Abstract—Most blockchains follow Bitcoin’s model and, as a result, their security relies on proof-of-work. In order to add blocks to the chain, users must prove that they used a certain amount of computational power. Proof-of-work is very energy-intensive, with Bitcoin’s energy consumption being on par with a country like Ireland.

This thesis aims at proposing a novel proof-of-space alternative to proof-of-work that reduces the energy requirements of blockchain protocols. In a blockchain that uses proof-of-space, miners must prove that they are dedicating non-trivial amounts of memory to the protocol. Before being able to start mining, miners must perform some computations whose results will be stored in the memory. Whenever they add a block to the blockchain, they must prove that they are correctly storing the result of those computations. The usage of proof-of-space in blockchains is a recent topic, and the majority of existing work in this area is only theoretical. This work proposes a concrete implementation of a blockchain that uses proof-of-space, built on top of the Ethereum protocol, one of the most popular blockchains.

I. INTRODUCTION

In recent years blockchains have become a popular topic. A blockchain is a data structure maintained by a distributed network of nodes that provides an immutable append-only log of records. The records are grouped into blocks, and each block has the cryptographic hash of the previous one. Thus any change to the order of blocks or the content of any block will invalidate the chain. Nodes must also run a consensus protocol to ensure that all nodes agree on the blocks that are added to the chain.

Blockchains are essential in scenarios where we want to run a decentralized application on an untrusted network with nodes that can have arbitrary behaviors. The most popular type of applications that use blockchains are cryptocurrencies. A cryptocurrency is a digital asset that provides an alternative to conventional currencies. Cryptocurrencies run in a decentralized network where there are no trusted third-parties (in opposition to conventional currencies that rely on central banks). Another appealing feature of blockchains is smart contracts. Smart contracts are pieces of code that run on top of the blockchain. They allow two or more parties to build trusted distributed applications since the underlying blockchain ensures that the code is executed and that all nodes agree on the result of executing that code. Ethereum [1] is a prominent blockchain platform that can be used to develop such smart contracts.

Most blockchains use a variant of the protocol known as Nakamoto Consensus [2], originally introduced in Bitcoin [3]. One can think of Nakamoto Consensus as the combination of 3 things: incentives, a chain selection rule, and proof-of-work. Proof-of-work is a cryptographic puzzle that nodes must solve in order to add blocks to the blockchain that binds the ability to contribute to the protocol with the expenditure of non-counterfeitable resources, namely computational power, that prevents Sybil attacks [4]. Nevertheless, this mechanism is not sufficient to select which blocks should be added to the chain. In a big network where all nodes are working to create new blocks, two nodes might create two blocks concurrently, which results in different nodes with different valid blockchains, a fork. In order to make nodes agree on the same chain, we need a chain selection rule. In Bitcoin, nodes choose the longest proof-of-work chain i.e., the chain that required more computational power to generate. The last but maybe the most important part is the incentive. Similarly to BitTorrent [5], incentives build robustness in Bitcoin by making nodes want to follow the protocol. When a node adds a block to the blockchain by solving a proof-of-work, it gets an incentive in the form of coins in the system. This incentive ensures that nodes follow the chain selection rule. If they do not follow this rule and mine on top of arbitrary chains, they will not receive the reward. Since producing proofs-of-work on those chains still requires computational power that wastes electricity, they will end up losing money.

Proof-of-work has become popular and is used in several blockchains such as Ethereum [1], Litecoin [6] and Fastcoin [7]. Unfortunately, it comes with two crucial limitations. The amount of energy it wastes is significant. O’Dwyer and Malone [8] showed that the energy used by Bitcoin is comparable to Ireland’s energy consumption. Their study was performed in 2014; since then, Bitcoin’s energy consumption increased [9]. This energy consumption comes primarily from proof-of-work.

Proof-of-work also limits the scalability of blockchains. Fast record processing is important for blockchains, especially for blockchains that provide cryptocurrencies since fast payments are an essential feature of any electronic payment system. In order to accept a transaction, most Bitcoin clients require that 6 blocks are mined on top of the block that contains it so that the probability that the block will ever leave the blockchain becomes low enough [2]. Since a block is mined roughly every 10 minutes [3], users must wait for an hour before being able to accept a transaction. At current rates, Bitcoin is able to process 7 transactions per second (tps). Paypal is able to process 115 tps per second, and Visa is able to handle 47 000 tps (although it only needs about 2 000 tps) [10], [11]. In
Bitcoin, the block size limit is 1 MB; increasing this size would increase transaction throughput. However, increasing this value without speeding up the dissemination of blocks weakens the security of the system [11].

Finding a mechanism that can replace proof-of-work in blockchains is, therefore, an important research topic. This is not an easy task because a sybil-proof mechanism requires wasting non-counterfeitable resources (like processing power in proof-of-work), and there are not many non-counterfeitable resources. An interesting under-explored alternative to proof-of-work is proof-of-space [12], [13]. In proof-of-space, the miner must dedicate memory to the protocol instead of computational power. Usually, the miner starts by doing some computations until the amount of memory he wants to allocate is full. After this initialization, he is able to start mining. Each block will have a different challenge that the miner must answer with the space he allocated. A blockchain that uses proof-of-space will be more energy-efficient than a blockchain that uses proof-of-work. Accessing memory periodically requires less energy than constantly computing proofs-of-work.

To the best of our knowledge, there are no blockchains on the wild that use proof-of-space1, and the project that is closer to achieving that goal is Chia [15]. Chia is a recent cryptocurrency proposal that combines proof-of-space with proof-of-time. It uses an elegant proof-of-space based on inverting random functions [13]. In proof-of-time, users must prove that they invested some predetermined amount of time to the protocol. In order to accomplish this, it uses Verifiable Delay Functions (VDF) [16]–[19]. A VDF is a function that takes a predetermined amount of time to compute, and running the function on a parallel computer will not speed up the computation. In Chia the user starts by computing a proof-of-space, then he computes the proof-of-time. Each proof-of-space has a quality level. This quality gives the time that computing the VDF must take. Better proofs-of-space will give lower times and vice versa. If successful in practice, this approach might change the way we think of blockchain protocols. Although it still has a component that requires some degree of computation (proof-of-time), it will not require as much energy as a pure proof-of-work blockchain. Chia uses this combination of proof-of-space with proof-of-time to prevent attacks resulting from costless mining, i.e., attacks that exploit the fact that adding blocks to the blockchain is inexpensive, such as adding multiple sequential blocks at once. Proof-of-time makes mining on more than one chain more expensive than in a proof-of-stake blockchain, preventing the nothing-at-stake problem. The objective of most miners will be having good proofs-of-space instead of having to compute bigger proofs-of-time, thus making this system more energy efficient.

The main problem in Chia is that VDFs are a recent area of research with several open questions. There are two significant VDF proposals, from Pietrzak [18], and Wesolowski [17].

Pietrzak’s construction requires that the public parameters are either created by a trusted third party or created by a multiparty-computation. Wesolowski’s construction requires that the adaptive root assumption holds [19], this is a problem that is not well studied. It is also unknown if there might appear Application-Specific integrated circuits (ASIC) that break VDFs’ security properties. Moreover, an attacker with a faster VDF implementation, even if still sequential, will be able to compromise the system.

This thesis aims at proposing a novel proof-of-space alternative to proof-of-work that reduces the energy requirements of blockchain protocols. To study this problem, we implemented an approach, named Etherspace, on top of Ethereum. Our approach uses the same proof-of-space model used in Chia [13]. However, unlike Chia, we will exclusively use proof-of-space. Etherspace tries to solve a common problem of some non-proof-of-work blockchain models, generating secure randomness. Some proof-of-stake models, Algorand [20] and Ouroboros [21], deal with this problem in different ways. Our model borrows techniques from the latter and adapts them to a new context. More precisely, we use secure coin flipping protocols to generate randomness, with some elements of blockchains that use byzantine committees to serialize transactions [22]–[24]. To the best of our knowledge, this has never been attempted with proof-of-space models. Moreover, Etherspace solves conceptual problems that come from combining the previous techniques and problems related to the data structure were the proof-of-space data is stored. The techniques and results of this thesis have been partially presented in a peer-reviewed publication [25].

II. RELATED WORK

A proof-of-work is a cryptographical puzzle whose solution is hard to compute and easy to verify. These puzzles were originally proposed as a measure to prevent spam email and denial-of-service attacks [26], [27]. However, its potential only was uncovered with the appearance of Bitcoin [3], the first decentralized cryptocurrency. It requires finding a nonce \( N \) such that \( H(N, B, h − 1) < T \) where \( H \) is a hash function (in Bitcoin it is double SHA-256), \( B \) is the current block, \( h − 1 \) is the hash of the previous block and \( T \) is the target difficulty. This scheme is probabilistic since the number of different nonces a user must try, in order to find a suitable solution, grows exponentially with the number of zero bits the target difficulty begins with.

Proof-of-work comes with some limitations, it does not scale, and it wastes vast amounts of energy. To address this, some alternatives have appeared in recent years. Proof-of-Stake, originally discussed in a Bitcoin Forum [28], is the most popular alternative to proof-of-work. It requires users to prove ownership of wealth (in the corresponding cryptocurrency) instead of computational resources. This method is difficult to generalize because there are lots of different algorithms. Usually, there is some source of randomness that is used to select a single stakeholder that will be able to add the next block to the chain. Blockchains that use this type of proofs

1 One could argue that Burstcoin [14] also uses this kind of proofs. However, since they do not provide a clear specification, we will not discuss it further.
have the advantage of consuming less energy because, unlike in proof-of-work blockchains, users do not need to waste resources to mine blocks.

Proof-of-stake blockchains have several well-known problems. Since mining is cheap, users can simulate mining different blocks. Then, they can check which block gives them better chances of being selected again and add it to the chain. This sort of behavior is known as costless simulation [29]. This problem exists in proof-of-stake systems that use the miner as a randomness source. Ouroboros [21] solves this problem by drawing randomness from a distributed coin tossing algorithm, while in Algorand [20] this problem is solved by using a verifiable random function (VRF) [30] that randomly selects users. However, both these models rely on strong synchrony assumptions and that a majority of stakeholders are online constantly.

Proof-of-Elapsed-Time (PoET) is a novel technique that uses trusted execution environments (TEE) to run a “lottery” that selects the user that will mine the next block [31]. This system uses the TEE to create a timer that once expired can be attested to check if the user really did wait the right amount of time. The user that gets the shortest timer from the TEE is selected as the leader of the round. This proof might be the closest we have to Nakamoto’s vision of “one-CPU-one-vote” [3]. PoET was originally contributed by Intel and is used in Hyperledger Sawtooth Lake [31]. The TEE is Intel’s Software Guard Extensions (SGX) [32]. SGX protects selected code and data from disclosure and modification.

This approach requires that we trust hardware from a single vendor, Intel. Moreover, Chent et al. [33] show that by corrupting a fraction of \( \Theta(\log \log n) \) nodes an attacker is able to compromise the system. Thus the possible impact of attacks on SGX [34]–[36] needs to be further studied.

**Proof-of-Space** has been suggested as an ecological alternative to proof-of-work [12], [13], [37]. Proofs-of-space are usually defined as a protocol where a prover (miner) must prove to a verifier that he is storing some data of size \( N \). Moreover, this protocol is divided into two stages, the initialization stage and the execution stage. There are two significant approaches to this type of proofs.

The first approach is based on graph pebbling lower bounds [12]. In this model, a malicious user either needs to use \( N \) space or to compute \( N \) times the hash function in order to make the verifier accept. A rational user will always choose the first alternative because it is cheaper and faster. These are the best security guarantees for this type of proofs.

However, this proof-of-space approach has some problems that prevent it from being a good direct replacement for proof-of-work. The size of the proof is large, in the order of Mbytes. The proof cannot be made completely non-interactive. Spacemint requires that another miner includes in his block a transaction with a commitment before a user is able to start mining. Thus, the system loses one of the nice properties of permissionless blockchains: the node’s ability to join the mining protocol just by listening to the network. If all miners stop accepting new commit transactions, new nodes cannot join the protocol.

The second type of proof-of-space is based on inverting random functions and has two stages [13]. During the initialization, the verifier sends to the prover the description of a function \( g_f : [N] \to [N] \), where \( g : [N] \times [N] \to [N] \) is a random function and \( f : [N] \to [N] \) is a random permutation. It follows that \( g_f(x) = g(x, x') \) and \( f(x) = \pi(f(x')) \) for any involution \( \pi \). Then the prover computes the function table of \( g_f \). During the execution phase, the verifier sends to the prover a random \( y \in [N] \). If the prover replies with a pair \((x, x')\) such that \( f(x) = f(x') \) (in the case that \( f \) is a random function) and \( g(x, x') = y \), the verifier accepts.

The proof-of-space by Abusalah et al. [13] was created to be used in Chia [15], a cryptocurrency project by Bran Cohen, creator of BitTorrent protocol. It will use proof-of-space in conjunction with proof-of-time. Proof-of-time is based on verifiable delay functions (VDF) [16]–[19]. A VDF is a function that takes a predetermined amount of time to compute. The output of the function must be quickly verifiable, and for every input to the function, there must be a unique output. Running the function on a parallel computer will not speed up its computation. In this model, “space” miners will add their block to the chain and broadcast it. The higher the quality of the proof-of-space, the lower the time to compute the proof-of-time will be. After receiving a block, the “time” miners will finalize the block with the proof-of-time.

This approach is promising. However, it has some issues. VDFs are a recent area of research with several open questions. Moreover, an attacker with a faster VDF implementation, even if still sequential, will be able to compromise the system.

### III. Etherspace

In this section, we will start by presenting a naive approach where we replace proof-of-work with proof-of-space in the Ethereum protocol. Then we discuss why this approach fails. Finally, we present an abstraction that addresses the identified problems and presents a solution.

**A. Naive Approach**

In this section, we discuss a naive approach that we will use as the base for our solution. This naive approach introduces some important aspects of our model.

From the two proof-of-space models that we discussed in Section II, we have chosen the one that is based on inverting random functions [13]. We decided to use this approach because it allows for non-interactive proofs, which is important for blockchain protocols because it enables miners to start mining without depending on other miners to act as the verifiers of the initialization stage. Moreover, this proof-of-space model has lower sized proofs.

Before the miner is able to start mining, he must allocate his target amount of memory, which corresponds to computing the function table of \( g_f \) [13]. This is done as follows - the miner starts by computing the table of \( f \). Every time

\[ A \text{ involution is a function that is its own inverse, such as a bit flip.} \]
he finds a collision in function $f$, it means that he has a suitable input pair $(x, x')$ for the function $g$, i.e., the condition $f(x) = f(x') : x \neq x'$ must hold. Then he must compute $g(x, x')$ and store it along with $x, x'$. When the table $g_f$ reaches the desired size, the miner can discard the table of $f$ and start mining.

In this approach, the last $n$ bits of the hash of the last block will be used as the proof-of-space challenge $y$, i.e., the value that the miner must invert. Since the size of the full table of $g_f$ is very large, and because no miner can generate the entire table, there is the probability that no miner has the pair $(x, x')$, such that $g_f(x) = y$, for a given challenge $y$. Instead of presenting the exact inversion of $y$, the miners will have to present the inversion of $y'$, from the locally stored pairs, such that the absolute difference between $y$ and $y'$ is minimal. Then, we can assign quality levels to each answer of the miners. The bigger the difference between $y$ and $y'$, the lower the quality of the proof-of-space. The worst possible answer for a given challenge will be given either by 0 or $2^n$. The value of the worst possible answer will be used as a reference for the quality of each proof-of-space.

Since there is the probability that different miners have answers with the same quality for the same challenge $y$, we make them invert more than one challenge. This will reduce the odds of a tie. The hash of the last block gives the first of these values; the remaining ones are given by the hash of the previous challenge.

In this approach, blocks will be slightly different from blocks in the Ethereum blockchain. Blocks in Ethereum do not have signatures. At first, this might seem like a security flaw; without signatures, integrity is not guaranteed. However, we have to take into account the fact that blocks must contain a valid proof-of-work. If we change a single bit from a valid block, the hash of the block will change, invalidating the proof-of-work. So in order to change an existing block, a miner would need to compute a new proof-of-work. We can conclude that proof-of-work gives us integrity guarantees over the block. Nevertheless, we do not have non-repudiation. It is, therefore, possible to mine blocks on behalf of other miners. Although this is possible, it is unlikely that there are miners wasting resources while someone else gets the reward. By removing proof-of-work from our model, we lose integrity. Thus, in our approach, all blocks must contain the signature of the miner that created them.

One of the most critical parts of blockchain protocols is the chain selection rule. In our approach, we will use a variation of the chain selection rule used in Nakamoto Consensus [2]. While on Nakamoto Consensus, the active chain is the chain that needed more computational power, in our model, the active chain will be the one that needed more space (i.e., the sum of the proof-of-space qualities of all blocks in the chain is higher).

At first, this approach might seem correct. However, it has two flaws. By using the hash of the last block as the challenge of the next block, we are giving the miner control over the challenge of the next block. Allowing a malicious miner to manipulate the challenge of the next block. Before adding his block to the chain, the miner can check if he has a good answer (i.e., an answer with high quality) to the proof-of-space challenge that is generated by his block. If his answer has low quality, he is able to change the block that he is adding to the chain, by changing the order of the transactions, which will change the hash of the block and generate a different proof-of-space challenge.

The second problem is that nothing ensures a time interval between blocks. If the time that a miner takes to create a block is less than the time that it takes to propagate the block to the network, it will be harder for the blockchain to converge. Besides, since proofs-of-space can be generated quickly because they essentially boil down to looks on local disk, a malicious miner can quickly create a long sequence of blocks. Even if the quality of the proof-of-space of each block is low, the total quality of the chain might be higher than that of a chain with few blocks with proof-of-space with high quality. Due to these problems, a malicious miner with limited space can rewrite the blockchain starting at any point.

After considering these problems, we conclude that any blockchain needs to satisfy the following requirements:

1) Concurrent deterrence - No miner should be able to gain an advantage by generating blocks concurrently for the same chain height.
2) Gradual generation - No miner should be able to create long sequences of continuous blocks quickly on top of the blockchain.
3) Chain integrity - No miner should be able to rewrite the blockchain by producing a block with a high-quality proof-of-space for a lower height.

The first two requirements are related to the problems of our naive approach. Concurrent deterrence is related to the first problem. In this blockchain model miners have an incentive to to produce blocks concurrently. When they create a block, they get to peak at the challenge of the next block. Creating multiple blocks allows miners to select the block that generates the challenge for which they have the best answer. In a traditional proof-of-work blockchain, miners do not win anything by producing blocks concurrently for the same height. The probability that they solve one proof-of-work puzzle does not increase. Gradual generation is related to the second problem. If we can ensure that there is a time interval between each block, we also assure that miners cannot generate a continuous chain of blocks quickly. Although the third requirement is not related to the problems of the naive approach, it is essential and transversal to blockchains in general. We want to prevent adversaries from rewiring the blockchain.

B. Etherspace

In this, section we present the general approach used in Etherspace that solves the previous problems.

To solve the problems of the naive approach we start by defining an oracle. All nodes (note that miners are the subset of nodes that add blocks to the blockchain and some nodes do
not add blocks to the blockchain) have access to this oracle, which ensures all the requirements presented in the previous section. The oracle recieves as argument a number \( h > 0 \) that represents a blockchain height, and returns the proof-of-space challenge for which the block at height \( h \) must contain an answer. Moreover, the oracle only returns the challenge for height \( h \), after an interval of \( t \) seconds has passed since it returned the challenge for height \( h-1 \). Initially, the oracle only returns the challenge for height 1, and for any other height, it returns nothing. When a node asks the challenge for height 1 for the first time, the oracle starts a timer. After \( t \) seconds, when the timer finishes, the oracle, when prompted, returns the challenge for height 2. When a node asks for the challenge for height 2 for the first time, the oracle will once again start the timer, and so forth. In the next subsections, we present two possible implementations for this oracle, but before, we discuss how the oracle satisfies the requirements.

In a system that uses the naive approach discussed in the previous section, miners could bias the challenge of the following block by generating blocks concurrently. Thus, granting malicious miners an advantage. This problem comes from using the blockchain as a source of randomness. Etherspace uses the oracle as a source of randomness. Therefore, preventing miners from biasing proof-of-space challenges and assuring concurrent deterrence.

The oracle directly ensures gradual generation. It is only possible to create blocks for the top of the blockchain when the oracle releases a challenge. If the oracle only releases the challenge on strict intervals, miners will always have to wait before creating another new block for a higher blockchain height. The speed by which miners can add continuous chains of blocks to the top of the blockchain becomes bounded by the oracle.

Proof-of-space grants the last requirement. In order for a miner to cause a rewrite of the blockchain from an older block, most of the blocks of his private chain would need to have proofs-of-space with higher quality. However, he will only be able to produce better proofs-of-space for every height if he has more space dedicated to the protocol then the others. We can use a variation of the common assumption used in blockchains; the resources of the attacker are bounded (persistent memory in this case). Further research is needed to define the bound that this approach can tolerate. The closest blockchain model, Chia [38], is reported to be secure as long as honest miners control at least approximately 61.5% of the space dedicated to the protocol.

This oracle enforces a strict notion of rounds in our protocol. In each round, every miner will try to add its block to the chain. This will lead to a large number of blocks flooding the network. All but one of the blocks will be orphaned since there can only be a winner per round. In order to reduce the number of blocks that will end up orphaned in the network, each node keeps a record with the quality of the best proof-of-space it saw in each round. If a node receives a block with a proof-of-space with lower quality than the best of that round it will discard it immediately. Conversely, if the quality of the proof-of-space is higher than the best, the node will update the register and broadcast the block.

C. Etherspace using SGX

The first option to implement our oracle is using a trusted execution environment (TEE) such as SGX [32]. We can use the TEE to generate the same timer (across all nodes). When the timer finishes, it will return the proof-of-space challenge for the current blockchain height (the SGX will ensure that this challenge is the same for all nodes in the network). Moreover, we can ensure by using the TEE that the challenge in incoming blocks is correct.

Using this approach would require miners to trust the protocol as well as hardware from an individual vendor. Moreover, if an attacker was able to compromise its TEE into returning the challenges earlier, he could use the time advantage to find a better proof-of-space. Due to these problems, we decided not to explore this alternative.

D. Etherspace using Coin Flipping

This section covers the second possible oracle implementation. This is the approach we chose to implement and further explore in this work. It consists of combining the usage of committees, similar to what is used in Byzcoin [22], with a coin flipping protocol, akin to what is used in Ouroboros [21], to generate a random challenge for the next block. As long as the space resources of a malicious miner are bounded, we know that the majority of the members of the committee will be honest. This protocol requires that miners communicate with each other; its execution will take some time that is limited by the latency of the network. This way, we ensure that there is a time interval between each block, bounded by the time that the protocol takes. The protocol that we use for coin flipping is Practical Hound, a variant of the RandHound protocol [39], further discussed in Section IV. This protocol has a useful feature, it generates a log. This log can be used to check if the challenge was created correctly.

![Practical Hound Sliding Window](image)

Fig. 1: Example of a sliding window of size \( w = 5 \) and \( k = 4 \)

Since a large number of miners add blocks asynchronously to the blockchain, the last \( k \) blocks might change. Let \( H \) be the height of the block at the top of the blockchain. If we select a big enough \( k \), we get that, with high probability, the blocks below height \( H - k \) will never leave the blockchain. Then, we can select a fixed-size continuous group of \( w \) blocks from height \( H - k \) to \( H - (k + w) \). This fixed-size group of \( w \) blocks will be the Practical Hound sliding window, described in more detail in the next section and similar to what is used in Byzcoin [22]. In Figure 1 we can see an example of a sliding window. The miners of the blocks in the sliding window will
run the Practical Hound protocol to generate the challenge of the next block. When an iteration of the Practical Hound protocol ends, the challenge of the next block is released, the sliding window moves one block forward, and a new iteration of Practical Hound starts. After each round finishes, all nodes will learn of the challenge in two different ways. Either they will receive the challenge creation log, or they receive a new block. In this approach, whenever a node sends a block, it also sends the challenge creation log. Even if a malicious miner wanted to hide the result of the challenge, he would need to reveal it in order for its block to be accepted.

In this model, the blockchain will end up having a slightly different structure. Practical Hound ensures that there will be a log of the creation of the challenge. This log proves that the challenge was created correctly. All blocks will have a reference to a proof-of-space challenge and its creation log. Each Practical Hound iteration depends on the challenge of the previous iteration. In turn, we get that the challenge creation logs will be connected and that each iteration cannot start before the previous one finished. Thus, apart from maintaining the regular blockchain, nodes also maintain a parallel chain of proof-of-space challenges, as depicted in Figure 2.

IV. PRACTICAL HOUND

In order to generate randomness securely, we decided to explore RandHound [39] because it has good scalability and generates a log that allows checking whether or not the randomness was created correctly. Nevertheless, in order to use RandHound, we had to make some modifications. RandHound uses a client/server model. The client is the participant that divides the servers into smaller sub-groups and moves the protocol forward. In order to bias the protocol, the client would need to control \( f \) other nodes, but he can prevent the protocol from finishing by aborting.

We can consider the client in RandHound the leader of the protocol; we will use this denomination from now on. We need a mechanism that enables us to detect if the leader is not making progress and replace it. To achieve this, we can use a view-change mechanism like in Practical Byzantine Fault Tolerance [40]. This mechanism allows us to replace the leader if he is not making progress. We will call Practical Hound to this version of RandHound.

Let \( n \) be the size of the committee, let \( h \) be a height of the blockchain (the blockchain height for which the challenge must be connected), let \( v \) be the current view number (which will start as zero for every new committee), let \( C \) be the most recent proof-of-space challenge, and let \( H \) be a secure hash function. The leader of the new committee will be given by \( H(h, v, C) \mod n \). If, at any step of the protocol, the leader stops making progress, the members of the committee will start sending view-change messages. Once a node receives 2\( f \) view-change messages, he will increase the view number and check who is the leader of the new view. The leader of the new view will send a new initialization message to signal the start of the Practical Hound run of the new view. Whenever a node receives a view-change message, it will add it to his copy of the log.

When the current committee finishes creating the challenge at height \( h \), the sliding window moves one step forward, changing one of the members of the committee. The new committee will start creating the challenge for height \( h + 1 \). The Practical Hound run for height \( h + 1 \) cannot start before the challenge of height \( h \) because each run uses the challenge of the previous run as entropy for selecting the leader. The rest of the Practical Hound protocol is equal to RandHound [39].

V. EVALUATION

In this chapter, we evaluate Etherspace. The main goal of the evaluation is to answer the following questions:

- How much energy does Etherspace consume? This question is essential to our evaluation since the primary goal of this work is reducing the energy consumption of blockchain protocols.
- Is Etherspace able to converge? We want our approach to offer similar levels of security to a proof-of-work blockchain, and thus ensure that the blockchain converges.
- How much throughput is Etherspace able to handle? We want our approach at least to have the same performance as Ethereum.

To evaluate our approach, we deployed a private Ethereum network with 50 miners, spread across three machines, with the following specifications:

- one machine with an Intel(R) Xeon(R) Gold 6138 CPU with 20 cores with a clock rate of 2.00 GHz and 62GB of RAM
- one machine with an Intel(R) Xeon(R) CPU E5 − 2660 v4 with 28 cores with a clock rate of 2.00 GHz and 62GB of RAM
- one machine with an Intel(R) Xeon(R) CPU E5−2648L v4 with 28 cores with a clock rate of 1.80 GHz and 31GB of RAM

For the workload, we used real Ethereum transactions (from block 0x50f8f4 to block 0x5f308) taken from Etherscan [41]. The transactions were fed to the miners by a Python script running in the corresponding machine. We ran each experiment for 1 hour and discarded the first and last ten minutes of each experiment.

A. Implementation

To test our approach, we implemented the Etherspace prototype on top of the Go Ethereum implementation version 8.
To instantiate the functions $f$ and $g$ required for the proof-of-space, we followed the proposal of Abusalah et al. [13], and used truncated AES with mode CBC (by truncating the output of AES, it becomes a one-way function). The 128-bit block that serves as input is formed by a nonce (this nonce has a fixed size of $n$ bits), preceded by $128 - n$ zeros. We use as initialization vector (IV) the hash of the nonce combined with the address of the miner. After computing AES, we will remove the most significant bits of the result, leaving only $n$ bits.

Unfortunately, the time to construct the proof-of-space using goleveldb was significant. Most of the time, miners will not have the exact inversion of the challenge. Since goleveldb does not have a successor or a predecessor method, we will have to iterate the table until we find the closest value. This results in a time complexity of $O(mn)$, where $n$ is the number of entries in the table, and $m$ is the number of values the user must invert.

To reduce the time complexity of constructing proof-of-space took with goleveldb, we decided to use a B-tree. We based our B-tree implementation on a publicly available implementation\(^3\). We had to modify the original implementation because it was an in-memory B-tree, and we needed persistent storage B-tree. By using a B-tree, we reduced the time complexity of computing proof-of-space from $O(mn)$ to $O(m \log_n^2)$.

\(^3\)\url{https://github.com/google/btree}

In Ethereum, blocks do not have a size limit. However, in practice, they are limited by the amount of gas used in transactions. The gas limit is not constant; miners can increase it or decrease it over time. We decided to use the gas limit registered in 30/09/2019, which was 9 976 323 in the Ethereum blockchain, according to Etherscan [41], as the starting gas limit.

For simplicity, transactions do not affect the state of the system in our experiments, i.e., they are not executed.

### B. Probabilistic Approximation in Etherspace

Testing the protocol with a large number of miners would require large amounts of space, which is not practical to obtain for our experiments. Therefore, we decided to replace the proof-of-space with a probabilistic approximation. In order to do this, we measured the distribution of the quality of the proofs-of-space. We instantiated the size $n$ of the values required by the proof-of-space $(x, x', y)$ to be $16^4$. Theoretically, with this parameter, the maximum number of entries of the table of $q_{11}$ is $65 \times 536$ (a single account cannot produce the entire table of $q_{11}$). In reality, the tables of all accounts will be different, and some of them will not contain inversions of some of the possible values. We started by generating the proof-of-space table of 500 addresses. These tables ended up having a maximum of nearly 48 000 entries. We also instantiated the number of inversions that each miner must present for each proof to be 100. The first of the values the miner must invert was created at random. The next ones were created by applying two times the hash function Keccak-256\(^5\) to the previous value.

We created a total of 42 693 proof-of-space challenges and measured the quality of the answers generated by each table. The quality of the proofs-of-space follows a normal distribution with a mean of 95 514.2 and a standard deviation of 470.7. Then, we replaced the proof-of-space in our Etherspace prototype with a random number generator that follows this distribution.

### C. Probabilistic Approximation in Ethereum

In order to run experiments with a much larger number of miners than our available number of physical nodes, we disabled the proof-of-work component. This enabled co-locating multiple miners on the same machine. Proof-of-work was replaced by a probabilistic mining selection process, implementing a poisson distribution. After tuning the process, we got an average block time of 14.26 seconds which is approximately one second higher than the current Ethereum average block time (i.e., approximately 13 seconds) and similar to last year’s block time [41].

### D. Practical Hound Prototype

For our tests, we did not develop a full Practical Hound implementation due to time constraints. We consider that our system is in a steady state when the Practical Hound committee does not go through reconfiguration (changing the leader). In our evaluation, we only take into account the steady state, so we did not implement the view-change mechanism. We use 30 as the $k$ parameter, the difference between the height of the head of the blockchain and the block of the last member of the committee. Note that choosing a safe $k$ parameter will require further research. For our testing purposes, we only need a high value.

The Practical Hound protocol requires the usage of collective signatures known as CoSi [42]. Since we only use honest miners, we did not implement this signature scheme. The computational cost of Cosi in a group of 512 members is approximately 0.5 seconds [42]. We used smaller groups in our tests, so we expect this time to be negligible with respect to the time to run the protocol, which requires the exchange of messages in a WAN environment. Moreover, not using Cosi in our prototype does not reduce message complexity. We can conclude that the time difference between a Practical Hound round that uses Cosi, and one round that does not, is negligible. Even though we do not use Cosi, all messages that are sent through our prototype still contain an ECDSA signature that is verified by all miners that receive it (even when using Cosi, Practical Hound still requires that all messages are signed).

We also implemented Schoenmakers’s PVSS scheme [43]. We started by trying to reuse EPFL DEDIS lab’s Cothority framework\(^6\) implementation, but unfortunately, we could not

\(^5\)We chose to use Keccak-256 because it is the hash function used by the Ethereum protocol.

\(^6\)\url{https://github.com/dedis/cothority}

\(^3\)https://github.com/dedis/cothority

\(^4\)In a real proof-of-space implementation, this value would be higher. However, reducing the size of the problem will most probably not affect the distribution of the quality significantly.
If Etherspace had the same amount of miners as the current number of nodes in the Ethereum main network is 7 369 (44), a committee of 100 nodes would only account for approximately 1.36% of the nodes. However, the available resources limit the number of nodes and, in turn, the size of the committee. We are only running 50 nodes, and we want no more than 50% of the nodes participating in the committee. In Table I, we can see the committee configurations used in our experiments.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Number of sub-groups</th>
<th>Participants per sub-group</th>
<th>Total number of participants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration 1</td>
<td>2</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>Configuration 2</td>
<td>4</td>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>Configuration 3</td>
<td>6</td>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td>Configuration 4</td>
<td>2</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>Configuration 5</td>
<td>4</td>
<td>6</td>
<td>25</td>
</tr>
<tr>
<td>Configuration 6</td>
<td>2</td>
<td>9</td>
<td>19</td>
</tr>
</tbody>
</table>

TABLE I: Configuration of the Practical Hound committee used in our experiments.

Moreover, we do not check if the log produced by the Practical Hound protocol is correct. In our testing setting, we know that this log will always be correct because all miners are honest.

Naturally, in a full Practical Hound implementation, these simplifications are not acceptable. However, for our testing purposes and because of the reasons provided above, these simplifications do not affect the main experimental conclusions.

E. Practical Hound Committee Size

The size of the Practical Hound committee size is an important parameter. Moreover, even with a fixed committee size, we can change the number of sub-groups and the cardinality of sub-groups. The sub-groups should be balanced, i.e., approximately have the same number of members. Increasing the size of the committee increases the number of malicious participants that the committee tolerates, at the expense of increasing the running time of Practical Hound, and in turn, the time between blocks. Note that increasing the time between blocks is not necessarily a bad thing as it allows bigger blocks with more transactions.

In an ideal setting, we want a large committee. If the Etherspace network had the same size as the Ethereum Main-net (7 369 nodes [44]), a committee of 100 nodes would only account for approximately 1.36% of the nodes. However, the available resources limit the number of nodes and, in turn, the size of the committee. We are only running 50 nodes, and we want no more than 50% of the nodes participating in the committee. In Table I, we can see the committee configurations used in the experiments.

F. Energy Consumption

To compare the energy consumption of Etherspace and Ethereum, we estimate how much energy is needed to add a block to both blockchains. At the time of this writing, the current number of nodes in the Ethereum main network is 7 369 [44]. If Etherspace had the same amount of miners as the Ethereum network has nodes, each miner would call the hash function 198 times to generate all the proof-of-space challenge values for every block. (if we use the same number of iterations that we used to measure the quality distribution of the proof-of-space answers, each miner will have to generate 99 values. To generate each value, it will need to call the hash function 2 times. All miners, in total, will call the hash function 1 459 062 times for each block. At the time of the writing of this work, the proof-of-work difficulty in the Ethereum main network is 2 466 TH [41], meaning that a solution to the proof-of-work is found, on average, once every 2 466+1012 calls to the hash function. We can use this number as an estimate of the number of calls to the hash function that are required for adding a block to Ethereum blockchain.

We can see that the number of calls to the hash function that is needed per block for Ethereum is several orders of magnitude higher than for Etherspace.

To have a rough idea of how those hash functions calls translate to energy, we will use as a reference an Nvidia Geforce GTX 1070 (not overclocked) that has a hash rate of 27 MH/s and a consumption of 135 W [45]. With this GPU, adding a block to the Ethereum blockchain would cost 12 329 999 999.99 J, while adding a block to the Etherspace blockchain would cost 7.30 J. As expected, the energy required to add a block to the Ethereum blockchain is several orders of magnitude higher than in Etherspace. We can conclude that Etherspace achieves its goal of being substantially more energy efficient than Ethereum.

G. Blockchain convergence

To analyze the convergence of the blockchain, we will count the number of times that miners switched active chains, i.e., the occurrence of forks. Note that when a miner adds a block to its active chain, it is not switching chain, it is extending its active chain. If the miner switches between two chains with the same number of blocks, where the only different block is the block with the highest height, we consider it a fork (i.e., a fork with a single block). We consider that the size of a fork is equal to the number of blocks that the miner pruned to switch between chains.

Fig. 3: Size and number of occurrences of forks during the course of the experiments.
The number of forks can be seen in Figure 3. Note that the figure shows the total number of forks over 40 minutes and does not show the number of forks for each blockchain height. We can see that Etherspace has approximately ten times more forks of one block than Ethereum. This result is expected since in Etherspace, all miners create blocks for every blockchain height, while on Ethereum, only a limited (sometimes only one) miner create blocks for each height. Nevertheless, Etherspace had no forks bigger than one block, while, Ethereum had forks of two and three blocks. This is probably due to Etherspace having a better differentiation mechanism for blocks at the same height (i.e., the quality of the proof-of-space).

H. Throughput

To evaluate the throughput of Etherspace and Ethereum, we must start by answering the question: How many blocks should we wait until we consider that a block is finalized? If we want to compare both models on a similar level, it is essential to use the same number of blocks for both of them. The time between blocks is similar in both models; if we give one of the models a larger margin, then the transaction confirmation time (TCT) of that model will always be better. According to Ethereum’s creator [46] in a blockchain with an estimated block time of 17 seconds, ten confirmation blocks give a secure waiting margin, which corresponds to the number of blocks added in three minutes. For this evaluation, we decided to use a slightly larger margin, 12 blocks, which corresponds approximately to the number of blocks added in three minutes in the Ethereum experiment. Note that since the most extensive fork that we had observed was of 3 blocks, we could have used a smaller margin (i.e., four blocks).

In our test environment, we have a Python script that feeds transactions to the miners. Every time a miner receives a transaction, we store the timestamp at which the transaction entered the network (note that miners only store the timestamp when they receive the transaction from the script, if they receive it from another miner, they do not store the timestamp). Whenever a block stored in the chain of a miner gets 12 blocks on top of it, we store a timestamp and associate it with all the transactions of the block. These timestamps allow us to measure how much time it took for each transaction to be accepted in the blockchain - the transaction confirmation time (TCT). We can also consider the number of transactions that were finalized during the experiment, divide it by the time of the experiment (i.e., 40 minutes since we discard the first and last 10 minutes) and get an idea of the throughput of the system in transactions per second (TPS).

We can see in Table II both the TCT and TPS of each protocol. We can conclude that both Ethereum and Etherspace have similar throughputs. Moreover, in Figure 4, we can see a relation between the Etherspace block time and the number of transactions per second that the system can handle. This relation is probably caused by the significant ratio of committee members to miners in the network. A decrease of the ratio of committee members to miners will probably increase the throughput of Etherspace significantly.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>TCT (sec)</th>
<th>TPS (txs/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Etherspace config 1</td>
<td>3 min 07</td>
<td>5.26</td>
</tr>
<tr>
<td>Etherspace config 2</td>
<td>3 min 50</td>
<td>4.46</td>
</tr>
<tr>
<td>Etherspace config 3</td>
<td>5 min 04</td>
<td>3.60</td>
</tr>
<tr>
<td>Etherspace config 4</td>
<td>3 min 47</td>
<td>4.44</td>
</tr>
<tr>
<td>Etherspace config 5</td>
<td>5 min 13</td>
<td>3.48</td>
</tr>
<tr>
<td>Etherspace config 6</td>
<td>4 min 52</td>
<td>3.67</td>
</tr>
<tr>
<td>Ethereum</td>
<td>3 min 06</td>
<td>4.11</td>
</tr>
</tbody>
</table>

TABLE II: Transaction confirmation time and transactions per second of each protocol.

![Fig. 4: Relation between average block time and transactions per second](image)

VI. Conclusions

Proof-of-work has been successful in maintaining the security of blockchain protocols while wasting vast amounts of energy. This work explored the usage of proof-of-space in blockchains. We analyzed the problems that come from replacing proof-of-space with proof-of-work, such as secure randomness generation, and proposed a novel approach that combines multiple techniques used in different blockchain models.

With this work, we were able to implement a practical proof-of-space blockchain on top of Ethereum, named Etherspace. Etherspace is an important milestone in making proof-of-space more than a promising theoretical alternative. Etherspace consumes considerably less energy than Ethereum. Each block in Etherspace requires ten orders of magnitude less energy to be generated than an Ethereum block. Besides, Etherspace has shorter forks than Ethereum and similar levels of throughput. We can conclude that Etherspace provides a promising blockchain model that can one day replace traditional proof-of-work blockchains and reduce the environmental impact of blockchain protocols.

A blockchain that uses proofs-of-space will consume less energy. Therefore, mining will also be cheaper, which will result in more small miners and, in turn, increase the decentralization of the protocol while providing a sustainable
blockchain. A sustainable future demands green blockchains. The techniques and results of this thesis have been partially presented in a peer-reviewed publication [25].

REFERENCES


