_{u'+L-1})$ got adopted by an honest party in round r_2; for all $i \in [u', u'+L-1]$ we have $\texttt{round}(B_i) \in S$. Integrity of the trusted execution environment ensures

$$\begin{aligned} ADV(S) &\geq |\{B \in (B_u, B_{u+1}, \ldots, B_{u+\ell_q-1}) | \texttt{id}(B) \text{ is corrupted}\}| \\ &= L - x > (1-\mu)L \end{aligned} \tag{6}$$

The last inequality uses inequality (5), which is our contradiction assumption. Now, Lemma 4 implies $u' + L - 1 \geq u' + HON^{\geq 1}(S)$ or equivalently $L > HON^{\geq 1}(S)$. Hence inequality (6) can be rewritten as $ADV(S) > (1-\mu)HON^{\geq 1}(S)$. As we have seen before, $|S| \geq \lambda$. Hence. by part (a) and part (d) of Lemma 2

$$\begin{aligned}(1-\mu)HON^{\geq 1}(S) &= \left(\left(1+\frac{\delta}{2}\right)\frac{t}{n-t} + \frac{\epsilon}{1-\epsilon}\right)HON^{\geq 1}(S) \\ &\geq \left(1+\frac{\delta}{2}\right)\frac{t}{n-t}HON^{\geq 1}(S) + \epsilon f|S| \\ &> ADV(S)\end{aligned}$$

Hence, we have a contradiction $ADV(S) > (1-\mu)HON^{\geq 1}(S) > ADV(S)$. □

The above theorem shows that a chain $\mathcal{C}$ of length at least $2\lambda f$ adopted by an honest party will have at least μ fractions of honest blocks. In table 2 we show some examples with possible values of ϵ, f, δ and how the parameters τ, k, μ corresponding to the security properties(namely, chain growth, common prefix and chain quality properties) vary.

Theorem 7 (Security of ET Consensus with Trusted Timer)**.** *For a security parameter λ, total number of parties $n \in poly(\lambda)$, and given the honest majority assumption defined in Definition 5, suppose the following parameter inequalities hold:*

- $\ell_{cf} \geq \max(2\lambda f, 4)$
- $r_g \geq \lambda$, $\tau = (1-\epsilon)f$, $\sigma = 2f$
- $\ell_q \geq \max(2\lambda f, 4)$, $\mu = 1 - (1+\frac{\delta}{2})\frac{t}{n-t} - \frac{\epsilon}{1-\epsilon} > 1 - (1+\frac{\delta}{2})\frac{t}{n-t} - \frac{\delta}{2}$.

Then it is the case that no efficient adversary can win the blockchain security game *as defined in Definition 4 with more than negl (λ) probability.*

Proof. Follows directly from Theorem 4, Theorem 5 and Theorem 6. □

Theorem 7 shows that elapsed time consensus with trusted timer provides security guarantees similar to bitcoin — the chain quality, chain growth, and common prefix properties hold for the honest majority assumption defined in Definition 5. To be concrete, in Table 2 we show some examples with possible values of ϵ, f, δ and how the security parameters $\tau, \sigma, \ell_{cf}, \mu$ vary.

Table 2: How the security property parameters $(\tau, \sigma, r_g, \ell_{cf}, \ell_q, \mu)$ of ET consensus with Trusted Timer corresponding to chain growth, common prefix and chain quality properties vary based on the protocol parameters $\delta, \epsilon, f, \lambda$. The first column presents the values of ϵ and f defined in the honest majority assumption, the second column the minimum δ (accurate up to two decimal places) to satisfy the honest majority assumption; the third, fourth, fifth, sixth, seventh, and eighth columns are τ, σ, r_g, ℓ_{cf}, ℓ_q and μ respectively as defined in Theorem 7. We assume $\lambda >> 4$.

Protocol parameters	δ	τ	σ	r_g	ℓ_{cf}	ℓ_q	μ
$\epsilon = 0.05, f = 0.05$	0.3	0.05	0.1	λ	0.1λ	0.1λ	0.14
$\epsilon = 0.1, f = 0.1$	0.6	0.09	0.2	λ	0.2λ	0.2λ	0.36
$\epsilon = 0.1, f = 0.2$	0.9	0.18	0.4	λ	0.4λ	0.4λ	0.74

6 ET Consensus with Z-test

One of the biggest benefits of proof-of-work algorithms is that they are very difficult to compromise (so long as basic assumptions about cryptography continue to hold). Operations within a TEE, however, may be subject to attack [KHF+19, VMW+18, LSG+18, KKSAG18]. In PoET, PoL, and ET consensus with TEEs, we are mainly concerned about two forms of attack: accelerating the trusted timer and impersonating (or stealing the secret keys from) a TEE. In both cases, an attacker can create an invalid leadership claim and break the security of the protocol.

To address this issue, PoET implements a "z-test" (or more accurately, it implements a "1 sample z-test") that limits the number blocks any validator can win over a period of time. The "z-test" is based on the observation that while any validator can win any block (the fairness principle), the probability that a particular validator wins a disproportionately large number of blocks is extremely low. The "z-test" computes the value of the standard deviation of a specific observation relative to the expected value. So the z-value computed for a set of observations is the number of standard deviations the observations are from the expectation.

To be concrete, with a population of 1000 validators, the probability of winning is 1/1000. That means that for a sample of 100000 blocks, a validator would be expected to claim 100 blocks. If we observe that a validator has actually won 125 blocks, that would mean the "z-value" would be about 2.5. We can then use the z-value to compute the probability that a given validator is winning more often than it should. For this example, with 99.5% confidence, we can say that a validator that wins 125 is winning more often than it should.

If we set the "z-value" to be a value that honest parties should only achieve with negligible probability, then the z-test will function as a rate-limiter for an adversary without ever impacting the honest parties.

6.1 Our ET Consensus Protocol with z-test

While the PoET z-test is an excellent innovation, it unfortunately does not help us to prove the security of any ET consensus protocols. In our case, we implement a more complicated z-test than what is currently present in PoET. Instead of rate-limiting what percentage of blocks any given participant can win on the chain, we rate-limit each participant's block wins over a sliding window of time (or, in our protocol, a sliding window of rounds). Given the distribution of block creation, we can figure out the expectation of (and appropriate other statistics around) how many blocks a player wins over a particular time period. Deviating too far from this results in future blocks being declared invalid.

More precisely, in terms of the protocol formalizations, we limit the number of blocks any given participant can win over some number of rounds of the protocol based on the distribution of block wins. We set our z-test parameters so that honest parties will only be affected with negligible probability, so our z-test has no negative impact on honest blockchain operation. Our new z-test more effectively stops an adversary from concentrating a high number of blocks in a very small amount of time and allows us to state concrete facts about the security of our protocol with the z-test.

In this section, we do not assume any integrity of the trusted execution environment. Hence, adversarial parties can generate arbitrarily many valid blocks per round. However, honest parties apply the 'z-test' before accepting a chain, which checks that no single player is producing more than their fair share of the blocks. For a consecutive set of rounds S, we denote number of adversarial blocks in a chain $\mathcal{C}$ by $ADV_C(S)$. In other words, $ADV_C(S) = |\{B \in \mathcal{C} : \texttt{round}(B) \in S \text{ and } \texttt{id}(B) \text{ is corrupted. }\}|$. For a player $i \in [1, n]$, $Z_{\mathcal{C}}(i, S)$ denotes the number of blocks in chain $\mathcal{C}$ produced by player i, created in rounds in S, i.e. $Z_{\mathcal{C}}(i, S) = |\{B \in \mathcal{C} : \texttt{round}(B) \in S \text{ and } \texttt{id}(B) = i\}|$. Hence, for a chain $\mathcal{C}$ and set of rounds S, we have $ADV_{\mathcal{C}}(S) = \sum_{i \text{ is corrupted}} \sum_{i \in [1,n]} Z_{\mathcal{C}}(i, S)$.

Definition 7. *Let $\mathcal{C}$ be a chain. For $\epsilon' \in (0, 1)$ and $\lambda > 0$ the function $z_{\epsilon',\lambda} : \mathcal{C} \times [1, ..., n] \to \{0, 1\}$ be defined in the following way:*

$$z_{\epsilon',\lambda}(\mathcal{C}, i) = \begin{cases} 0 & \text{if } \exists \text{ set of consecutive rounds } S \text{ s.t.} \\ & |S| \geq \lambda \text{ and } Z_{\mathcal{C}}(i, S) > (1+\epsilon')p|S| \\ 1 & \text{otherwise} \end{cases} \quad (7)$$

$z_{\epsilon',\lambda}(\mathcal{C}, i)$ denotes whether party i contributed less than allowed number of blocks in chain $\mathcal{C}$ or not.

6.2 Protocol Description

The ET protocol with z-test remains almost the same, except that now the TEE is not considered to be secure in any way. However, there is an additional check for chains to ensure that they satisfy the z-test criteria (Definition 7). We call this version of ET consensus ET_{ztest} which borrows all the functionalities from ET_{timer} (refer to figure 11), except the chain validation mechanism. The new chain validation mechanism for ET_{ztest} is defined in Figure 14.

Optionally, for performance improvement, ET_{ztest} does not have to wait on the TEE to generate the block. Instead, a player can just query the $WaitTime$ and $WaitCert$. This can be very useful practically because if each round is small enough, querying the TEE every round can be really inefficient. We define the improved version in figure 15 of Section 9. However, for our security proof we use the version from figure 11.

Although we do not assume any security of any TEE (or even the existence of a TEE), the probability any honest party generates a block in any given round still remains p. This holds because adversarial nodes cannot compromise honest parties and change their wait time distributions. We capture this in the following lemma. The proof is very similar to the proof of Lemma 1.

Lemma 7. *The event that any honest party generates a block in a given round happens with probability p, over the random coins of that party's TEE. This event is independent of all other random coins in the protocol as well as adversarial choices.*

IsChainValid(chain $\mathcal{C} = \{B_1, B_2, \ldots B_\ell\}$):
  if B_1 is not the genesis block **then** return $false$ **end if**
  for each block B_i in $\mathcal{C}$ except B_1 **do**
    $\mathcal{C}_{i-1} \leftarrow (B_1, \cdots, B_{i-1})$
    // Verify that the current block is pointing to the previous block
    if $B_i.s \neq \mathcal{H}(B_{i-1})$ **then** return $false$
    // Verify that the block is not created ahead of its previous block
    else if $(B_i.\pi.timestamp < B_i.\pi.WaitTime + B_{i-1}.\pi.timestamp)$ **then** return $false$
    // Verify the $WaitCert$ for the block
    else if $!ValidBlock(B_i, \mathcal{C}_{i-1}) \vee$ **!IsCertValid**$_{TEE}(B_i.\pi.WaitCert,$
      $B_i.\pi.WaitTime \| WaitForBlock \| (\mathcal{C}_{i-1}, B_i.x))$
    then return $false$
    else continue
    end if
  end for
  // Verify that the block is not created ahead of time
  if ($B_\ell.\pi.timestamp \geq$ current-round) **then** return $false$ **end if**
  // Verify that the chain satisfies z-test property
  if ($z_{\epsilon,\lambda}(\mathcal{C}, B_\ell.\pi.playerID) = 0$) **then** return $false$ **end if**

Figure 14: Chain Validation mechanism for ET_{ztest}

7 Security of ET Consensus with Z-test

With the z-test, we strictly bound how many blocks an adversary can win on any given chain in any given period of time. This is a powerful restriction, but it still gives an adversary quite a bit of freedom. The adversary can still essentially allocate these blocks however they want over the given time periods. In addition, the adversary can also create many different small forks or chains like a "nothing at stake" attacker [BCNPW18].

7.1 Intuition about the security proof

It might seem that this would allow an adversary to perform powerful attacks. For instance, since we assume that an adversary always wins "race conditions" with honest parties (i.e. if an adversary and an honest party publish blocks at the same time, the adversary wins), it is possible for an adversary to "cover up" more honest blocks in this manner than would occur with random chance.

However, the proofs of bitcoin in the bitcoin backbone protcol [GKL15] and associated works (including our proofs without the z-test) typically use a bound on the number of adversarial blocks in most of their proofs and are agnostic to where the adversarial blocks are present. We shall follow a similar strategy (with a slight inflation of the percentage of honest users required) to prove the security of ET_{ztest}. However, the z-test bound the behaviour of the adversary on *each* chain, not globally. Therefore, we use a slightly modified honest majority assumption (where we require slightly more honest parties) compared to bitcoin (and ET with TEE), and prove that honest parties are "stronger" than the adversary on each chain.

It is important to recall here that, to prove the desired security properties for ET_{ztest}, we assume cryptographic security of the hash function and the signature scheme, and an honest majority (formally stated below). However, we do not assume the TEEs to be secure in any way.

We start by presenting our honest majority assumption. Note that it is substantially more complicated than in the protocol with the trusted timer, as are other theorem statements in this section. For simplification, in subsection 8.2, we show how to pick parameter choices that dramatically simplify many of these statements.

Definition 8 (Honest Majority Assumption)**.** *Suppose n is the total number of parties, and out of them t parties are corrupted. Then we require that $t < (1-\delta)(n-t)$, where $\max\left(\frac{2f+\epsilon+\epsilon'-f^2-f\epsilon}{1+\epsilon'}, \frac{1+\epsilon+2\epsilon'-2\epsilon' f-2\epsilon f}{2(1+\epsilon')(1-f)}\right) < \delta \leq 1$, $\epsilon \in (0,1)$ and $\epsilon' > 0$.*

7.2 Typical Execution

We start be (re)defining a typical execution and prove some simple properties about it.

We slightly modify the definition of typical execution here compared to Section 5. Let $HON_{r,i}$ be a boolean random variable that denotes whether honest player i successfully generated a block at round r or not. We denote $HON_{\cdot,i}(S) = \sum_{r \in S} HON_{r,i}$. We also do not impose any condition on adversarial success, because without integrity of the trusted execution environment the adversary is free to generate as many valid blocks as it wants (although, if they generate too many, they will be rejected by honest parties).

Definition 9 (Typical Execution)**.** *An execution of r_{end} rounds, is $(\epsilon, \epsilon', \lambda)$-typical, for some $\epsilon \in (0,1), \epsilon' > 0$, if for any set S of at least λ consecutive rounds the following hold:*

(a) $(1-\epsilon)\mathbb{E}[HON^{\geq 1}(S)] < HON^{\geq 1}(S) < (1+\epsilon)\mathbb{E}[HON^{\geq 1}(S)]$ *and* $(1-\epsilon)\mathbb{E}[HON^{1}(S)] < HON^{1}(S)$

(b) For all honest players i, $HON_{\cdot,i}(S) < (1+\epsilon')\mathbb{E}[HON_{\cdot,i}(S)]$

Theorem 8. *An execution of r_{end} rounds is $(\epsilon, \epsilon', \lambda)$-typical with probability at least $1 - r_{end}(e^{-\Omega(\epsilon^2 \lambda f)} + (n-t)e^{-\Omega(\epsilon'^2 \lambda p)})$.*

Proof. Note, there are $(n-t)$ honest parties and for every honest party i, we have $\mathbb{E}[HON_{\cdot,i}(S)] = p|S|$. The theorem follows with a Chernoff bound and a line of argument very similar to proof of Theorem 3. □

Now, let us look at some more properties of a typical execution under z-test. Later in this section, we are going to use those properties to analyze the chain growth and chain quality properties.

Lemma 8. *For any set S of at least λ consecutive rounds, the following properties hold in a typical execution where $\mathcal{C}$ is a chain adopted by an honest party.*

(a) $(1-\epsilon)f|S| < HON^{\geq 1}(S) < (1+\epsilon)f|S|$

(b) $ADV_{\mathcal{C}}(S) < \frac{(1+\epsilon')t}{(n-t)(1-f)(1-\epsilon)} HON^{\geq 1}(S)$

(c) $2ADV_{\mathcal{C}}(S) < HON^{1}(S)$

(d) $ADV_{\mathcal{C}}(S) < (1-f-\epsilon)f|S|$

Proof. Recall, $\mathbb{E}[HON^{\geq 1}(S)] = f|S|$. Hence, part (a) readily follows from definition of typical execution (Definition 9). The chain $\mathcal{C}$ got adopted by an honest party, hence it passed the 'z-test' for all parties. As $|S| \geq \lambda$, for all players $i \in [1, n]$ we have $Z_{\mathcal{C}}(i, S) < (1+\epsilon')p|S|$. Hence,

$$ADV_{\mathcal{C}}(S) = \sum_{\substack{i \in [1,n] \\ i \text{ is corrupted}}} Z_{\mathcal{C}}(i, S) < (1+\epsilon')pt|S| < \frac{(1+\epsilon')tf}{(n-t)(1-f)}|S| \tag{8}$$

The last inequality above uses inequality (1). Now we can prove part (b) by applying $HON^{\geq 1}(S)$ lower bound from part (a). For part (c), from the definition of typical execution and , we have

$$\begin{aligned}
HON^{1}(S) &> (1-\epsilon)\mathbb{E}[HON^{1}(S)] \\
&= (1-\epsilon)(n-t)p(1-p)^{n-t-1}|S| \\
&> (1-\epsilon)(n-t)p(1-p)^{n-t}|S| \\
&> (1-\epsilon)(n-t)p(1-p(n-t))|S| && \text{By Bernoulli's inequality} \\
&> (1-\epsilon)(n-t)p(1-\frac{f}{1-f})|S| && \text{By Inequality (1)} \\
&= \frac{(1-\epsilon)(1-2f)}{(1+\epsilon')(1-f)} \frac{(n-t)}{t}(1+\epsilon')pt|S| \\
&> \frac{(1-\epsilon)(1-2f)}{(1+\epsilon')(1-f)} \frac{(n-t)}{t} ADV_{\mathcal{C}}(S) && \text{By Inequality (8)} \\
&> \frac{(1-\epsilon)(1-2f)}{(1+\epsilon')(1-f)(1-\delta)} ADV_{\mathcal{C}}(S) && \text{Definition 8}
\end{aligned}$$

From the honest majority assumption or Definition 8 we also have

$$\frac{1+\epsilon+2\epsilon'-2\epsilon' f-2\epsilon f}{2(1+\epsilon')(1-f)} < \delta < 1,$$

which in turn implies $\frac{(1-\epsilon)(1-2f)}{(1+\epsilon')(1-f)(1-\delta)} > 2$. This completes the proof of part (c). For part (d), from Inequality (8) we have

$$\begin{aligned} ADV_{\mathcal{C}}(S) &< (1+\epsilon')pt|S| \\ &< (1+\epsilon')(1-\delta)p(n-t)|S| && \text{By Definition 8} \\ &< (1+\epsilon')(1-\delta)\frac{f}{1-f}|S| && \text{By Inequality (1)} \\ &= \frac{(1+\epsilon')(1-\delta)}{(1-f)(1-f-\epsilon)}(1-f-\epsilon)f|S| \end{aligned}$$

From honest majority assumption or Definition 8 we also have $\frac{2f+\epsilon+\epsilon'-f^2-f\epsilon}{1+\epsilon'} < \delta < 1$, which in turn implies $\frac{(1+\epsilon')(1-\delta)}{(1-f)(1-f-\epsilon)} < 1$. This completes the proof of part (d). □

7.3 Chain Growth Properties

Now, we want to prove the chain growth property for ET consensus with z-test. First, we state the modified versions of Lemma 3 and Lemma 4, which now depend on the guarantees of typical execution. The proofs remain essentially the same. However, they crucially depend upon the fact that in a typical execution, honestly generated blocks never get rejected because of the 'z-test'. We note that this property is easy to achieve with what should be fairly typical parameter settinhgs: namely, with $\epsilon \leq \epsilon'$.

Lemma 9. *In a typical execution, let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two chains which were adopted by some honest parties. Suppose B_1 and B_2 are the k-th blocks of chains $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. If $\texttt{id}(B_1)$ is an honest user and $\texttt{round}(B_1)$ is a uniquely successful round, then either $B_1 = B_2$ or $\texttt{id}(B_2)$ is corrupted.*

We skip the proof here because the proof is almost identical to the proof of Lemma 3.

Lemma 10 (Chain Growth Lemma)**.** *In a typical execution, suppose an honest party has adopted a chain of length ℓ at round r. Then, by round $h > r$, every honest party has adopted a chain of length at least $\ell + \sum_{i=r}^{h-1} HON_i^{\geq 1}$.*

Again, this proof is almost identical to the proof of Lemma 4, so we omit it here.

Lemma 11 (Chain Growth Upper Bound)**.** *Suppose $\mathcal{C}$ is a chain adopted by an honest party during a typical execution. For any $k \geq \max(2\lambda f, 4)$, let $B_m, B_{m+1}, \cdots, B_{m+k-1}$ be k consecutive blocks of the chain $\mathcal{C}$. Then, we have*

$$|[\texttt{round}(B_m), \texttt{round}(B_{m+k-1})]| \geq \frac{k}{2f}.$$

Proof. Suppose $S' = [\texttt{round}(B_m), \texttt{round}(B_{m+k-1})]$. For contradiction let us assume $|S'| < \frac{k}{2f}$. Consider the set S of consecutive rounds such that $S \supseteq S'$ and $|S| = \lceil \frac{k}{2f} \rceil$. Security of the signature scheme along with the fact chain $\mathcal{C}$ has been adopted by an honest party ensures $HON^{\geq 1}(S') + ADV_{\mathcal{C}}(S') \geq k$. As $S' \subseteq S$, this in turn implies $HON^{\geq 1}(S) + ADV_{\mathcal{C}}(S) \geq k$. As, $|S| \geq \lambda$, we can apply Lemma 8 and it implies the following.

$$
\begin{aligned}
&HON^{\geq 1}(S) + ADV_{\mathcal{C}}(S) \\
&< (1+\epsilon)f|S| + (1-f-\epsilon)f|S| \\
&< (2-f)f|S| \\
&< (2-f)f(\frac{k}{2f}+1) && \text{Since } |S| = \lceil \frac{k}{2f} \rceil < \frac{k}{2f} + 1 \\
&\leq k - \frac{kf}{2} + 2f - f^2 < k + f(1-k/4) < k. && \text{Since } k \geq 4
\end{aligned}
$$

This shows we have a contradiction. □

Lemma 11 provides us an upper limit on the rate of chain growth. It says that at least $\frac{k}{2f}$ rounds are required for a valid chain to grow by k blocks. Additionally, note that, for $f > 0.5$ the number of rounds to generate k blocks becomes less than k, which is not possible because multiple blocks in the same round will only increase forks, not the chain length. That necessarily means any $f > 0.5$ will not improve the chain growth, instead only increase the fork rate. That is why we should always consider $f \leq 0.5$.

A corollary to chain growth lemma(Lemma 10), Lemma 8 and Lemma 11 is the following theorem.

Theorem 9 (chain-growth)**.** *In a typical execution, the chain growth property holds with parameters $\tau = (1-\epsilon)f$, $\sigma = 2f$ and $r_g > \lambda$.*

The above theorem provides an upper bound as well as a lower bound on the total number of blocks added to a chain $\mathcal{C}$ given a sequence of rounds S with a length $s > r_g$. For s rounds, the number of blocks x added to the chain $\mathcal{C}$ is upper bounded by σs and lower bounded by τs.

7.4 Common Prefix Property

In this section we prove the common prefix property for our ET consensus with z-test protocol. This proof requires us to deviate from our proof for ET consensus with trusted timer pretty substantially (and thus, also deviate from the bitcoin backbone analysis [GKL15]).

The main issue is the following: In bitcoin and in ET consensus with trusted timer, we can bound the total number of blocks an adversary can produce (with some probability, of course). However, in our ET consensus with z-test protocol, the z-test only allows us to bound the number of blocks *per chain*. So an adversary could create a theoretically infinite number of chains and generate blocks on all of them. This would result in a huge total number of blocks created.

It turns out, though, that this per-chain restriction is actually pretty strong. All of our proofs rely on proofs of contradiction using chain comparisons–so we only ever need to work with two chains at a given time. In the bitcoin backbone analysis and our analysis of the ET consensus with trusted timer, we need to assume the fact that honest parties can "outmine" the adversary overall by some margin. In our proof of ET consensus with z-test, we need to assume that the honest parties can produce more blocks than the adversary *on both chains in our analysis*, effectively doubling the power of the adversary. This means we lose a factor of 2 compared to all of our earlier proofs, but can still prove security in a very adversarially tuned setting.

Below we present the formal proofs.

Lemma 12 (Common Prefix Lemma)**.** *In a typical execution, for two chains $\mathcal{C}_1$ and $\mathcal{C}_2$ with $\texttt{len}(\mathcal{C}_2) \geq \texttt{len}(\mathcal{C}_1)$, if $\mathcal{C}_1$ is adopted by an honest party at round r, and $\mathcal{C}_2$ is either adopted by an honest party or broadcasted by an honest party at round r, then $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$ and $\mathcal{C}_2^{\lceil k} \preceq \mathcal{C}_1$, for all $k \geq \max(2\lambda f, 4)$.*

Proof. Let us assume, for contradiction, there exists a $k > 2\lambda f$ such that $\mathcal{C}_1^{\lceil k} \not\preceq \mathcal{C}_2$ or $\mathcal{C}_2^{\lceil k} \not\preceq \mathcal{C}_1$. Suppose, B^* be the last block on the common prefix of $\mathcal{C}_1$ and $\mathcal{C}_2$ such that $\texttt{id}(B^*)$ is honest. Let us denote $\texttt{round}(B^*) = r^*$. Note, B^* can be genesis block, in which case $r^* = 0$.

Now, we define $S = \{i : r^* < i < r\}$. Suppose, $B_m, B_{m+1}, \cdots, B_{m+k'-1}$ are k' consecutive blocks of the chain $\mathcal{C}_1$, where B_m is the next block after $B*$ and $B_{m+k'-1}$ is the last block of $\mathcal{C}_1$. Clearly, $k' \geq k \geq \max(2\lambda f, 4)$ and we can apply Lemma 11. This implies, $|[\texttt{round}(B_m), \texttt{round}(B_{m+k'-1})]| \geq \frac{k'}{2f} \geq \lambda$. We also know, $S \supseteq [\texttt{round}(B_m), \texttt{round}(B_{m+k'-1})]$. Hence, $|S| \geq \lambda$ (i.e., the execution during S is a typical execution with overwhelming probability) and Lemma 8 applies for the set of rounds S.

For a uniquely successful round $u \in S$, let j_u be the position at which the uniquely successful honest party created the block. J be the set of positions at which honest parties created the blocks on uniquely successful rounds. $J = \{j_u : u \in S, HON_u^1 = 1\}$. Suppose the maximum value of the set J is $\max(J)$. Then, $\texttt{len}(\mathcal{C}_1) \geq \max(J)$, since $\mathcal{C}_1$ is adopted by an honest party at round r, by which the honest party has already received a chain of length $\max(J)$.

Since, $\texttt{len}(\mathcal{C}_2) \geq \texttt{len}(\mathcal{C}_1)$, j^{th} block exists in both the chains $\mathcal{C}_1$ and $\mathcal{C}_2$ for all $j \in J$. We denote such blocks by $B_{1,j}$ and $B_{2,j}$ respectively. Now, we want to claim for all $j \in J$ at least one of the players between $\texttt{id}(B_{1,j})$ and $\texttt{id}(B_{1,j})$ is corrupted. By Lemma 9, if both $\texttt{id}(B_{1,j})$ and $\texttt{id}(B_{1,j})$ are honest then we must have $B_{1,j} = B_{2,j}$. Cryptographic strength (collision resistance) of the hash function implies $B_j = B_{1,j} = B_{2,j}$ belongs to the common prefix of chains $\mathcal{C}_1$ and $\mathcal{C}_2$. However, we also know $\texttt{round}(B_j) > \texttt{round}(B^*)$ and B^* is the last block in the common prefix such that $\texttt{id}(B^*)$ is honest. This implies a contradiction.

Now, we have established the fact that for all $j \in J$ at least one of the players between $\texttt{id}(B_{1,j})$ and $\texttt{id}(B_{2,j})$ is corrupted. Hence, total number of blocks B such that $\texttt{id}(B)$ is corrupted, $B \in \mathcal{C}_1 \cup \mathcal{C}_2$ and $\texttt{round}(B) \in S$ must be more than or equal to size of set J. Hence,

$$\begin{aligned} ADV_{\mathcal{C}_1}(S) + ADV_{\mathcal{C}_2}(S) &\geq |\{B : B \in \mathcal{C}_1 \cup \mathcal{C}_2 \text{ and } \texttt{round}(B) \in S\}| \\ &\geq |J| = HON^1(S). \end{aligned}$$

However, for a typical execution with $|S| \geq \lambda$, by Lemma 8 we have

$$ADV_{\mathcal{C}_1}(S), ADV_{\mathcal{C}_2}(S) < \frac{HON^1(S)}{2}.$$

Hence, contradiction. Therefore, we can say that for all $k > 2\lambda f$, it holds that $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$ and $\mathcal{C}_2^{\lceil k} \preceq \mathcal{C}_1$. □

Intuitively, if $\mathcal{C}_1^{\lceil k} \not\preceq \mathcal{C}_2$ or $\mathcal{C}_2^{\lceil k} \not\preceq \mathcal{C}_1$, the number of adversarial blocks for both the chains combined is more than the total number of honest blocks, for the parts of the chains where they don't have a common honest block. And that is not possible for a typical execution, because the number of adversarial blocks for a chain $\mathcal{C}$ during a sequence of rounds S is limited by $ADV_{\mathcal{C}}(S) < \frac{HON^1(S)}{2}$.

Common Prefix Lemma shows that the honest parties eventually agree on a common chain. Once a transaction is included in a block B, the transaction becomes irreversible once honest parties have mined enough number of blocks extending after B. The common prefix lemma directly implies the following security theorem about the common prefix property.

Theorem 10 (Common Prefix)**.** *In a typical execution the common prefix property holds with parameter* $\ell_{cf} \geq \max(2\lambda f, 4)$.

7.5 Chain Quality Property

Theorem 11 (Chain Quality)**.** *In a typical execution, the chain quality property holds with parameters* $\ell_q \geq \max(2\lambda f, 4)$ *and* $\mu = 1 - \frac{(1+\epsilon')t}{(n-t)(1-f)(1-\epsilon)}$ *for any chain adopted by any honest party.*

Proof. Let us consider a chain $\mathcal{C}$, which has been adopted by an honest party P at round r, such that $\texttt{len}(\mathcal{C}) > \ell_q$. Suppose $\mathcal{C}$ consists of sequence of blocks $(B_1, B_2, \ldots, B_{\texttt{len}(\mathcal{C})})$ and $(B_u, B_{u+1}, \ldots, B_{u+\ell_q-1})$ is an arbitrary ℓ_q length subsequence of $\mathcal{C}$, such that $\ell_q \geq \max(2\lambda f, 4)$.

Let $(B_{u'}, B_{u'+1} \ldots, B_{u'+L-1})$ be the shortest subsequence of $\mathcal{C}$ containing $(B_u,\ B_{u+1}, \ldots, B_{u+\ell_q-1})$ (i.e. $u' \leq u$ and $L \geq \ell_q$) such that:

1. $\texttt{id}(B_{u'})$ is honest
2. there exists an honest party which adopted the chain $(B_1, B_2, \ldots, B_{u'+L-1})$

Observe that B_1 is genesis block and $\texttt{id}(B_1)$ is honest by definition. We know that an honest party P adopted the chain $\mathcal{C}$ and $\texttt{len}(\mathcal{C}) > \ell_q$. Hence, the whole chain $\mathcal{C}$ trivially satisfies the above properties, except it might not the shortest one. This shows existence of the shortest subsequence $B_{u'}, B_{u'+1} \ldots, B_{u'+L-1}$.

Suppose, $r_1 = \texttt{round}(B_{u'})$ and the earliest round at which the chain $(B_1, B_2,\ \ldots, B_{u'+L-1})$ got adopted by an honest party is r_2. Let S be the sequence of rounds defined as $S = \{r : r_1 \leq r < r_2\}$. Observe that

$$S \supseteq [\texttt{round}(B_{u'}), \texttt{round}(B_{u'+L-1})] \supseteq [\texttt{round}(B_u), \texttt{round}(B_{u+\ell_q-1})].$$

Hence, by Lemma 11, we have $|S| \geq \ell_q/2f \geq \lambda$ and the properties of typical execution are applicable (Lemma 8) for the set of rounds S.

Let x be the number of honest blocks in the ℓ_q length sequence. In other words $x = |\{B \in (B_u, B_{u+1}, \ldots, B_{u+\ell_q-1}) | \texttt{id}(B) \text{ is honest}\}|$. For contradiction, we assume the chain quality property does not hold for this ℓ_q length sequence of blocks $(B_u, B_{u+1}, \ldots, B_{u+\ell_q-1})$. Hence,

$$x < \mu\ell_q \leq \mu L \tag{9}$$

As the chain $(B_1, B_2, \ldots, B_{u'+L-1})$ got adopted by an honest party in round r_2; for all $i \in [u', u'+L-1]$ we have $\texttt{round}(B_i) \in S$. As $[u, u+\ell_q-1] \subseteq [u', u'+L-1]$, we have

$$\begin{aligned} ADV_{\mathcal{C}}(S) &\geq |\{B \in (B_u, B_{u+1}, \ldots, B_{u+\ell_q-1}) | \texttt{id}(B) \text{ is corrupted}\}| \\ &= L - x > (1-\mu)L \end{aligned} \tag{10}$$

The last inequality uses inequality (9), which is our contradiction assumption. Now, Lemma 10 implies $u' + L - 1 \geq u' + HON^{\geq 1}(S)$ or equivalently $L > HON^{\geq 1}(S)$. Hence inequality (10) can be rewritten as $ADV_{\mathcal{C}}(S) > (1-\mu)HON^{\geq 1}(S)$. As we have seen before, $|S| \geq \lambda$. Hence, by Lemma 8

$$\begin{aligned} (1-\mu)HON^{\geq 1}(S) &= \frac{(1+\epsilon')t}{(n-t)(1-f)(1-\epsilon)} HON^{\geq 1}(S) \\ &> ADV_{\mathcal{C}}(S) \end{aligned}$$

Therefore we have our desired contradiction $ADV_{\mathcal{C}}(S) > ADV_{\mathcal{C}}(S)$. □

The chain quality property guarantees that there will be at least $\mu\ell_q$ honest blocks given a chain of length ℓ_q. For example, when 20% of the miners are dishonest, $\epsilon' = \epsilon = 0.2$, and $f = 0.2$, we have $\mu = 0.53$ — which means at least 53% blocks in the chain are honest blocks. We refer to Table 3 for more examples.

Theorem 12 (Security of ET Consensus with Z-test)**.** *For a security parameter λ, total number of parties $n \in poly(\lambda)$, and given the honest majority assumption defined in Definition8, suppose the following parameter inequalities hold:*

- $\ell_{cf} \geq \max(2\lambda f, 4)$
- $r_g \geq \lambda$, $\tau = (1-\epsilon)f$, $\sigma = 2f$
- $\ell_q \geq \max(2\lambda f, 4)$, $\mu = 1 - \frac{(1+\epsilon')t}{(n-t)(1-f)(1-\epsilon)}$.

Then it is the case that no efficient adversary can win the blockchain security game *as defined in Definition 4 with more than negl* (λ) *probability.*

Proof. Follows directly from Theorem 9, Theorem 10 and Theorem 11. □

8 Discussion and Practical Application

In this section we discuss how to instantiate our protocol and some of the implications of particular parameter settings. We note that our parameter choices here are *theoretical* in nature and reflect provable security properties. In practice, most blockchain system parameters are only set to avoid known attacks (e.g. assuming that bitcoin consensus is final 6 blocks deep into the cahin), so practical systems will typically have much better performance than is implied by these figures.

8.1 When n, t are variable

We prove the security considering fixed n, t — which parameters tend to change over time. If n and t changes, f also changes, for a chosen constant value of p. Therefore, in real life, the protocol needs to periodically estimate n and update the parameters of the probability distribution(and p) such that the probability of a successful round is upper bounded by f. However, the probability should not be really small compared to f because in that case the chain growth will be too slow. Therefore, the period has to be long enough ($>> \max(2\lambda f, 4, \lambda)$) for the security to hold, but not too long that can hurt performance. The exact procedure of calculating p is out of scope of this paper; the security holds as long as f follows the honest majority assumption, independent of the exact method to calculate p.

8.2 Parameter Choices

We want to set the z-test parameter ϵ' in such a way that an honest block is excluded from a chain only with negligible probability. If we can ensure that an honest block is never excluded by the z-test during a typical execution, that automatically implies the above. Therefore, we recommend $\epsilon = \epsilon'$ — that is the minimum value of ϵ' which ensures that no honest block is excluded by z-test for a typical execution. With $\epsilon' = \epsilon$, the typical execution definition and theorem will reduce to following:

Definition 10 (Typical Execution)**.** *An execution of r_{end} rounds, is (ϵ, λ)-typical, for some $\epsilon \in (0,1)$, if for any set S of at least λ consecutive rounds the followings hold:*

(a) $(1-\epsilon)\mathbb{E}[HON^{\geq 1}(S)] < HON^{\geq 1}(S) < (1+\epsilon)\mathbb{E}[HON^{\geq 1}(S)]$ *and* $(1-\epsilon)\mathbb{E}[HON^{1}(S)] < HON^{1}(S)$

(b) For all honest players i, $HON_{.,i}(S) < (1+\epsilon)\mathbb{E}[HON_{.,i}(S)]$

Theorem 13. *An execution of r_{end} rounds is (ϵ, λ)-typical with probability at least $1 - r_{end}(e^{-\Omega(\epsilon^2 \lambda f)}$.*

This simplifies our honest majority assumption as well as the security theorem related to the chain quality property, as the following:

Definition 11 (Simplified Honest Majority Assumption)**.** *Suppose n is the total number of parties, and out of them t parties are corrupted. Then we require that $t < (1-\delta)(n-t)$,*
where $\max\left(\frac{(2-f)(f+\epsilon)}{1+\epsilon}, \frac{1+3\epsilon-4\epsilon f}{2(1+\epsilon)(1-f)}\right) < \delta \leq 1$, $0 < f \leq 0.5$, *and* $\epsilon \in (0,1)$.

Theorem 14 (Chain Quality)**.** *In a typical execution, the chain quality property holds with parameters* $\ell_q \geq \max(2\lambda f, 4)$ *and* $\mu = 1 - \frac{(1+\epsilon)t}{(n-t)(1-f)(1-\epsilon)}$ *for any chain adopted by any honest party. This assumes cryptographic strength of the hash function and the signature scheme.*

The security theorem related to the chain growth property and the common prefix property remain the same. Our main security theorem is modified as following:

Theorem 15 (Simplified Security of ET Consensus z-test)**.** *For a security parameter* λ*, total number of parties* $n \in poly(\lambda)$*, and given the honest majority assumption defined in Definition 8, suppose the following parameter inequalities hold:*

- $\ell_{cf} \geq \max(2\lambda f, 4)$
- $r_g \geq \lambda$, $\tau = (1-\epsilon)f$, $\sigma = 2f$
- $\ell_q \geq \max(2\lambda f, 4)$, $\mu = 1 - \frac{(1+\epsilon)t}{(n-t)(1-f)(1-\epsilon)}$.

Then it is the case that no efficient adversary can win the blockchain security game *as defined in Definition 4 with more than* $negl(\lambda)$ *probability.*

In table 3 we show some examples with possible values of ϵ, f, δ and how the parameters $\tau, \sigma, \ell_{cf}, \mu$ corresponding to the security properties(namely, chain growth, common prefix and chain quality properties) vary. It shows that we can vary $(f + \epsilon)$ up to 1, when $\delta = 1$ (which means all the protocol parties are honest). For $\epsilon = 0.2$ and $f = 0.2$, δ can be as low as 0.75, which means the protocol can tolerate up to 20% dishonest protocol parties. For carefully chosen small values of ϵ and f, ET consensus with z-test can tolerate up to 33% dishonest protocol parties.

Table 3: How the security property parameters $(\tau, \sigma, r_g, \ell_{cf}, \ell_q, \mu)$ of ET consensus with z-test corresponding to chain growth, common prefix and chain quality properties vary based on the protocol parameters $\delta, \epsilon, f, \lambda$. The first column presents the values of ϵ and f defined in the honest majority assumption, the second column the minimum δ (accurate up to two decimal places) to satisfy the honest majority assumption; the third, fourth, fifth, sixth, seventh, and eighth columns are τ, σ, r_g, ℓ_{cf}, ℓ_q and μ respectively as defined in Theorem 12. For all the cases, we use $\epsilon' = \epsilon$, and $\lambda >> 4$.

Protocol parameters	δ	τ	σ	r_g	ℓ_{cf}	ℓ_q	μ
$\epsilon = 0.05, f = 0.05$	0.58	0.0475	0.1	λ	0.1λ	0.1λ	0.51
$\epsilon = 0.1, f = 0.1$	0.64	0.09	0.2	λ	0.2λ	0.2λ	0.51
$\epsilon = 0.2, f = 0.2$	0.75	0.16	0.4	λ	0.4λ	0.4λ	0.53
$\epsilon = 0.3, f = 0.3$	0.85	0.21	0.6	λ	0.6λ	0.6λ	0.60
$\epsilon = 0.4, f = 0.3$	0.88	0.18	0.6	λ	0.6λ	0.6λ	0.6
$\epsilon = 0.4, f = 0.4$	0.93	0.24	0.8	λ	0.8λ	0.8λ	0.72
$\epsilon = 0.5, f = 0.4$	0.95	0.2	0.8	λ	0.8λ	0.8λ	0.65
$\epsilon = 0.6, f = 0.4$	0.96	0.16	0.8	λ	0.8λ	0.8λ	0.73
$\epsilon = 0.5, f = 0.5$	1	0.25	1	λ	λ	λ	1

Table 4: Relationship between f and minimum δ in ET consensus with trusted timer, ET consensus with z-test

f	ET with Trusted Timer $\delta_{\min}$	ET with z-test $\delta_{\min}$
0.05	0.3	0.58
0.1	0.6	0.64

8.3 Implications of z-test Security

In addition to lending evidence to support that the actual PoET protocol (and other similar protocols like proof of luck) is resilient to the compromise of some TEEs, we show a pretty surprising fact: basic proof of work consensus with a z-test *but no actual proofs of work, just "promises" from users* still remains secure with an honest majority assumption! Table 4 shows that ET protocol without TEE is not terribly worse (in terms of security) than ET with trusted timer (and hence bitcoin backbone protocol).

While these numbers are not incredibly tight, the δ factors indicate that our proofs hold even when the number of adversarial parties is a (relatively large) constant fraction (up to 33%) of the total number of players. This indicates that current TEE-based consensus systems like PoET that are used "in the wild" are, at least in theory, secure, although we would need to change the z-test in PoET in order for our proofs to apply.

8.4 Applications of Our Protocol

In [GKL15], the authors show that the bitcoin backbone protocol almost immediately implies a Byzantine fault tolerant consensus protocol and a public ledger. The same results apply to our protocols, so we omit the full proofs and descriptions here. An inquisitive reader can refer to sections 5 and 6 of [GKL15].

9 Performance Improvement

Suppose we wanted to improve the efficiency of the elapsed time consensus protocol. In our elapsed time consensus with trusted timer protocol and in proof of luck, players must constantly query the TEE to see if the wait time has elapsed. From a practical perspective, this must be done at regular intervals of, say, approximately half of the network latency time. If we wanted to rigorously prove security over continuous time (i.e. not using a round-based protocol) then we might have to query even more, perhaps even continuously. This is obviously very inefficient. In our round-based protocols, this means the TEE must be queried at least once per round. In practice, this might mean a query a second, or hundreds of TEE queries per block.

Since, in the context of z-test, we do not consider the TEE to be secure in any way, a player does not need to wait on the TEE to generate a block. Instead, an honest player can just query the wait time and corresponding wait certificate from the TEE, and construct the block by themselves. In figure 15 we present our optimized version of ET consensus protocol with z-test.

In the optimized version of the protocol, instead of the player repeatedly querying the TEE, the TEE issues a $WaitCertificate$. This $WaitCertificate$ indicates how long a player has to wait before the TEE would release the block. In other words, the protocol allows players to know how long it will take for their "mining" efforts to succeed. This means that players can wait to query the TEE until their wait time is up, at which point the TEE will issue a block. The number of TEE queries goes down to two per block, which is very good.

The above optimization/modification does not change anything for the security proofs, and therefore, the security properties described in Section 7 are valid for this version as well.

10 Conclusion and Future Work

In this work, we defined and proved the security of our elapsed time consensus protocol both with and without trusted execution environments. This provides evidence for the security of the PoET protocol and other related TEE-based protocols like proof of luck even when some of the TEEs are compromised. In addition, our results imply that "bitcoin without mining"–where players wait for a certain randomly

$\mathcal{C} := empty$ // A chain - an ordered list of blocks
$playerID$; // denotes the index of the player provided by the challenger
$WaitTime := \infty$
$remainingWaitTime := \infty$
$WaitCert := null$ // denotes the certificate validating $WaitTime$

Initialize(integer i, the genesis block $B_{genesis}$):
$playerID \leftarrow i$; $TEE.Initialize(i)$; Add $B_{genesis}$ to $\mathcal{C}$;
$TEE.Run(GetWaitTime(), \mathcal{C})$
$(WaitTime, \cdot, WaitCert) \leftarrow TEE.Poll()$
$WaitTime \leftarrow remainingWaitTime$

RunPlayer():
Check $PLAYER\_QUEUE[playerID]$ for all new received chains.
// The player has received a chain from some other player
if QUEUE is not empty **then**
$\mathcal{C}_{new} \leftarrow PickChain(playerID)$
if $\mathcal{C}_{new} \neq \mathcal{C}$ **then**
$\mathcal{C} \leftarrow \mathcal{C}_{new}$; $r \leftarrow$ current round number
$TEE.Run(GetWaitTime(), \mathcal{C})$
$(waitTime, \cdot, WaitCert) \leftarrow TEE.Poll()$
$remainingWaitTime \leftarrow waitTime + \mathcal{C}.head.\pi.timestamp - r$
end if
else
// The player has NOT received a chain from some other player
$remainingWaitTime \leftarrow remainingWaitTime - 1$
if $remainingWaitTime = 0$ **then**
// It seems the player is a leader
$x \leftarrow$ Collected transactions from users
$B \leftarrow CreateBlock(waitTime, \mathcal{C}, x)$, $B.\pi.WaitCert = WaitCert$
Add B to $\mathcal{C}$; $Broadcast(\mathcal{C})$
// Again, repeat for the next leader election
$TEE.Run(GetWaitTime(), \mathcal{C})$
$(WaitTime, \cdot, WaitCert) \leftarrow TEE.Poll()$
$WaitTime \leftarrow remainingWaitTime$
end if
end if

GetWaitTime(chain $\mathcal{C}$):
if (TEE.GetCounterValue() $\geq Length(\mathcal{C})$) **then** return $\perp$
else TEE.CounterSet($Length(\mathcal{C})$) **end if**
$waitTime \leftarrow$ draw an element from P
return $(waitTime, \mathcal{H}(\mathcal{C}.head))$

Figure 15: Ideal functionality of performance optimized ET protocol with z-test

sampled amount of time before publishing a block–is secure in a permissioned environement. This, to us, is a surprising result that may have many applications in distributed consensus.

However, there are many interesting topics for future research. We explain some of these below.

Tighter Proofs. The proofs in our work for ET consensus with a z-test seem (at least to us) to be very loose. Tightening these proofs would be useful for practitioners using similar protocols to know exactly what they could assume about the security of their systems.

Model and Protocol Improvements. There are many potential ways our framework could be improved. Adding in dynamic participation and putting the protocol in the UC framework are two adaptations that would improve our results. In [GKL17], the authors added dynamic participation and some other improvements to the bitcoin backbone protocol, and in [BMTZ17], the authors prove bitcoin secure in the UC model. Corresponding results for our model would be nice advancements (and may encompass these previous works as well).

Improved Wait Time Distributions. Our protocol only works for geometric wait time distributions. This is mostly just a proof artifact, as using a non-geometric distribution would substantially complicate the proof. It might be the case that a non-geometric wait time distribution can substantially outperform traditional geometric distributions, but, to our knowledge, no research even has been done on the performance of TEE-based blockchain consensus algorithms with non-geometric wait times. We think that this is an exciting area for future work.

Better Scalable Blockchain Algorithms. As we live in an increasingly interconnected world, we will need distributed consensus protocols that can handle large numbers of participants. Blockchain protocols that efficiently scale solve many of these problems, so we think that future work in this direction could be very rewarding.

References

[ABB+18] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM, 2018.

[ABK+18] Sbastien Andreina, Jens-Matthias Bohli, Ghassan O. Karame, Wenting Li, and Giorgia Azzurra Marson. Pots - a secure proof of tee-stake for permissionless blockchains. Cryptology ePrint Archive, Report 2018/1135, 2018. `https://eprint.iacr.org/2018/1135`.

[AGJS13] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent 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, 2013.

[AMNR18] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, and Ling Ren. Dfinity consensus, explored. *IACR Cryptology ePrint Archive*, 2018:1153, 2018.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Advances in Cryptology – CRYPTO 2018, Part I*, pages 757–788, 2018.

[BCNPW18] Jonah Brown-Cohen, Arvind Narayanan, Christos-Alexandros Psomas, and S. Matthew Weinberg. Formal barriers to longest-chain proof-of-stake protocols. *CoRR*, abs/1809.06528, 2018.

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017.

[BG18] J Benet and N Greco. Filecoin: A decentralized storage network. *Protoc. Labs*, 2018.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology – CRYPTO 2001*, volume 2139, pages 1–18, 2001.

[BGM16] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *FC 2016 Workshops*, volume 9604, pages 142–157, 2016.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401, pages 324–356, 2017.

[BSA14] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. State machine replication for the masses with bft-smart. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362. IEEE, 2014.

[But18] V Buterin. Ethereum 2.0 spec–casper and sharding, 2018.

[Cac16] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, volume 310, 2016.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[CL+99] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[Cor19] Amie Corso. Performance analysis of proof-of-elapsed-time (poet) consensus in the sawtooth blockchain framework. 2019.

[CP19] Bram Cohen and Krzysztof Pietrzak. The chia network blockchain, 2019.

[CV17] Christian Cachin and Marko Vukolić. Blockchains consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

[CXS+17] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In *Stabilization, Safety, and Security of Distributed Systems*, pages 282–297, 2017.

[DDL+19] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 International Conference on Management of Data*, pages 123–140, 2019.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216, pages 585–605, 2015.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology – EUROCRYPT 2018, Part II*, pages 66–98, 2018.

[dig] Bitcoin engergy consumption index.

[DPS16] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. Technical report, Cryptology ePrint Archive, Report 2016/919, 2016., 2016.

[DR85] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *J. ACM*, 32(1):191–204, January 1985.

[EGSVR16] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *NSDI*, pages 45–59, 2016.

[EKA13] Jan-Erik Ekberg, Kari Kostiainen, and N. Asokan. Trusted execution environments on mobile devices. In *Proceedings of the 2013 ACM SIGSAC conference on Computer communications security*, CCS '13, pages 1497–1498, 2013.

[ES18] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM*, 61(7):95–102, June 2018.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology – EUROCRYPT 2015, Part II*, pages 281–310, 2015.

[GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *Advances in Cryptology – CRYPTO 2017, Part I*, pages 291–323, 2017.

[GKW+16] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[Int13] Intel. Software Guard Extensions (Intel SGX) Programming Reference, 2013. `https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf`.

[KHF+19] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[KJG+16] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, 2016.

[KKSAG18] Esmaeil Mohammadian Koruyeh, Khaled N Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. Spectre returns! speculation attacks using the return stack buffer. In *12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18)*, 2018.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology – CRYPTO 2017, Part I*, pages 357–388, 2017.

[Kwo14] Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 2014.

[LLKA19] J. Liu, W. Li, G. O. Karame, and N. Asokan. Scalable byzantine consensus via hardware-assisted secret sharing. *IEEE Transactions on Computers*, 68(1):139–151, Jan 2019.

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 973–990, 2018.

[Mad88] James Madison. Federalist no. 51. *The Federalist Papers*, 1788.

[Maz15] David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, 2015.

[MHWK16] Mitar Milutinovic, Warren He, Howard Wu, and Maxinder Kanwal. Proof of luck: An efficient blockchain consensus protocol. In *Proceedings of the 1st Workshop on System Software for Trusted Execution*, page 2. ACM, 2016.

[Nak] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf.

[NKMS16] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 305–320. IEEE, 2016.

[pla] Trusted time and monotonic counters with intel software guard extensions platform services. `https://software.intel.com/sites/default/files/managed/1b/a2/Intel-SGX-Platform-Services.pdf`.

[poe] Poet 1.0 specification. https://sawtooth.hyperledger.org/docs/core/releases/1.0/architecture/poet.html.

[PS17a] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 315–324. ACM, 2017.

[PS17b] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *Advances in Cryptology – ASIACRYPT 2017, Part II*, pages 380–409, 2017.

[PS18] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Advances in Cryptology – EUROCRYPT 2018, Part II*, pages 3–33, 2018.

[sal] Salesforce breaks the martech ceiling; dashes ahead with new blockchain crm. https://martechseries.com/mts-insights/staff-writers/salesforce-dashes-ahead-with-new-blockchain-crm/.

[saw] Introduction to hyperledger sawtooth. https://sawtooth.hyperledger.org/docs/core/releases/latest/introduction.html.

[sex] Btp sextant. https://blockchaintp.com/sextant/.

[SVCNB+13] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. Efficient Byzantine fault-tolerance. *IEEE Transactions on Computers*, 62:16–30, 2013.

[SZH+19] Zeshun Shi, Huan Zhou, Yang Hu, Surbiryala Jayachander, Cees de Laat, and Zhiming Zhao. Operating permissioned blockchain in clouds: A performance study of hyperledger sawtooth. In *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 50–57. IEEE, 2019.

[tas] Tel aviv exchange unveils blockchain platform. https://www.investopedia.com/news/tel-aviv-exchange-unveils-blockchain-platform/.

[tru09] ARM security technology: Building a secure system using TrustZone technology. White paper, ARM, 2009.

[TZL+16] Florian Tramèr, Fan Zhang, Huang Lin, Jean-Pierre Hubaux, Ari Juels, and Elaine Shi. Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 19–34, 2016.

[VMW+18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 991–1008, 2018.

[VO99] P. Valduriez and M. T. Ozsu. *Principles of Distributed Database Systems*. Prentice Hall, 1999.

[Vuk15] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International Workshop on Open Problems in Network Security*, pages 112–125. Springer, 2015.

[YMR+19] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356. ACM, 2019.

A Appendix: Additional Lemmas and Proofs

Lemma 13. *If the range $\mathcal{R} = [0, \infty)$ is divided into intervals of width δ each, suppose the intervals are denoted with $I_1 = [0, \delta), I_2 = [\delta, 2\delta), \ldots, I_j = [(j-1) \cdot \delta, j \cdot \delta), \ldots,$ for a sample t from exponential distribution,*
$Pr[t \in I_j | t \geq (j-1) \cdot \delta] = Pr[t \in I_{j-1} | t \geq (j-2) \cdot \delta)] = \cdots = Pr[t \in I_1]$.

Proof. According to memoryless property of exponential distribution,

$Pr[t > s + \delta | t > s] = Pr[t > \delta]$

$\iff 1 - Pr[t > s + \delta | t > s] = 1 - Pr[t > \delta]$

$\iff Pr[t \leq s + \delta | t > s] = Pr[t \leq \delta]$

$\iff Pr[t < s + \delta | t \geq s] = Pr[t < \delta]$ [continuous distribution]

And the above is true for any chosen value of s, and hence, is true for $s = i \cdot \delta$, given any positive integer i. □