b
UNIVERSITÄT
BERN

PoET: an eco-friendly alternative to PoW

A security analysis

Bachelor Thesis

Marius Paulius Asadauskas

from

Bern, Switzerland

Faculty of Science, University of Bern

30. July 2021

Prof. Christian Cachin
Ignacio Amores Sesar
Cryptology and Data Security Group
Institute of Computer Science
University of Bern, Switzerland

Abstract

Proof of Elapsed Time (PoET) was first proposed by Intel in 2016 as a possible alternative to Proof of Work (PoW), which to this day remains responsible for drastic amounts of energy consumption worldwide. With the main idea of waiting instead of working it promises a much more efficient consensus mechanism, while the security is enforced by a Trusted Execution Environment and a statistical z-test. In the following thesis, we perform a security analysis on PoET's statistical z-test with the help of a simulation, to observe whether a PoET based blockchain remains secure in a setting where the security of the Trusted Execution Environment can not be guaranteed.

Acknowledgments

First and foremost, I would like to thank my advisor Ignacio Amores Sesar for supporting and helping me with the many difficulties that come with researching and writing a thesis. I would also like to thank Professor Christian Cachin for the many meetings where he shared his expertise and guided my research. Lastly, I would like to thank my family for their support in these trying times.

Contents

Chapter 1

Introduction

1.1 Bitcoin's energy consumption

To this day, Bitcoin is known to be the first and most prominent cryptocurrency. However, it is less known that the Bitcoin network currently consumes as much electricity as does a small nation. Since Bitcoin is a distributed online ledger, no exact value for its energy consumption is available. But current estimates lie at around 74.18 TWh [1] a year. To put this into perspective, the entire Swiss Confederation consumed around 57.198 TWh [2] in 2019. Furthermore, it should be noted that Bitcoin is not the only energy-intensive blockchain. Cryptocurrencies like Ethereum, Monero, Litecoin and others consume highly excessive amounts of energy as well. Ethereum alone is estimated to consume 52.94 TWh [3] a year. The cause of this high energy consumption can be attributed directly to the consensus mechanisms that these blockchains rely upon.

1.2 Proof of Work

The most common consensus mechanism, which is used by all the aforementioned blockchains, is known as Proof of Work or PoW. As the name implies, PoW is a type of proof that can be verified with minimal effort and shows that a mathematically difficult computation has been performed. The idea to use this in a blockchain network was first proposed by Satoshi Nakamoto in the Bitcoin whitepaper [4]. The Bitcoin implementation of PoW works together with the SHA256 cryptographic hash function and requires for the hash of a block to begin with a certain amount of zeros before it can be broadcast to the network. An example of this can be seen in fig. 1.1 below.

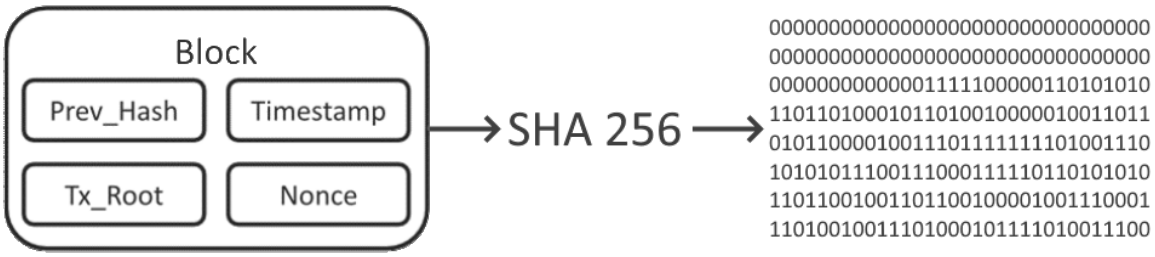


Figure 1.1. Illustrating Example of PoW

The SHA256 cryptographic hash function takes any input data and generates a pseudorandom 256-Bit string as the output. Meaning when two individuals enter the same input, they both receive the same output. However, if a single bit in their input is not the same, then the outputs become completely different from each other.

As the name further implies, the work part stems from the chance of finding a block whose hash begins with so many zeros being severely unlikely. The probability of the first bit being zero is $\approx \frac{1}{2}$, the

probability of the first k bits being zero is $\approx \frac{1}{2^k}$. To find a block like this participants must change the nonce field, which is a random number, until a block is found whose hash begins with k many zeros or more. This process is referred to as mining. The value k is chosen so that on average the network would require 10 minutes of continuous computation to find a new block.

These 10 minutes in between blocks help to ensure that a block has enough time to propagate throughout the network, as well as providing time to perform any other checks on the validity of the block. It also guarantees that blocks can not be pushed in quick succession of each other.

It is also important to note that PoW is not only used in blockchain based technologies but also in other areas such as Denial-of-service attack countermeasures or spam mail prevention [5].

1.3 Eco-friendliness

Although energy consumption itself is not necessarily an issue, it becomes problematic once the energy sources are not eco-friendly. Surveys in this regard find that on average 39% of Proof of Work based mining is powered by renewable energy [6]. Despite this percentage being fairly large compared to any other industry, it still leaves more than half of the energy to be obtained from non-renewable sources. This makes the environmental impact of Proof of Work based mining difficult to deny.

1.4 Proof of Elapsed Time

Proof of Elapsed Time, also known as PoET, is a lottery-based consensus mechanism that was first proposed by Intel in 2016 and implemented into Sawtooth Hyperledger [7] shortly after. With the main idea of waiting instead of working, each node creates a random exponentially distributed waiting time and proceeds to go into a sleep-like state for the given amount of time. Once the waiting time has elapsed, the nodes broadcast their blocks to the rest of the network. Whoever has the shortest waiting time broadcasts their block first and subsequently wins the leadership election. As the leader, they then get their respective block added to the blockchain.

To ensure that the waiting times are generated correctly and that the nodes are waiting for the given amount of time, Sawtooth Hyperledger employs a Trusted Execution Environment, also known as a TEE. This TEE is built into Intel CPUs and is known as Intel's Software Guard Extensions (SGX) [8]. The SGX is an extension to the Intel architecture that provides integrity and confidentiality even on a computer that could potentially contain malicious software. This way the generation of waiting times, the communication with other nodes and the creation of wait certificates can all be entrusted to the TEE instead of relying on the honesty of participants.

As a comparison, if a PoET network is created with a target wait time of 10 minutes, this would simulate the Bitcoin network in a much more efficient manner. Furthermore, we would have a system with the concept of "One CPU one vote", which was the original goal of PoW in the Bitcoin whitepaper [4].

1.5 Security Uncertainty of PoET

The main issue behind the PoET security is the heavy reliance upon the use of a TEE and these are not without fault. There have been multiple kinds of attacks on TEEs in the past [9], [10]. For this reason, Proof of Elapsed Time also implements another measure, namely the use of a statistical z-test. This test is explained in further detail below. To put it briefly, the purpose of the statistical z-test is to mathematically detect adversaries which win too many leadership elections. Hence, even when the security of TEEs can not be guaranteed, the system remains secure to some degree.

1.6 Motivation to Simulate PoET

Due to PoET being designed to expand as little computational effort as necessary, it is possible to simulate an entire PoET network on just a single device. Furthermore, adding adversaries to the simulation would allow the simulation of attacks. Doing so might increase or diminish security concerns.

Chapter 2

Background

In this chapter, we elaborate on PoET and the security analysis that has been performed on it thus far. This is necessary as to understand the reasons behind our research and as to gain a better comprehension of PoET and the calculations surrounding it, such as the wait time generation and others.

2.1 Sawtooth PoET Specification

The PoET Specification [11] contains detailed explanations of how PoET functions and on how to implement the consensus mechanism in a software environment. This specification was used as the main reference when constructing the simulation software, described in the next chapter.

2.1.1 Wait Time Generation

In PoET the nodes create random wait times with the following equation:

$$Duration = minimumDuration - localMean \cdot \log(tagd).$$

$Duration$ is equivalent to the wait time that is supposed to be generated.

$minimumDuration$ is a lower bound for the $Duration$ to prevent it from reaching zero. A system, with a possible wait time of zero, would struggle to prevent the pushing of multiple blocks in quick succession.

$tagd$ stems from converting the lowest 64 bits of the most recent wait certificate tag into a double that lies in $[0, 1]$. In PoET wait certificates are applied in order to prove that a party has waited for a given amount of time. These certificates are created by the TEE and are composed of a wait timer, a nonce field and a hash of their respective block. The nonce field is a random number created by the TEE which ensures, that the wait certificate tag becomes unpredictable. The tag of a wait certificate is generated in the following way:

$$tag = \text{AES-CMAC}_{PoetSealKey}(WaitCertId_n)$$

AES-CMAC is a function that receives a message and a key as an input and outputs an authentication code [12]. It should be noted that in PoET's case the wait certificate is given as the message, while the PoET seal key is given as the key. The output is then used as the tag of the wait certificate.

For each node the Poet seal key is both unique and known only to itself [13], which causes the wait time to differ between nodes. Tag being a random value causes $tagd$ to become a random value in $[0, 1]$. This leads to the expected values of $tagd$ being $E[tagd] = 0.5$, and it follows that

$$E[Duration] = minimumDuration - localMean \cdot \log(0.5) = minimumDuration + localMean.$$

2.1.2 Population Estimate

Although Hyperledger Sawtooth is a permissioned blockchain, it allows dynamic participation. This means that nodes are not forced to take part in the leadership elections if they do not desire to do so. The amount of participants in a network at any given time is therefore unknown to the network and has to be estimated instead. For this reason, PoET calculates a population estimate after each round, to know what kind of local means to use when generating the next wait time. The process of estimating the population size is given in the PoET specification by the pseudocode in alg. 1, which can be seen below.

Algorithm 1 PoET population size computation [11]

```
sm = 0
sw = 0
foreach wait certificate wc stored on the ledger:
    sw = sw + wc.waitTimer.duration - minimumWaitTime
    sm = sm + wc.waitTimer.localMean
populationSize = sm/sw
```

Algorithm 1 outputs the population size because of the manner in which the wait times are created. To show this we first define the following random variable:

$$W_i = Duration - minimumDuration \text{ of node } i$$

W_i is exponentially distributed with the distribution $W_i \sim Exp(1/localMean)$. We define M as the minimum of n many independent exponentially distributed random variables with the parameter $\lambda_1 = 1/localMean$. In other words $M := min(W_1, ..., W_n)$. It follows that M is also exponentially distributed with parameter $\lambda = n/localMean$. In other words $M \sim Exp(n/localMean)$. From the distribution of the random variables, we can conclude that

$$E[W_i] = localMean$$

and

$$E[M] = \frac{localMean}{n} \approx targetWaitTime.$$

So it follows that

$$n = \frac{E[W_i]}{E[M]} \approx \frac{sm}{sw}.$$

2.1.3 Local Mean

As previously mentioned, the local mean is needed for the wait time generation. PoET determines the local mean, so that $E[M]$ would be equal to a desired target time. In Bitcoin's case this target would be 10 minutes, however for PoET this time should be chosen depending on the network size, as to avoid collisions when broadcasting blocks.

The calculation of the local mean is given by the pseudocode in alg. 2 , which can be found below.

Algorithm 2 PoET local mean computation [11]

1. if $b < sampleLength$ then $r = 1.0 * b/sampleLength$ and
   $localMean = targetWaitTime * (1 - r^2) + initialWaitTime * r^2$
2. else $localMean = targetWaitTime * populationSize$

From this it follows that

$$E[M] = (targetWaitTime * populationSize)/n \approx targetWaitTime$$

2.1.4 Z-Test

A statistical z-test is a type of hypothesis test that is used when the data can be approximated by a normal distribution [14]. The hypothesis that PoET desires to test is that an individual has won too many rounds above the expected value. However, to be able to apply a z-test we first need a normal distribution, which can approximation the amount of rounds won by an individual.

For this, we assume that there are n_j nodes taking part in the leadership election during round j. We then define the following indicator random variable:

$$X_{ij} := \begin{cases} 1, & \text{if node } i \text{ wins round } j \\ 0, & \text{otherwise.} \end{cases}$$

The chances of winning should be equal among all nodes, so it follows that

$$P[X_{ij} = 1] = \begin{cases} \frac{1}{n_j}, & \text{if node } i \text{ participated in round } j \\ 0, & \text{otherwise.} \end{cases}$$

As mentioned above, Sawtooth Hyperledger allows dynamic participation. This causes the exact number of nodes taking part in an election at any given time to remain unknown. $P[X_{ij} = 1]$ must be approximated instead and PoET does this by assuming that

$$P[X_{ij} = 1] \approx \frac{1}{\text{populationEstimate in round } j}.$$

We then define the random variable $Z_i := \sum_j X_{ij}$, which gives the amount of blocks won in total by node i. Z_i is the sum of differently distributed Bernoulli random variables. This is known as a Poisson-binomial distribution. Similar to the binomial distribution, the Poisson-binomial distribution can be approximated by a normal distribution, once the amount of rounds is large enough [15].

The expected value of Z_i is equal to

$$\mu = E[Z_i] = E[\sum_j X_{ij}] = \sum_j E[X_{ij}] \approx \sum_j \frac{1}{\text{populationEstimate in round } j}.$$

We then approximate $\sigma^2 \approx npq = \mu(1 - \frac{\mu}{blockCount})$.

Once we have the mean and standard deviation, we approximate the distribution as $Z_i \sim \mathcal{N}(\mu, \sigma^2)$. Now that Z_i can be approximated by a normal distribution, we may perform a statistical z-test on the participants, as to determine how many times the standard deviation, an adversary has won above the expected value. PoET implements this test with the pseudocode seen below in alg. 3.

Algorithm 3 PoET z-test [11]

```
observed = expected = blockCount = 0
foreach b = (id, populationEstimate) in blockArray:
    blockCount += 1
    expected += 1 / populationEstimate

    if id is equal to testIdentifier:
        p = expected / blockCount
        σ = sqrt(blockCount * p * (1.0 - p))
        z = (observed - expected) / σ
        if z > zmax:
            return False
return True
```

2.2 Prior Security Analysis of PoET

PoW is the most widespread consensus protocol and has been extensively researched in regard to security. Contrary to that PoET has had exactly two papers, of which we know, that performed a security analysis. The first paper being by Chen et al. [16] and the second being by Bowman et al. [17]. As to understand the contributions and contradictions of these papers, they will be discussed in further detail below.

2.2.1 On Security Analysis of PoET

The first paper by Chen et al. [16] set out to mathematically analyse how many dishonest nodes would be needed to compromise a PoET system, where the security of the TEE could not be guaranteed. In other words a system where dishonest nodes could win as many leadership elections as they wanted to as long as they did not get caught by the statistical z-test employed in PoET. Chen et al. conclude that "In a PoET system similar to Sawtooth Lake Scheme adversaries can hijack or compromise the whole system if they compromise $\Theta(\frac{log(log(n))}{log(n)})$ fraction of the nodes." [16].

This conclusion paints a rather negative picture for PoET, since once n becomes large enough the fraction becomes far smaller than 50%. With 50% being the fraction of CPUs needed to take over a PoET based network where the TEEs function as intended. Furthermore, it is noteworthy to point out that Chen et al. used an abstraction of PoET in their paper, which may put the accuracy of the conclusion into consideration.

2.2.2 On Elapsed Time Consensus Protocols

The second and most recent paper, which performed a security analysis on PoET, was by Bowman et al. [17]. Contrary to the first paper, the goal was to argue on behalf of PoET's security in a setting where adversaries could win as many leadership elections as they wanted to as long as they did not get caught by the statistical z-test. The paper was structured in a fashion similar to that of the Bitcoin Backbone by Garay et al. [18], which back in 2015 formally defined PoW and proved the security of PoW based systems. Bowman et al. defined a different z-test, which they called an ET-ztest and proceeded to prove that a network relying on said ET-ztest remained secure up to a fraction of 50% dishonest nodes.

Bowman et al. concluded that "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" [17]. By comparison the conclusion of Chen et al. opposes the conclusion of Bowman et al. in a stark manner. For this reason we saw the need to further analyse the security of PoET based systems.

ET-ztest

The following section shows the ET-ztest, which was proposed by Bowman et al. [17].

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

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

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

The primary difference between the two tests is that the ET-ztest does not allow dynamic participation and instead knows how many nodes are participating at any given time.

Chapter 3

Simulation

In the following section, we will explain the structure of our simulation software and what measures were taken to simulate a PoET network accurately, while also keeping efficiency in mind. We will also explain the types of attacks that were simulated with the software.

3.1 Design

To simulate the PoET consensus, we used Python. This choice was made for multiple reasons. Python's robust standard library contains many high-level data structures which we could use without having to spend time implementing them from scratch. Python also gives access to the NumPy package [19], which helped with different types of scientific computations and made the simulation overall faster. In particular NumPy's random methods provided us with a fast way to generate exponentially distributed numbers, which we could then turn into exponentially distributed waiting times for the nodes. NumPy's array methods were also useful, as they helped speed up other important calculations surrounding the PoET consensus such as the z-test.

The structure of the simulation software was then distributed into three parts.

3.1.1 Main Function

First, the main function, which was responsible for creating the nodes, creating the blockchain and managing the communication between the two. It was also responsible for running the leadership election and figuring out how many dishonest nodes had to be controlled by an adversary for an attack to be successful.

Simulating the Blockchain

Before figuring out whether an attack was successful, we first had to simulate the evolution of the blockchain. This process began by initialising the blockchain with a genesis block and then running the leadership election. The winner of the leadership election would become the leader and get to push a block onto the blockchain. This election was repeated until the blockchain reached a specified length after which the results of the simulation were determined.

To judge whether an attack was successful, we had to use the measure of the highest aggregate local mean. This differs from PoW, which uses the longest chain rule to determine the valid chain. Using this measure meant that if the dishonest sum of local means was larger than the honest sum of local means, then the attack had succeeded. Otherwise it had failed.

Binary Search

Before simulating the blockchain we first had to decide what fraction of the network would be dishonest. At first this was done by starting off with 1 dishonest node and $n - 1$ honest nodes. If an attack was

unsuccessful with this amount, we would increase the amount of dishonest nodes to 2 and decrease the amount of honest nodes to $n - 2$. We did this until an attack succeeded. We then knew the minimum amount of dishonest nodes in a network of size n, which had to be controlled by an adversary for an attack to succeed.

The proceedings of this algorithm can be seen in fig. 3.1 below and were similar to that of a linear search algorithm. This led to us having a runtime of $\Theta(n)$.

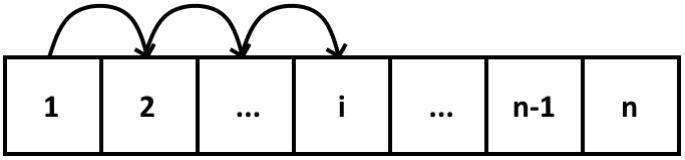


Figure 3.1. Linear Search Illustration

Compared to linear search, there are many more efficient alternatives. The algorithm which we ended up settling for was similar to a binary search algorithm. In the first round we set the boundaries to the values $(1, n)$. We then tried an attack in the middle, so with $\lfloor \frac{n+1}{2} \rfloor$ many dishonest nodes. If the attack succeeded, we set the boundaries to $(1, \lfloor \frac{n+1}{2} \rfloor)$ and if the attack had failed, we set them to $(\lfloor \frac{n+1}{2} \rfloor + 1, n)$.

This process was repeated until the left side of the boundary equalled the right side and then the answer was found. An illustration of this can be found below in fig. 3.2. The usage of this method improved our runtime from $\Theta(n)$ to $\Theta(log(n))$. The only downside was that the binary search method always took $\lceil log(n) \rceil$ many steps whereas linear search could also find an answer in two steps, if the starting point was chosen correctly.

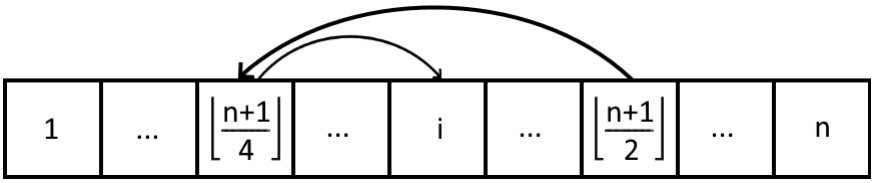


Figure 3.2. Binary Search Illustration

Print to JSON

This method converted the last simulated blockchain into a readable Json format. This was done mostly for debugging purposes, and as to see if the calculations were running correctly.

3.1.2 Nodes Class

The nodes class was responsible for managing many things related to nodes. This included creation and management of wait times and also winner determination for each leadership election.

Wait Time Generation

For the honest nodes, the wait times were generated inside the nodes class with a function from the NumPy random package, namely `np.random.exponential(loc_mean,n)`. At first, each node would individually create wait times however later this was changed to creating wait times in batches. Choosing to do so caused a significant performance benefit since the function `np.random.exponential` now only had to be called once per leadership election instead of $\theta(n)$ times.

Using NumPy for random number generation instead of wait_certificate tags, as specified in the PoET specification [11] was decided due to performance reasons. Both wait time creation methods led to the same result of having exponentially distributed random wait times. The only observable difference being

that the NumPy's method required less computational effort and allowed the generation of wait times in batches.

Dishonest nodes created wait times depending on the type of attack that was being simulated. The dishonest wait time creation for specific attacks will be described below.

Election Winner Determination

The nodes class was also responsible for determining the winner of each leadership election. Firstly the honest winner was chosen with the function `np.argmin`. The shortest honest wait time was then used to determine whether dishonest nodes could beat this time or not. Depending on the answer the winner of the election was chosen. This process can be seen below in fig. 3.3.

3.1.3 Blockchain Class

The blockchain class was responsible for saving the blockchain state, adding new blocks to the chain and many of the calculations surrounding PoET. This included the z-test, the population estimate, and the local mean computation. These calculations would normally be performed by each node separately however, since each node would get the same results, we instead performed these calculations once and saved the results onto the blockchain. This avoided redundant computation and allowed for further performance benefits.

Adding Blocks

Adding a block to the blockchain created a population sample and some block information. The population sample was composed of the duration with which a node had won, the local mean that was used to create the duration and also a population estimate that was calculated for each population sample. The block information was composed of the id of the winning node, a boolean which contained whether a node was honest or not and a boolean on whether the node had passed the z-test or not.

Caching

Caching was used in many parts of our simulation, to speed up calculations and to avoid redundant operations. For computing the population size, as seen in alg. 1, the values `sm` and `sw` were cached and continuously updated every time a new block was added to the blockchain. This way we only had to compute

```
populationSize = sm/sw
```

to receive the population size instead of having to traverse the entire blockchain history.

To make the z-test more efficient, we only ever calculated the z-test for the most recent block and added the results to the block information under an attribute called `valid_values`. Moreover, we cached the values `observed`, `expected` and `blockCount` as to make the calculation of the z-test more efficient. Similarly to `sm` and `sw` these values were updated every time a new block was added to the blockchain. The final implementation of the z-test can be seen below in alg. 4.

Algorithm 4 z_test_present(id)

```
        expected = population_samples.expected
        if observed[id] > minObserved and observed[id] > expected:
            p = expected / blockCount
            σ = sqrt(blockCount * p * (1.0 - p))
            z = (observed[id] - expected) / σ
            if z > zmax:
                return False
    return True
```

Z-Test Future

Once the TEE can not be relied upon, the z-test becomes the only defence against adversaries. However, a smart adversary would never get captured by the z-test, since before pushing a block they could check whether doing so would get them caught or not. Knowing the outcome of the z-test allows adversaries to push as many blocks as possible without ever getting caught.

For this reason, we implemented a function called `z_test_future`. This function took a `duration` as the input and would give an array with a boolean for each dishonest node as the output. If the value for a dishonest node was true, then the dishonest node would pass the z-test and could push a block. If the value was false, then the dishonest node would fail the z-test and instead of getting caught, they would refrain from pushing a block.

Algorithm 5 shows us an implementation of the z_test_future function for a single node. We adapted the code given in alg. 4 by pretending that the node had won the election and running the z-test.

Algorithm 5 z_test_future(duration, id)

```
blockCount += 1
observed += 1
sw = population_samples.sw + duration - minimum_wait_time
expected += sw / population_samples.sm
    if observed[id] > minObserved and observed[id] > expected:
        p = expected / blockCount
        σ = sqrt(blockCount * p * (1.0 - p))
        z = (observed[id] - expected) / σ
        if z > zmax:
            return False
return True
```

To make the leadership election process more clear fig. 3.3 represents a sequence diagram of the leadership election from inside our simulation software.

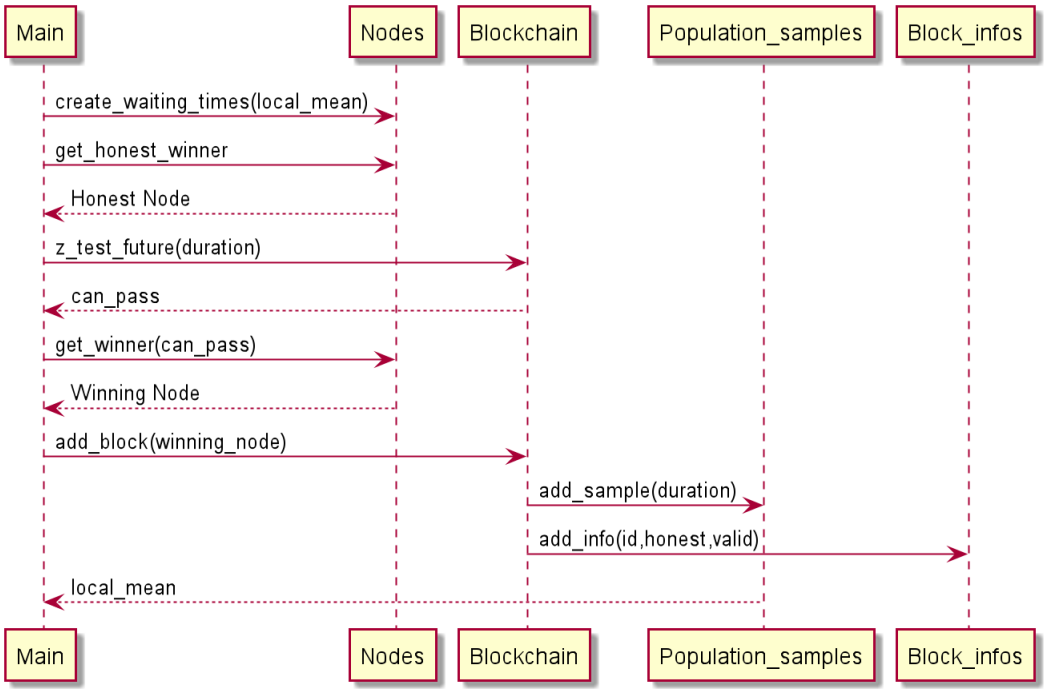


Figure 3.3. Leadership Election Sequence Diagram

3.2 Simulating Attacks

The `z_test_future` function mentioned above was used as the basis for every attack. Once an adversary was equipped with the `z_test_future` function they no longer had to fear getting caught by the statistical z-test employed in PoET. This allowed adversaries to win as many leadership elections as the z-test permitted. We then used the simulation framework to test different types of attacks.

3.2.1 Black-Box Attacks

The first type of attack we simulated was referred to as a black-box attack. In the black-box attack setting, the adversary did not possess any additional knowledge pertaining to the honest nodes that was not available to each participant. The adversary did however, have the ability to choose the wait times of each dishonest node.

In this type of setting, the best plan of attack was to present a dishonest node, which could pass the z-test, with a wait time smaller than the target wait time. Having a wait time smaller than the target wait time meant that the dishonest node was the most likely to win the leadership election. Furthermore, winning the leadership election with a wait time smaller than the target wait time caused the population estimate to rise, which lead to the local mean going up and caused the honest nodes to generate longer wait times. This method however, was a double-edged sword, since the population estimate going up also meant that the z-test would catch dishonest nodes more often. The dishonest wait time generation turned into:

$$\texttt{dishonestWaitTime} = \texttt{targetWaitTime} - \texttt{val}.$$

With `val` being a random number chosen between $[0, 1)$. `val` was needed so that constant wait times, which would get dishonest nodes caught, could be avoided.

3.2.2 White-Box Attacks

We also simulated an attack that was referred to as a white-box attack. In the white-box attack setting, the adversary possessed extensive knowledge of the network as well as each participant. To be precise, the adversary knew the exact network size instead of a population estimate and also the wait times of each node.

In this type of setting, the best plan of attack was for a dishonest node, which could pass the z-test, to create something we called a knowing time. The knowing time was a waiting time, which was slightly better than the best honest waiting time. To slightly improve the best honest wait time we multiplied it by 0.99 as to decrease it by a slight margin and added 0.01 times the `MinimumDuration` as to make sure that the wait time did not become smaller than the `MinimumDuration`.

$$\texttt{knowingTime} = 0.99 \cdot \texttt{honestDuration} + 0.01 \cdot \texttt{MinimumDuration}.$$

Other values than 0.99 and 0.01 could be used, however it is important to stay as close to the original `honestDuration` as possible. This is done as to make sure that the population estimate does not increase by a large margin.

With a `knowingTime` a dishonest node would certainly win the election. In the process, the dishonest node would also steal the win from an honest node while barely increasing the population estimate compared to if the honest node had won under normal circumstances. The population estimate not going up allows dishonest nodes to win more leadership elections without getting caught by the z-test.

Chapter 4

Findings

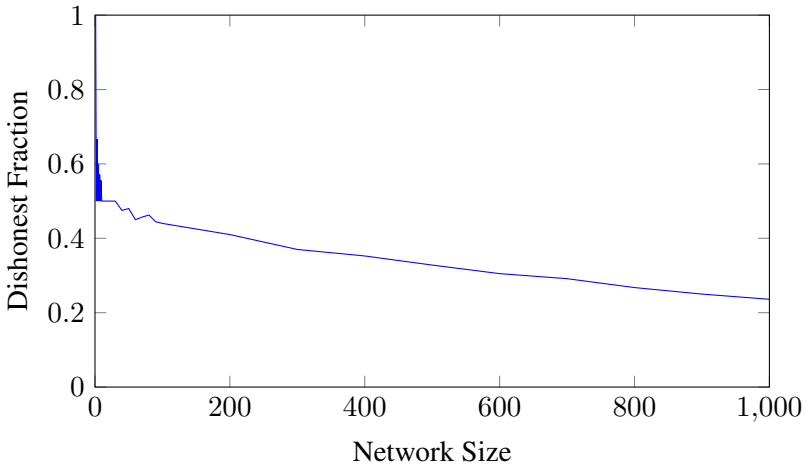
In the following section we will present the findings of our research, where we simulated a PoET network, while only taking the security provided by the z-test into account. This was done as to gain a general understanding of how secure a PoET based blockchain behaved under different circumstances. To get the most conclusive results, we conducted the simulation in regard to different chain lengths, diverse network sizes and different attack types.

The chain length dictated how long the blockchain would become before halting the simulation and evaluating the results. The chain length also represented the confirmation time, after which a block could be trusted. The network size was the sum of all honest and dishonest nodes participating in a network which in our simulations varied between one and 1000 nodes. Lastly, we distinguished between two types of attacks, namely black-box and white-box attacks, which will be explained in further detail below.

4.1 Black-Box Attacks

In the black-box attack setting, the adversary did not possess any additional knowledge pertaining to the honest nodes. The adversary did however, have the ability to choose the wait times of all the dishonest nodes. With this ability and the lack of additional knowledge, the best plan of attack was to give short wait times to dishonest nodes while ensuring that none of the dishonest nodes got caught by the statistical z-test. We named this attack the black-box attack since the network seemed enclosed in a black-box from which only finite amounts of information could be extracted.

Figure 4.1. Black-box attack with a chain length of 10'000

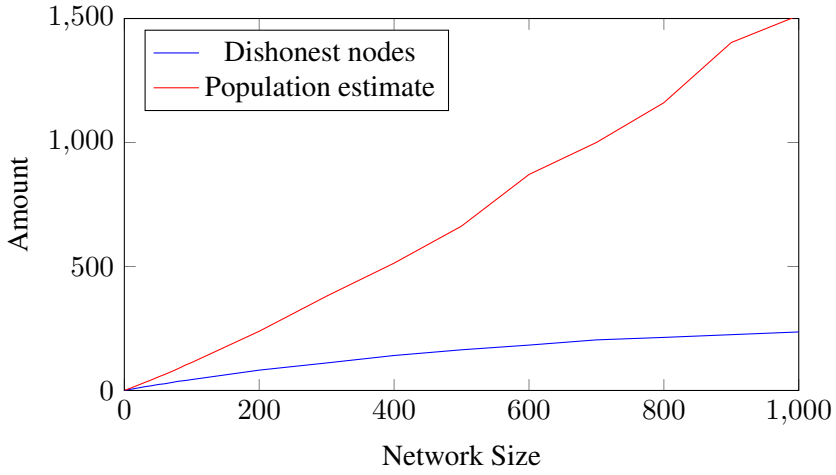


In fig. 4.1 we recorded the results of the black-box attacks with a chain length of 10'000 blocks. The exact results in table format can be found below in appendix A. To get these results we ran the simulation for a given network size and different amounts of dishonest nodes until we found the smallest amount of dishonest nodes needed for an attack to succeed. These simulations ran for 10'000 blocks and if the dishonest chain was ahead of the honest chain after 10'000 blocks the attack was a success. Different network sizes were tested by starting the simulation with a network size of one node and then increasing the network size in single steps until reaching a size of ten nodes. Next we increased the network size in steps of ten until subsequently reaching a size of 100 nodes. Finally, intervals of 100 were applied until reaching a network size of 1000 nodes.

The x-axis tells us how many nodes were actively participating in the network. In the case of our simulation, this value was the same as the network size due to each node taking part in every leadership election and none being inactive. The y-axis provides us with the smallest fraction of dishonest nodes needed for an attack to be successful at a given network size x. Values along the y-axis were calculated by taking the minimum dishonest node amount for an attack to succeed and then dividing it by the network size. In general this means that for any point below the blue line an attack was unsuccessful and for any point above or on the blue line the attack was successful. Under ideal conditions, with the TEEs working as intended, the graph would be equal to $f(x) = 0.5$ and values below $f(x)$ meant that the security was suboptimal.

We observed that the simulation started off with the entire network needing to be controlled by dishonest nodes. This was only the case when the network consisted of a singular node. Once the network reached a size of two nodes the fraction dropped to 50%, which was equivalent to the security of a PoET network, where the TEE functioned as intended. The fraction remained around 50% until we reach a network size of 40 after which the fraction dropped. With a network size of 500 nodes, we needed a fraction of 32.8%. Once the network reached a size of 1000 nodes and the required fraction was 23.6% we concluded the simulation. This meant that with a network size of 1000 nodes, an adversary needed to compromise less than $\frac{1}{4}$ of the network as to control the blockchain for around 10'000 blocks.

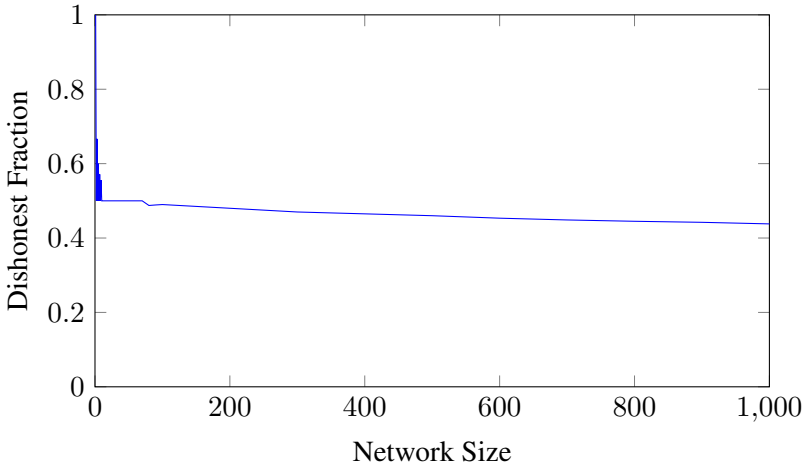
Figure 4.2. Black-box population estimate with a chain length of 10'000



In fig. 4.2 the population estimate for the successful black-box attacks is displayed via the red line, while the amount of dishonest nodes is displayed via the blue line. In the graph the x-axis signifies the amount of nodes actively participating in the network, while the y-axis signifies how large the population size respectively the dishonest node amounts were for a given x value. Under ideal conditions, with the TEEs working as intended, the red line would be equal to $f(x) = x$.

We observed that in our black-box setting the population estimate always exceeded the actual network size. The population estimate remained within an error margin of one until reaching a network size of 20 nodes, after which the population estimate began growing at a faster rate than the network size. At 1000 nodes, the population estimate was 1512.5, which was 1.5 times higher than the actual network size.

Figure 4.3. Black-box attack with a chain length of 100'000

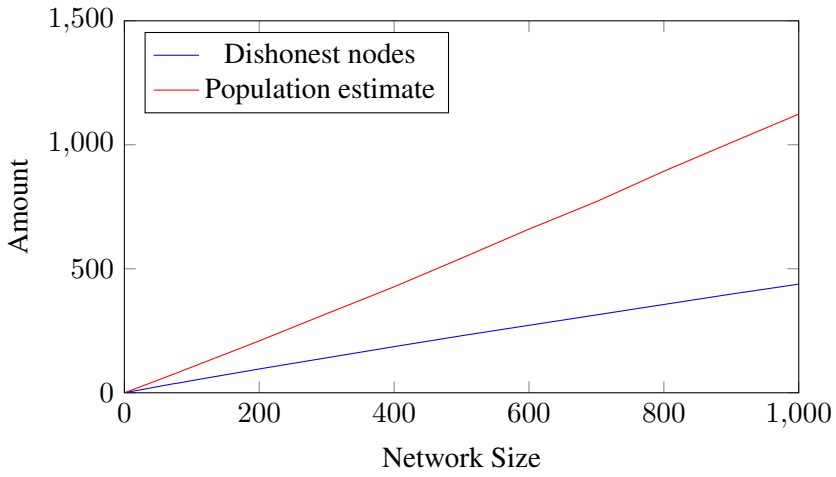


In fig. 4.3 we recorded the results of the black-box attacks with a blockchain length of 100'000 blocks and diverse network sizes. The process of recording the results was identical to fig. 4.1 with the only difference being the chain length. The x-axis represents the network size, while the y-axis represents the smallest dishonest fraction needed for an attack to succeed.

We observed that the dishonest fraction was larger or equal to fig. 4.1 for any given x value. This meant that an adversary needed to control more nodes for an attack to be successful than when the chain length was 10'000. It follows that the security of the blockchain heightened once we increased the chain length to 100'000. The fraction remained at 50% until we reached a network size of 80 nodes after which the fraction began to drop. At 500 nodes, the required fraction for an attack to succeed was 46%. At a network size of 1000 the required fraction increased from 23.6% to 43.8%.

However, it is important to note that with a target wait time of 20 seconds, it would take 23.1 days to reach a chain length of 100'000 and for 10'000 blocks this duration would be 2.3 days.

Figure 4.4. Black-box population estimate with a chain length of 100'000



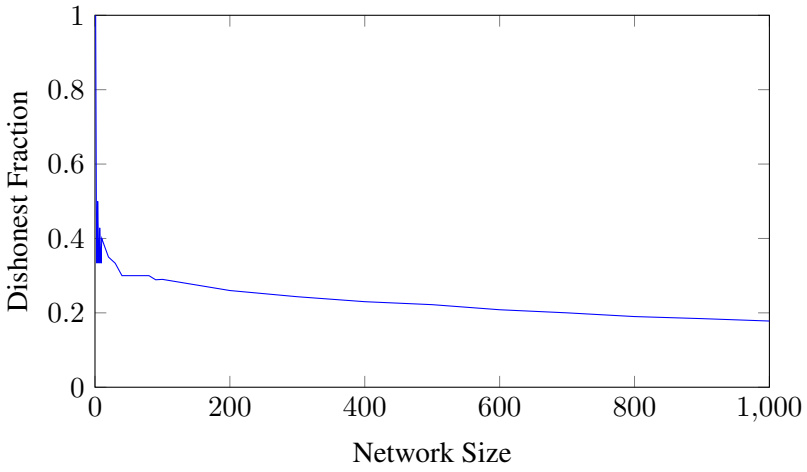
In fig. 4.4 we recorded the population estimate in red and the dishonest node amount in blue for black-box attacks with a chain length of 100'000. The x and y axis are the same as in fig. 4.2.

We observed that with a chain length of 100'000 the population estimate was closer to the actual network size compared to fig. 4.2. Although the estimate remained above the actual network size, the margin by which the population size was overestimated was considerably smaller.

4.2 White-Box Attacks

In the white-box attack setting, the adversary possessed extensive knowledge of the network, as well as each participant. The adversary knew the exact network size instead of a population estimate and also the wait times of each participant before they had elapsed. With this extensive knowledge the best plan of attack was to let dishonest nodes win as many leadership elections as possible while being only slightly faster than honest nodes. We named this attack the white-box attack or also the glass-box attack since the network seemed enclosed in a transparent box from which any information was accessible.

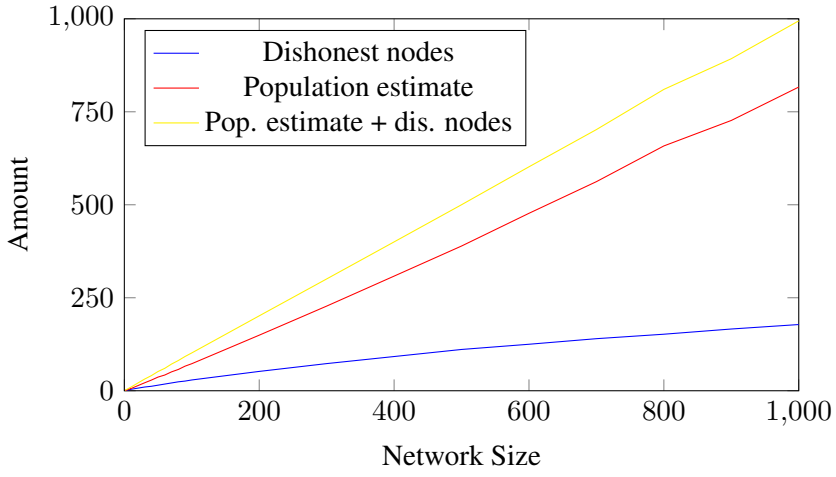
Figure 4.5. White-box attack with a chain length of 10'000



In fig. 4.5 we recorded the results of the white-box attacks with a chain length of 10'000 blocks. The process of obtaining these results was similar to that of fig. 4.1 with the only difference being the type of attack that was employed. Another similarity to the previous graph is that the x-axis represents the network size and the y-axis represents the smallest dishonest fraction needed for an attack to be successful.

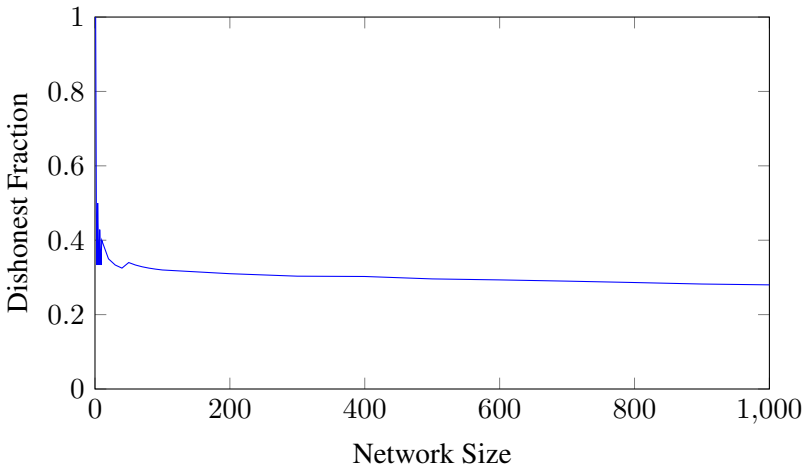
From the results, we recognised that the amount of dishonest nodes needed was always $\leq \lceil \frac{n}{3} \rceil$. With this upper bound the dishonest fraction quickly reached $\frac{1}{3}$. With a network size of three nodes, just one dishonest node was needed for an attack to be successful. This was in stark contrast to the black-box attacks with a chain length of 10'000 where the dishonest fraction reached $\frac{1}{3}$ only at around 500 nodes. For any network size the white-box attack had a smaller dishonest fraction compared to the black-box attacks, which can be seen in fig. 4.1 and fig. 4.3.

Figure 4.6. White-box population estimate with a chain length of 10'000



In fig. 4.6 we can see the population estimate in red, the amount of dishonest nodes in blue and the sum of the dishonest nodes and the population estimate in yellow. Contrary to the black-box attacks the population estimate lied below the actual network size for any given size. This meant that PoET underestimated the population size in our white-box setting. Furthermore, the yellow line was very close to the actual network size. This meant that the dishonest nodes went undetected by the population estimate.

Figure 4.7. White-box attack with a chain length of 100'000



In fig. 4.7 we recorded the results of our white-box attacks with a blockchain length of 100'000 blocks. The process of recording these results was similar to that of fig. 4.3 with the only difference being the type of attack that was employed.

From the results, we determined that the dishonest fraction was larger or equal to fig. 4.5 at any given network size. This showed us once more that increasing the chain length improved the security of a PoET system. The amount of dishonest nodes needed for a successful attack however, remained $\leq \lceil \frac{n}{3} \rceil$ for any given network size.

Figure 4.8. White-box population estimate with a chain length of 100'000

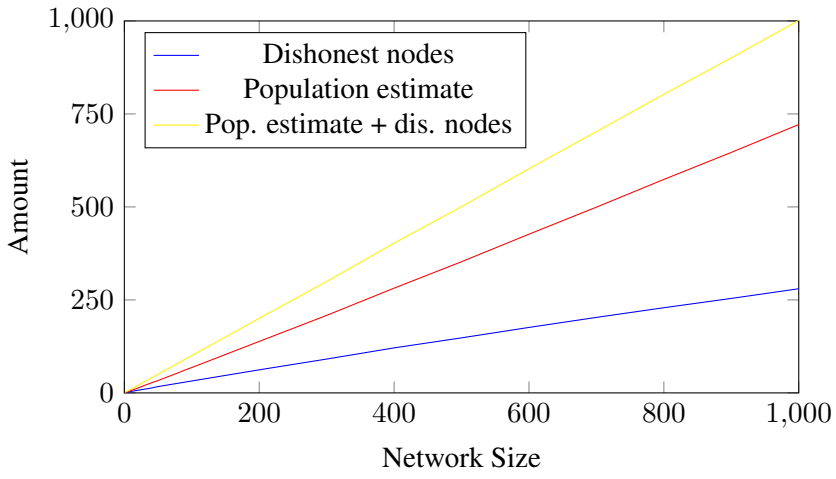
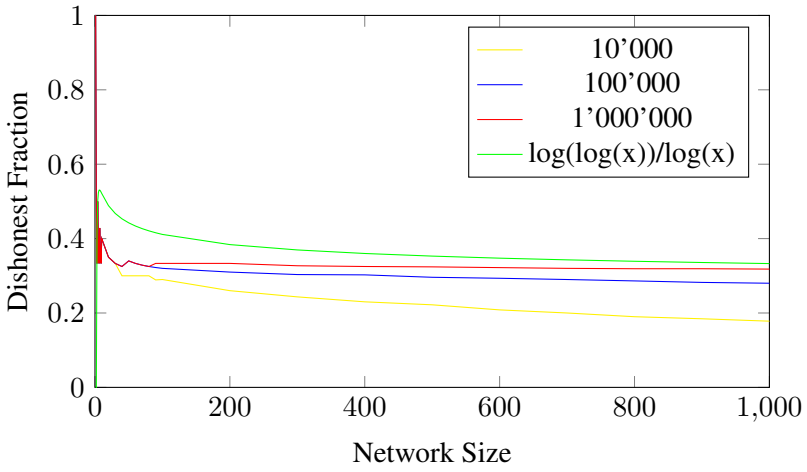


Figure 4.8 is structured in a way similar to that of fig. 4.6 with the only difference being the chain length. From the results, we observed that the population estimate was even further from the actual network size, compared to fig. 4.6. However, the yellow line was closer to the actual network size. This showed us that increasing the chain length did not always cause the population estimate to become more accurate. In a white-box setting it actually caused the population estimate to become less accurate.

Having an underestimated population estimate allowed the dishonest nodes to win more often than would otherwise be possible.

Figure 4.9. White-box Summary



In fig. 4.9 we recorded fig. 4.5 in yellow, fig. 4.7 in blue and the dishonest fraction needed for a white-box attack with a chain length of 1’000’000 in red. Furthermore, the function log(log(x))/log(x) is shown in green. We observed that the longer the chain length was, the closer the function got to log(log(x))/log(x). In our simulations log(log(x))/log(x) was an upper bound for the dishonest fraction for any given chain length. These results are more in line with the statements made by Chen et al. [16] than the ones made by Bowman et al. [17].

Chapter 5

Conclusion

5.1 Discussion

This thesis set out to perform a security analysis on the statistical z-test employed inside PoET based systems with the help of a simulation. To achieve this, we simulated diverse attacks, different network sizes and different blockchain lengths as to get the most conclusive results. Our findings show that simplistic attacks, such as presenting short waiting times to dishonest nodes, are rather ineffective. These attacks can be countered by increasing the confirmation time of a blockchain and seem to lead to requiring a dishonest fraction of 50%.

However, adversaries which implement some form of copying in their attacks prove to be quite effective. The results of our white-box attacks show that the z-test can withstand at most $\frac{1}{3}$ of the network being controlled by an adversary. Furthermore, our simulation seems to indicate that $\frac{log(log(n))}{log(n)}$ is an upper bound for the dishonest fraction as Chen et al. [16] implied.

In conclusion our simulation results point to the fact that the security provided by the TEE is a necessity rather than a cautionary measure. As long as changes do not happen to the population estimate it is unlikely that the security of the z-test alone will grow above $\frac{1}{3}$ dishonest node resistance.

5.2 Future Work

5.2.1 Simulation Software

There is always potential for expanding the simulation software. Adding new attacks, making the simulation more realistic and improving efficiency are just a few aspects that can be improved upon.

5.2.2 Dynamic Participation

Our findings show that a large problem with the PoET system lies with the population estimate, which can be heavily influenced by adversaries. Finding a way to improve the population estimate or finding a way to make PoET no longer allow dynamic participation would be a good next step to take.

Appendix A

Simulation Result Tables

In this section we show the exact data collected from the simulation in table form.

Table A.1. Black-box attack with a chain length of 10'000

Network Size	Dishonest nodes	Percentage	Population Estimate
1	1	100%	1
2	1	50%	2
3	2	66.6%	3.1
4	2	50%	4.1
5	3	60%	5.2
6	3	50%	6.2
7	4	57.1%	7.3
8	4	50%	8.2
9	5	55.6%	9.2
10	5	50%	10.3
20	10	50%	21.1
30	15	50%	32
40	19	47.5%	42.4
50	24	48%	54.2
60	27	45%	64.6
70	32	45.7%	76.1
80	37	46.3%	88.4
90	40	44.4%	101.6
100	44	44%	112.9
200	82	41%	238.8
300	111	37%	380.5
400	141	35.3%	513.5
500	164	32.8%	662.9
600	183	30.5%	870.9
700	204	29.1%	1000.1
800	214	26.8%	1160.5
900	225	25%	1403.2
1000	236	23.6%	1512.5

Table A.2. Black-box attack with a chain length of 100'000

Network Size	Dishonest nodes	Percentage	Population Estimate
1	1	100%	1
2	1	50%	2
3	2	66.6%	3
4	2	50%	4
5	3	60%	5
6	3	50%	6
7	4	57.1%	7.1
8	4	50%	8.1
9	5	55.6%	9.1
10	5	50%	10.1
20	10	50%	20.3
30	15	50%	30.5
40	20	50%	40.9
50	25	50%	51.2
60	30	50%	61.5
70	35	50%	72.1
80	39	48.8%	81.9
90	44	48.9%	93.3
100	49	49%	103.5
200	96	48%	209.2
300	141	47%	318.9
400	186	46.5%	427.3
500	230	46%	542.7
600	272	45.3%	660
700	314	44.9%	770.8
800	356	44.5%	893.4
900	398	44.2%	1009
1000	438	43.8%	1123.8

Table A.3. White-box attack with a chain length of 10'000

Network Size	Dishonest nodes	Percentage	Population Estimate
1	1	100%	1
2	1	50%	1
3	1	33.3%	2
4	2	50%	2
5	2	40%	3
6	2	33.3%	4
7	3	42.9%	4.1
8	3	37.5%	5
9	3	33.3%	6
10	4	40%	6
20	7	35%	13.7
30	10	33.3%	21.3
40	12	30%	28.6
50	15	30%	36.8
60	18	30%	42.3
70	21	30%	50.8
80	23	30%	57.2
90	26	28.9%	66.1
100	29	29%	72.7
200	52	26%	149.4
300	73	24.3%	227.6
400	92	23%	308.2
500	111	22.2%	389.1
600	125	20.8%	477.1
700	140	20%	562.2
800	152	19%	658.2
900	166	18.4%	726.7
1000	178	17.8%	816.5

Table A.4. White-box attack with a chain length of 100'000

Network Size	Dishonest nodes	Percentage	Population Estimate
1	1	100%	1
2	1	50%	1
3	1	33.3%	2
4	2	50%	2
5	2	40%	3
6	2	33.3%	4
7	3	42.9%	4
8	3	37.5%	5
9	3	33.3%	6
10	4	40%	6.1
20	7	35%	13
30	10	33.3%	20.2
40	13	32.5%	27
50	17	34%	33.2
60	20	33.3%	40.2
70	23	32.9%	47.2
80	26	32.5%	54.1
90	29	32.2%	61.5
100	32	32%	68.2
200	62	31%	138.4
300	91	30.3%	208.4
400	121	30.3%	281.6
500	148	29.6%	352.7
600	176	29.3%	426.7
700	203	29%	499.4
800	229	28.6%	573.9
900	254	28.2%	646.6
1000	280	28%	721.5

Table A.5. White-box attack with a chain length of 1'000'000

Network Size	Dishonest nodes	Percentage	Population Estimate
100	33	33%	67.3
200	66	33%	135.9
300	98	32.7%	203.2
400	130	32.5%	271
500	162	32.4%	339.6
600	193	32.2%	408.5
700	224	32%	478.2
800	255	31.9%	547.5
900	287	31.9%	617.4
1000	318	31.8%	686.2

Bibliography

[1] Cambridge Centre for Alternative Finance. (2021) Cambridge bitcoin electricity consumption index. Accessed: 2021-06-26 at 23:14. [Online]. Available: https://cbeci.org

[2] Bundesamt für Energie BFE, "Schweizerische gesamtenergiestatistik 2019," 2019.

[3] Digiconomist. (2021) Ethereum energy consumption index. Accessed: 2021-06-26 at 23:18. [Online]. Available: https://digiconomist.net/ethereum-energy-consumption/

[4] Nakamoto, Satoshi, "Bitcoin: A peer-to-peer electronic cash system," *Cryptography Mailing list at https://metzdowd.com*, 03 2009. [Online]. Available: bitcoin.org/bitcoin.pdf

[5] Back, Adam, "Hashcash - a denial of service counter-measure," 09 2002.

[6] Apolline Blandin and Dr. Gina Pieters and Yue Wu and Thomas Eisermann and Anton Dek and Sean Taylor and Damaris Njoki, "3rd global cryptoasset benchmarking study," 2020.

[7] Intel Corporation. (2016) Proof of elapsed time (poet). [Online]. Available: https://sawtooth.hyperledger.org/docs/core/nightly/0-8/introduction.html#proof-of-elapsed-time-poet

[8] Victor Costan and Srinivas Devadas, "Intel sgx explained," Cryptology ePrint Archive, Report 2016/086, 2016, https://eprint.iacr.org/2016/086.

[9] Ferdinand Brasser and Urs Müller and Alexandra Dmitrienko and Kari Kostiainen and Srdjan Capkun and Ahmad-Reza Sadeghi, "Software grand exposure: SGX cache attacks are practical," *CoRR*, vol. abs/1702.07521, 2017. [Online]. Available: http://arxiv.org/abs/1702.07521

[10] Paul Kocher and Jann Horn and Anders Fogh and Daniel Genkin and Daniel Gruss and Werner Haas and Mike Hamburg and Moritz Lipp and Stefan Mangard and Thomas Prescher and Michael Schwarz and Yuval Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy*, may 2019.

[11] Intel Corporation. (2016) Poet 1.0 specification. [Online]. Available: https://sawtooth.hyperledger.org/docs/core/releases/1.0/architecture/poet.html

[12] Alberto Solino. (2018) Algorithm aes-cmac. [Online]. Available: https://github.com/SecureAuthCorp/impacket/blob/master/impacket/crypto.py#L94

[13] (2021) Root sealing key (rsk). [Online]. Available: https://sgx101.gitbook.io/sgx101/sgx-bootstrap/attestation#root-sealing-key-rsk

[14] Stephanie Glen. (2021) Z test: Definition & two proportion z-test. [Online]. Available: https://www.statisticshowto.com/probability-and-statistics/hypothesis-testing/z-test/

[15] Wicklin, Rick, "The poisson-binomial distribution for hundreds of parameters," Oct 2020. [Online]. Available: https://blogs.sas.com/content/iml/2020/10/07/poisson-binomial-hundreds-of-parameters.html

[16] Chen, Lin and Xu, Lei and Shah, Nolan and Gao, Zhimin and Lu, Yang and Shi, Weidong, "On security analysis of proof-of-elapsed-time (poet)," in *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 2017, pp. 282–297.

[17] Bowman, Mic and Das, Debajyoti and Mandal, Avradip and Montgomery, Hart, "On elapsed time consensus protocols." *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 86, 2021.

[18] Garay, Juan and Kiayias, Aggelos and Leonardos, Nikos, "The bitcoin backbone protocol: Analysis and applications," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 281–310.

[19] The NumPy community. (2021) What is numpy? [Online]. Available: https://numpy.org/doc/stable/user/whatisnumpy.html

Erklärung

Erklärung gemäss Art. 30 RSL Phil.-nat. 18

Ich erkläre hiermit, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe r des Gesetzes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

Für die Zwecke der Begutachtung und der Überprüfung der Einhaltung der Selbständigkeitserklärung bzw. der Reglemente betreffend Plagiate erteile ich der Universität Bern das Recht, die dazu erforderlichen Personendaten zu bearbeiten und Nutzungshandlungen vorzunehmen, insbesondere die schriftliche Arbeit zu vervielfältigen und dauerhaft in einer Datenbank zu speichern sowie diese zur Überprüfung von Arbeiten Dritter zu verwenden oder hierzu zur Verfügung zu stellen.

Bern 30.7.2021
Ort/Datum


Unterschrift