Proofs of Space-Time and Rational Proofs of Storage

Tal Moran* and Ilan Orlov**

Abstract. We introduce a new cryptographic primitive: Proofs of Space-Time (PoSTs) and construct a practical protocol for implementing these proofs. A PoST allows a prover to convince a verifier that she spent a "spacetime" resource (storing data—space—over a period of time). Formally, we define the PoST resource as a linear tradeoff between CPU work and space-time (under reasonable cost assumptions, a rational user will prefer to use the lower-cost space-time resource over CPU work).
Compared to a proof-of-work, a PoST requires less energy use, as the "difficulty" can be increased by extending the time period over which data is stored without increasing computation costs. Our definition is very similar to "Proofs of Space" [ePrint 2013/796, 2013/805] but, unlike the previous definitions, takes into account amortization attacks and storage duration. Moreover, our protocol uses a very different (and simpler) technique, making use of the fact that we explicitly allow a space-time tradeoff.

1 Introduction

A major problem in designing secure decentralized protocols for the internet is a lack of identity verification. It is often easy for an attacker to create many "fake" identities that cannot be distinguished from the real thing. Several strategies have been suggested for defending against such attacks (often referred to as "sybil attacks"); one of the most popular is to force users of the system to spend resources in order to participate. Creating multiple identities would require an attacker to spend a correspondingly larger amount of resources, making this attack much more expensive.

Any bounded resource can be used as the "payment"; one of the more common is computing resources, since they do not require any additional infrastructure beyond that already needed to access the Internet. In order to ensure that users actually do spend the appropriate resource payment, the users must employ a "proof of work".

Proofs of work have been used for reducing spam [7], for defending against denial-of-service attacks [15] and fairly recently, as the underlying mechanism for implementing a decentralized bulletin-board—this is the technical heart of the Bitcoin protocol [11].

* IDC Herzliya
** Outbrain

While effective, proofs-of-work have a significant drawback; they require energy in direct proportion to the resource used (i.e., the amount of electricity required to run the CPU during the proof of work generally depends linearly on the amount of work being performed). This is especially problematic in the context of the Bitcoin protocol, since the security of the system relies on all honest parties *constantly* performing proofs of work. In addition to having an environmental impact, this also sets a lower bound on transaction fees (since rational parties would only participate in the protocol if their reward exceeds their energy cost). Recently, two (very similar) proposals for *Proofs of Space* (PoS) have been published [8,2], motivated in large part by the need to replace proofs-of-work as a basis for crypto-currencies. Indeed, Park et al. [12] designed an alternative crypto-currency that is based on Proofs of Space.

A PoS is a two-phase protocol[1]: it consists of an initialization phase and (sometime later) an execution phase. In an (N_0, N_1, T)-PoS the prover shows that she either (1) had access to at least N_0 storage between the initialization and execution phases and at least N_1 space during the execution phase, or (2) used more than T time during the execution phase.

At first glance, this definition might seem sufficient as a replacement for proof-of-work. However, in contrast to work, space can be reused. This creates two problems when using the PoS definition as a "resource payment" scheme:

1. **Amortization**: A prover with access to $\max(N_0, N_1)$ space can, without violating the formal PoS security guarantee, generate an arbitrary number of different (N_0, N_1, T)-PoS proofs while using the same amount of resources as an honest prover generating a single proof.
2. **Postponement**: The *cost* of storage is proportional to the product of the storage space and the time it is used (e.g., in most cloud storage services, it costs the same to store 10TB for two months or 20TB for one month). Under the PoS definition, a prover can pay an arbitrarily small amount by discarding almost all stored data after the initialization phase and rerunning the initialization in the execution phase (the prover only needs to store the communication from the verifier in the initialization phase). More generally, a *rational* prover will prefer to use computation over storage whenever the cost of storing the data between the phases is greater than the cost of rerunning the initialization; when this occurs the PoS basically devolves into a standard proof-of-work in terms of energy usage.
   Even if we ignore energy use, a postponement attack is a problem if the PoS is used in a protocol where the prover must generate many proofs, but only some will be verified: the dishonest prover will not have to expend resources on the unverified proofs in this case.

Although the protocols of [8,2] are not completely undermined by these attacks, they are more than just a definitional problem. In particular, in the suggested PoS protocols based on graph pebbling, the *work* performed by the honest prover in the initialization phase is proportional to the work required to access

[1] We use the formal definitions of [8], which are more general than those in [2]

the graph (i.e., $O(N_0)$). This strongly bounds the time that can be allowed between the initialization and execution phases—if we want rational provers to use space resources rather than CPU work. In the Spacecoin protocol, for example, the authors suggest running the proofs every minute or so [12] (though they do not explicitly consider the postponement attack, or whether a rational prover would prefer to use CPU work over storage).

With regards to the amortization question, intuitively it seems that using independent random oracles for different PoSs would imply that amortization is impossible (even in the protocols of [8,2]). However, it is not clear that a formal proof of this fact is trivial (in particular, the standard information-theoretic argument that a random oracle is "incompressible" isn't sufficient, since the prover has access to the same oracle during both phases; moreover, though counter-intuitive, it is possible to use space that is filled with incompressible data as a "catalyst" to help compute some functions with less space than otherwise required [5]).

1.1 Our Contributions

In this paper, we define a new proof-of-resource-payment scheme: a "Proof of Spacetime" (PoST), that we believe is better suited as a scalable energy-efficient replacement for proof-of-work. Our definition is similar to a Proof of Space, but addresses both the amortization and the postponement attacks.

In a PoST, we consider two different "spendable" resources: one is CPU work (i.e., as in previous proofs-of-work), and the second is "spacetime": filling a specified amount of storage for a specified period of time (during which it cannot be used for anything else); we believe spacetime is the "correct" space-based analog to work (which is a measure of CPU power over time). Like work, spacetime is directly convertible to cost.

Rather than require the prover to show exactly which resource was spent in the execution phase, we allow the prover to choose arbitrarily the division between the two, as long as the total amount of resources spent is enough.

That is, the prover convinces a verifier that she *either* spent a certain amount of CPU work, *or* reserved a certain amount of storage space for some specified period of time or spent some linear combination of the two. However, by setting parameters correctly, we can ensure that *rational* provers will prefer to use spacetime over work; when this is the case we say that a PoST is a *Rational Proof of Storage*.

We construct a PoST based on *incompressible* proofs-of-work (IPoW); a variant of proofs-of-work for which we can lower-bound the storage required for the proof itself. We give a candidate construction based on the standard "hash preimage" PoW. In comparison with [8], we think of the time between the initialization and proof phases as *weeks* rather than minutes (this could enable, for example, a crypto-currency in which the "miners" could be completely powered off for weeks at a time). Moreover, our protocols and proofs use a very different technique than the Proofs of Space protocols, and we believe they are simpler to implement.

Our constructions (and proofs) are in the random oracle model (like most previous work on memory-hard functions and proofs of space). The soundness proof for IPoW construction requires a non-standard assumption about the adversary (we use an assumption in the spirit of that made by [8]). As do [8], we conjecture that our construction is sound even without the extra assumption.

1.2 Related Work

Memory-Bound Functions One of the inspirations for our protocol is the memory-bound function defined by Dwork, Goldberg and Naor [6]. The goal in designing a memory-bound function is to make the performance bottleneck for function evaluation the memory latency rather than CPU speed. The DGN design uses a "pointer-chasing" technique in a random table to prove that an adversary who computes the function must touch many entries. Our protocol is based on a similar idea. However, since our security goals are different (e.g., we do not need to ensure the protocol actually uses a large amount of space, but do need to enforce a tradeoff between space and work) our protocol (and proof techniques) are different.

Memory-Hard Functions Loosely speaking, a memory-hard function is a function that requires a large amount of memory to evaluate [13,1]. One of the main motivations for constructing such functions is to construct proofs-of-work that are "ASIC-resistant" (based on the assumption that the large memory requirement would make such chips prohibitively expensive). Note that the proposed memory-hard functions are still proofs-of-*work*; the prover must constantly utilize her CPU in order to produce additional proofs. PoSTs, on the other hand, allow the prover to "rest" (e.g., by turning off her computer) while still expending space-time (since expending this resource only requires that storage be filled with data for a period of time).

Proofs of Storage/Retrievability In a proof-of-storage/retrievability a prover convinces a verifier that she is correctly storing a file previously provided by the verifier [9,4,3,10,14]. The main motivation behind these protocols is verifiable cloud storage; they are not suitable for use in a PoST protocol due to high communication requirements (the verifier must send the entire file to the server in the first phase), and because they are not publicly verifiable. That is, if the prover colludes with the owner of the file, she could use a very small amount of storage space and still be able to prove that she can retrieve a large amount of pseudorandom data.

2 Proofs of Spacetime

A PoST deals in two types of resources: one is processing power and the other is storage. All our constructions are in the random oracle model—we model processing power by counting the number of queries to the random oracle.

Modeling storage is a bit trickier. Our purpose is to allow an *energy-efficient* proof-of-resource-consumption for rational parties, where we assume that the prover is rewarded for each successful proof (this is, roughly speaking, the case in Bitcoin). Thus, simply proving that you used a lot of space in a computation is insufficient; otherwise it would be rational to perform computations without pause (reusing the same space). Instead, we measure spacetime—a unit of space "reserved" for a unit of time (and unusable for anything else during that time). To model this, we separate the computation into two phases; we think of the first phase as occurring at time $t = 0$ and the second at time $t = 1$ (after a unit of time has passed). After executing the first phase, the prover outputs a state $\sigma \in \{0,1\}^*$ to be transferred to the second phase; this is the only information that can be passed between phases. The size of the state $|\sigma|$ (in bits) measures the space used by the prover over the time period between phases;

Informally, the soundness guarantee of a PoST is that the *total* number of resource units used by the adversary is lower bounded by some specified value—the adversary can decide how to divide them between processing units and spacetime units.

We give the formal definition of a PoST in subsection 2.2, in subsection 2.4 we present a simple construction of a PoST, and in subsection 2.5 we prove its security.

2.1 Units and Notation

Our basic units of measurement are CPU throughput, Space and Time. These can correspond to arbitrary real-world units (e.g., 2^{30} hash computations per minute, one Gigabyte and one minute, respectively). We define the rest of our units in terms of the basics:

- Work: CPU×time; A unit of CPU effort expended (e.g., 2^{30} hash computations).
- Spacetime: space×time; A space unit that is "reserved" for a unit of time (and unusable for anything else during that time).

In our definitions, and in particular when talking about the behavior of *rational* adversaries, we would like to measure the total cost incurred by the prover, regardless of the type of resource expended. To do this, we need to specify the conversion ratio between work and spacetime:

Real-world Cost We define γ to be the work-per-spacetime cost ratio in terms of real-world prices. That is, in the real-world one spacetime unit costs as much as γ work units (the value of γ may change over time, and depends on the relative real-world costs of storage space and processing power).

We define the corresponding cost function, the *real-world cost* of a PoST to be a normalized cost in work units: a PoST that uses $|\sigma|$ spacetime units and x work units has real-world cost $c = \gamma|\sigma| + x$.

2.2 Defining a PoST Scheme

A PoST scheme consists of two phases, each of which is an interactive protocol between a prover P and a verifier V.[2] Both parties have access to a random oracle H.

Initialization Phase Both parties receive as input an id string $id \in \{0,1\}^k$. At the conclusion of this phase, both the prover and the verifier output state strings ($\sigma_P \in \{0,1\}^*$ and $\sigma_V \in \{0,1\}^*$, respectively):

$$(\sigma_P, \sigma_V) \leftarrow \left\langle P^H(id), V^H(id) \right\rangle \ .$$

Execution Phase Both parties receive the id and their corresponding state from the initialization phase. At the end of this phase, the verifier either accepts or rejects ($out_V \in \{0,1\}$, where 1 is interpreted as "accept"). The prover has no output:

$$(\cdot, out_V) \leftarrow \left\langle P^H(id, \sigma_P), V^H(id, \sigma_V) \right\rangle \ .$$

The execution phase can be repeated multiple times without rerunning the initialization phase.

PoST Parameters A PoST has two main "tunable" parameters: a *Space-time Tradeoff Bound*, γ^* and an *Honest PoST Cost*, w.

Space-time Tradeoff Bound We define γ^* to be the work/spacetime tradeoff bound for a PoST proof. This is a lower bound on the conversion ratio between work and spacetime required to generate a proof (i.e., a (dishonest) prover cannot trade 1 spacetime unit for less than γ^* work units).

The Space-time tradeoff bound is a parameter that can be tweaked in the protocol construction. We will set $\gamma^* > \gamma$, to ensure that it is cheaper to use storage than CPU Work whenever possible.

PoST Cost Function We also define a corresponding cost function, the *PoST Cost*: $c(|\sigma|, x) = \gamma^*|\sigma| + x$ for a PoST execution that uses x work units and $|\sigma|$ spacetime units. This will be used to bound the real-world cost of a proof execution, but is defined entirely in terms of the protocol parameters (so it can be used in a self-contained definition).

Honest PoST Cost We denote w the expected PoST Cost incurred by the *honest* prover in a proof execution (i.e., it includes both the storage cost and the work expended in the execution phase). Note that this is an upper-bound on the real-world cost incurred by the honest prover (since $\gamma^* > \gamma$).

[2] Although the definition allows general interaction, in our construction the first phase is non-interactive (the prover sends a single message) and the second consists of a single round.

Note that the Honest PoST Cost measures only the cost in the *execution* phase. However, the cost of the initialization phase must be at least w, otherwise the adversary can generate a proof more cheaply than an honest prover by deleting all data after initialization, then rerunning the initialization just before the proof phase. In this case, the adversary does not store any data between phases, so does not pay *any* space-time cost. We formalize this below as a *postponement* attack.

Definition 1 (PoST). *A protocol (P, V) as defined above is a (γ^*, w)-PoST with ϵ soundness if it satisfies the properties of completeness and ϵ-soundness defined below.*

Completeness

Definition 2 (PoST Completeness). *We say that a PoST is (perfectly) complete if for every $id \in \{0,1\}^{poly(k)}$ and every oracle H,*

$$\Pr\left[out_V = 1 : (\sigma_P, \sigma_V) \leftarrow \left\langle P^H(id), V^H(id)\right\rangle, (\cdot, out_V) \leftarrow \left\langle P^H(id, \sigma_P), V^H(id, \sigma_V)\right\rangle\right] = 1\,.$$

Note that the probability is exactly 1 and hence the completeness is perfect.

Soundness Our security definition can capture imperfect security, where a cheating adversary may have an advantage over an honest party with the same resources. We do this by introducing a "soundness parameter" $\epsilon \in (0, 1]$, where $\epsilon = 1$ means perfect soundness.

We define a security game with two phases; each phase has a corresponding adversary. We denote the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ and $\mathcal{A}_2$ correspond to the first and the second phases of the game. $\mathcal{A}_1$ and $\mathcal{A}_2$ can coordinate arbitrarily before the beginning of the game, but cannot communicate during the game itself (or between phases).

Definition 3 ((γ^*, w)-PoST ϵ-Security Game). *Each phase of the security game corresponds to a PoST phase:*

1. Initialization. *$\mathcal{A}_1$ chooses a set of ids $\{id_1, \ldots, id_n\}$ where $id_i \in \{0,1\}^*$. It then interacts in parallel with n independent (honest) verifiers executing the initialization phase of the PoST protocol, where verifier i is given id_i as input. Let $\sigma_{\mathcal{A}}$ be the output of $\mathcal{A}_1$ after this interaction and $(\sigma_{V_1}, \ldots, \sigma_{V_n})$ be the outputs of the verifiers.*
2. Execution. *The adversary $\mathcal{A}_2(id_1, \ldots, id_n, \sigma_{\mathcal{A}})$ interacts with n independent verifiers executing the execution phase of the PoST protocol, where verifier i is given (id_i, σ_{V_i}) as input.*[3]

The adversary "wins" the game if $n' \in \{1, \ldots, n\}$ of the verifiers output 1 and one of the following holds

[3] Each of the verifiers runs a copy of the honest verifier code with independent random coins; $\mathcal{A}_2$, however, can correlate its sessions with the verifiers.

1. *Space-Time attack:* $\mathcal{A}_2$'s PoST cost was less than $\epsilon \cdot w \cdot n'$ (i.e., it made less than $\epsilon \cdot w \cdot n' - \gamma^* |\sigma_{\mathcal{A}}|$ oracle calls to H).
2. *Postponement attack:* $\mathcal{A}_1$ made less than $\epsilon \cdot w \cdot n'$ oracle calls to H.

Intuitively, these conditions capture two different cheating strategies:

1. The first condition corresponds to a prover who can convince a verifier to accept despite the total of work and storage in the execution phase being less than an ϵ fraction of the total cost needed by an honest party to convince n' verifiers. This means the adversary can beat the allowed space-time trade-off.
2. The second condition corresponds to a prover who can convince a verifier despite doing less than an ϵ fraction of the required work in the initialization phase. This makes a "postponement attack" possible: since the initialization phase requires very little computational effort, the prover can "throw out" the stored data from the initialization phase and rerun the phase to regenerate any needed data during the execution phase.

The use of n ids rather than just one prevents an amortization attack, wherein the adversary reuses the same space for different proofs.

Definition 4 (PoST ϵ-Soundness). *We say that a PoST is sound if no adversary can win the ϵ-security game with more than negligible probability (in the security parameter).*

Comparison with the PoS definition As we remarked in the introduction, an (N_0, N_1, T)-PoS does not give any security guarantees with respect to the PoST definition (even if we ignore amortization), since it does not address postponement attacks at all. In the other direction, even a perfectly-sound (γ^*, w)-PoST can't guarantee a $(x, x, w - \gamma^* \cdot x)$-PoS, for any $x \in (0, w)$ (in particular, our construction allows an adversary to easily trade space for work with a linear ratio). Thus, the parameters are not truly comparable.

One can think of the two definitions as being targeted at different "regimes": a PoS forces the prover to use a lot of space, but is not well suited to long periods between the initialization and execution phases, while the PoST definition does allow long periods of elapsed time (with a suitably hard initialization step), but relies on the rationality of the adversary to enforce use of storage rather than work.

2.3 Constructing a PoST: High-Level Overview

Our proof of spacetime has each prover generate the data they must store on their own. To ensure that this data is cheaper to store than to generate (and to allow public verifiability), we require the stored data to be a proof-of-work. We construct our protocol using the abstract notion of an incompressible-proof-of-work (IPoW); this is a proof-of-work (PoW) that is non-compressible in the sense that storing n different IPoWs requires n times the space compared to storing one IPoW (we define them more formally below; see section 2.4).

As long as the cost of storing an IPoW proof is less than the cost of re-computing it, the prover will prefer to store it. However, this solution is very inefficient: it requires the prover to send its entire storage to the verifier. In order to verify the proof with low communication, instead of one large proof of work, we generate a table containing T entries; each entry in the table is a proof of work that can be independently verified.

Why the Naïve Construction Fails At first glance, it would seem that there is an easy solution for verifying that the prover stored a large fraction of the table:

1. In the initialization phase: the prover commits to the table contents (using a merkle tree whose leaves are the table entries)
2. In the execution phase: the verifier sends a random set of indices to the prover, who must then respond with the corresponding table entries and commitment openings (merkle paths to the root of the tree).

Unfortunately, this doesn't work: the prover can discard the entire table and reconstruct only those entries requested by the verifier during the execution phase.

Instead of challenging the prover on random table entries, we use a trick from Dwork et al.'s memory-bound function construction [6]: the challenge provided by the verifier defines *multiple sets* of entries. The prover must search through the table to find a set of entries that satisfies a "probing criterion". Intuitively we set the difficulty of the probing criterion, to ensure the prover must read a large fraction of the table in order to find a satisfying set; thus, the prover must either have stored or recomputed a large fraction of the table.

In a little more detail: for a challenge ch, every nonce $nonce$ selected by the prover will define the set of k entries $H(nonce||ch||1), \ldots, H(nonce||ch||k)$, where the output of the random oracle is interpreted as a pointer to the table. We use the random oracle for the probing criterion too; setting $p^* = O(k/T)$ to be the probability of success, for each set of entries $(X_1, \ldots, X_k)$ the prover must compute $H(X_1, \ldots, X_k)$, interpret the output of the oracle as a real fraction in $[0, 1)$ and check if it is less than p^*.

Intuitively, the only thing the prover can do is try many different nonces until one succeeds (in expectation $1/p^* = \Omega(T/k)$ attempts), but because the entries defined by each nonce are random and independent, with high probability the attempts will cover most of the table. Essentially, by setting p^* low enough, we force the prover to read the most of the table, even though the final output consists of only k entries and their Merkle paths to the root of the tree used to commit the table (k can be polylogarithmic in the table size).

2.4 Constructing a PoST: Formal Description

Formally, we describe the protocol in the presence of several random oracles, denoted by H_i; for $i \neq j$, H_i and H_j are independent random oracles. This is just for convenience of notation, we can implement them all using a single oracle by assigning a unique prefix to the oracle queries (e.g., $\mathrm{H}_i(x) = H(i||x)$).

Incompressible Proofs of Work The standard definitions of PoWs do not rule out an adversary that can store a small amount of data and can use it to regenerate an entire table of proofs with very low computational overhead. Thus, to ensure the adversary must indeed store the entire table we need a more restrictive definition:

An *Incompressible Proof of Work* (IPoW) can be described as a protocol between a verifier V and a prover P:

1. The prover P is given a challenge ch as input, and outputs a "proof" π:
2. The verifier receives (ch, π) and outputs 1 (accept) or 0 (reject).

For simplicity, we denote IPoW (ch) the output of the honest prover on challenge ch (this is a random variable that depends on the random oracle and the prover's coins).

Complexity and Completeness Let $q_P^{\#}$ denote the number of oracle calls made by P in the protocol (this is a random variable that depends on ch and the random coins of P).

Definition 5 ((w', m)-IPoW). *A protocol is a (w', m)-IPoW if:*

1. *$E[q_P^{\#}] \leq w'$ (the prover's expected work is bounded by w') and*
2. *$|\pi| \leq m$ (the prover's storage is bounded by m).*

Definition 6 (IPoW Completeness). *An IPoW protocol is* complete *if, for every challenge ch, the probability that the verifier rejects is negligible in the security parameter (the probability is over the coins of the prover and the random oracle).*

Compressibility Game For an IPoW protocol to be considered sound, it must prevent an adversary from amortizing work and/or storage across many proofs. (Note that we do not necessarily require generating a *single* proof to be hard with overwhelming probability.) We capture the adversary's goal by defining the (ϵ, n)-*compressibility game*:

1. $\mathcal{A}_1$ has no input (however, it has access to the random oracle). At the end of this phase it outputs a string σ.
2. $\mathcal{A}_2$ is given σ as input, and outputs a set of n distinct challenges $\mathcal{C} = \{ch_i\}_{i=1}^n$ and n proofs $(\pi_1, \ldots, \pi_n)$, one for each challenge in $\mathcal{C}$.
3. The verifier V^* outputs $\bigwedge_{i=1}^n V(ch_i, \pi_i)$.

Let $q_i^{\#}$ denote the number of oracle calls made by $\mathcal{A}_i$. The adversary wins the game if $V^* = 1$ and either:

1. *Amortization Attack:* $q_1^{\#} + q_2^{\#} < \epsilon \cdot w' \cdot n$ or
2. *Compression Attack:* $|\sigma| \cdot \frac{w'}{m} + q_2^{\#} < \epsilon \cdot w' \cdot n$.

The first win condition corresponds to a CPU amortization attack, in which the adversary can answer n challenges while doing less than $\epsilon w'$ work per challenge (this captures the standard notion of "proof of work"). The second condition corresponds to a compression attack, where an adversary can pay less than $\epsilon w'$ normalized cost per challenge, compared to a naïve trade-off ratio of w'/m oracle-queries-per-storage-bit (since a naïve prover can always exchange the m bits of the proof with w' oracle queries to reconstruct it).

Definition 7 (IPoW (ϵ, n)-Soundness). *We say a (w', m)-IPoW is (ϵ, n)-sound if there exists a constant c such that for every adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that makes at most k^c queries to the oracle (in total), the probability that $\mathcal{A}$ wins the (ϵ, n)-compressibility game is negligible in k (the security parameter).*

Intuitively, ϵ says how much of an advantage an adversary has over an honest user: to "simulate" an honest user with x resources, an adversary requires at least ϵx resources. In a $(1, n)$-sound IPoW, the adversary has no advantage over the honest user.

The formal PoST protocol description appears as Protocol 1. To construct it, we require a $(\frac{2}{3}, \frac{1}{2}T \cdot n')$-sound (w', m) IPoW. Our soundness proof requires that the amount of work per IPoW verification is at most $\frac{1}{10}\frac{n'}{n} \cdot w'$. We construct such a hash-based IPoW scheme in section 3.

Protocol 1 TABLE-POST (for $\gamma^* = w'/m$)

Public Parameters: k - security parameter, T - table size and IPoW (ch) is a (w', m)-IPoW with $(\frac{2}{3}, \frac{1}{2}T \cdot n')$ soundness. Denote $p^* = k/2T$

Storing Phase: (Performed by the prover P)

Inputs: $id \in \{0,1\}^*$.

1. Generate an array G of size T as follows:
   For each $0 \leq i < T$, set $G[i] \stackrel{\text{def}}{=} \text{IPoW}\,(id||i)$
2. Generate a commitment com on G using a Merkle tree (i.e., construct a Merkle tree whose leaves are labeled with the entries of G, and each internal node's label is the output of the random oracle on the concatenation of its children's labels; com is the root label).
3. Publish the string id and the commitment com.

Proof Phase: (Performed by the prover P)

Upon receiving a challenge ch from the verifier V:

for all $j \in \{1, \ldots, k\}$ **do**
  $count \leftarrow 0$
  repeat
    $count \leftarrow count + 1$ // Increment counter
    Set $nonce_j \leftarrow count$
    for all $t \in \{1, \ldots, k\}$ **do**
      Compute $i_{j,t} = \text{H}_1\,(nonce_j||ch||j||t) \mod T$
    end for
    Let $\pi_{decommit(j)}$ be the Merkle paths from the table entries $\{G[i_{j,1}], \ldots, G[i_{j,k}]\}$.
    Compute $pathprobe \leftarrow \text{H}_2\,(i_{j,1}||G[i_{j,1}]|| \cdots ||i_{j,k}||G[i_{j,k}]||\pi_{decommit(j)})$
  until $pathprobe < p^*$ // happens with prob. p^*
end for
Output to V the list $(nonce_1, \ldots, nonce_k)$ and $\{\pi_{decommit(1)}, \ldots, \pi_{decommit(k)}\}$.

Proof Phase: (Performed by the verifier V)

Generate a random challenge ch and send it to the prover. Wait to receive the list $(nonce_1, \ldots, nonce_k)$ and $\{\pi_{decommit(1)}, \ldots, \pi_{decommit(k)}\}$.

for all $j \in \{1, \ldots, k\}$ **do**
  for all $t \in \{1, \ldots, k\}$ **do**
    Compute $i_{j,t} = \text{H}_1\,(nonce_j||ch||j||t) \mod T$
    Verify using $\pi_{decommit(j)}$ that $G[i_{j,t}]$ has a valid commitment opening.
    Verify that $G[i_{j,t}]$ is a valid IPoW for the challenge $id||i_{j,t}$.
  end for
  Compute $pathprobe \leftarrow \text{H}_2\,(i_{j,1}||G[i_{j,1}]|| \cdots ||i_{j,k}||G[i_{j,k}]||\pi_{decommit(j)})$
  Verify that $pathprobe < p^*$
end for

2.5 Security Proof Sketch

Theorem 1 (Informal). *If the challenge ch is chosen with high min-entropy, Protocol 1 is a sound PoST protocol*

Proof. For simplicity of the proof, we assume that $\epsilon = 1/2$ Recall that Definition 3 offers the adversary two ways to win in the security game; we treat each case separately.

1. We analyze the first case (spacetime attack) by reducing from the incompressibility of IPoW; if there exists an adversary that can win with less work+space than expected, we can use this adversary to win the IPoW compression game. The idea is to use the adversary as a subroutine in a program that can reconstruct a large fraction of the table. To compress, we run the adversary's initialization phase. To decompress, we run the adversary's execution phase, simulating the oracle $\mathrm{H}_2(\cdot)$. Each path probe query (query to $\mathrm{H}_2(\cdot)$) should correspond to a set of k table entries and their commitment openings; we can use this query to "reconstruct" those table entries. (we can ignore path probe queries that don't have the right format—those are "useless", since they are independent of anything the verifier queries).
   Since the prover must output k good nonces and the oracle queries to $\mathrm{H}_2(\cdot)$ are independent for each nonce, with high probability it will have to make $\Omega(1/p^*)$ path probe queries. Moreover, since the table indices are chosen i.i.d. for each different nonce and challenge, if the adversary has previously asked about less than $1/2$ of the table entries, with high probability at least $1/4$ of the indices in each path probe query will be new. Thus, each path-probe query will add at least $1/4k$ new table entries. By setting $p^* = \Omega(k/T)$, we can ensure that with high probability at least half the table entries will be reconstructed.
   The total space we need to store is just the output of the adversary's initialization phase. Moreover, the number of work oracle queries we make while reconstructing the table is exactly the number of queries made by the adversary's execution phase. By our assumption about the adversary's winning strategy, the normalized total (of space+queries) is less than the total needed to honestly reconstruct $1/2$ of the entries. Thus, we "win" the IPoW compression game. (This argument is formalized in Lemma 2.)
2. Suppose there exists an adversary that wins with non-negligible probability under the second condition (postponement attack). In this case, the adversary cannot have solved more than half of the table entries' proofs of work. Therefore, when given a new random challenge it will either have to find a path that exists entirely in the half of the table it did solve, or break the Merkle commitment. Note that searching for a path of length k that falls entirely in one half of the table is equivalent to flipping k coins until they all return 1—the choice of nonce determines the entire path (given the challenge and existing table entries), and the random oracle ensures that each index in the path is chosen randomly and independently of the previous links. Thus,

the adversary would require a number of random oracle queries exponential in k to succeed (with high probability) (This argument is formalized in Lemma 1.)

2.6 Security Proof

For simplicity of the proof, we assume that $\epsilon = 1/2$. Recall that Definition 3 offers the adversary two ways to win in the security game; we treat each case separately.

First, assume that the adversary is deterministic (this is w.l.o.g. since the adversary is computationally unbounded). Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary in the PoST security game (defined in Definition 3) that wins the game with probability p_{win} (the probability is over the choice of challenge and the random oracles). Denote p_{win_1} the probability that $\mathcal{A}$ wins the game and $\mathcal{A}_1$ makes less than $\epsilon \cdot w \cdot n$ oracle calls to H (i.e., wins under the first condition).

Lemma 1. *If there exists an adversary $\mathcal{A}$ that wins with probability p_{win} in the PoST $\frac{1}{2}$-security game using a postponement attack, then there exists an adversary for the IPoW scheme that can win the $(\frac{2}{3}, \frac{1}{2}Tn')$-compressibility game with probability $p_{win} - \eta(k)$ (where $\eta(\cdot)$ is a negligible function).*

Proof. We construct an adversary that runs $\mathcal{A}$ once, and wins (except with negligible probability) whenever $\mathcal{A}$ wins. The IPoW adversary, $\mathcal{A}^{(IPoW)} = (\mathcal{A}_1^{(IPoW)}, \mathcal{A}_2^{(IPow)})$, works as follows:

1. $\mathcal{A}_1^{(IPoW)}$ runs $\mathcal{A}_1$, internally simulating the Merkle tree oracle (and recording all the queries).
2. When $\mathcal{A}_1$ sends $com_1, \ldots, com_n$, supposedly the root of the Merkle trees, $\mathcal{A}_1^{(IPoW)}$ can reconstruct the complete trees by checking the recorded queries to see the preimage of each tree node.
3. $\mathcal{A}_1^{(IPoW)}$ then runs the IPoW verifier on each of the leaves.
4. $\mathcal{A}_2^{(IPoW)}$ outputs the valid IPoW leaves (note that for the amortization attack, the communication between $\mathcal{A}_1^{(IPoW)}$ and $\mathcal{A}_2^{(IPoW)}$ is irrelevant).

Denote S the set of valid IPoW entries in the committed tables (i.e., the ones reconstructed by $\mathcal{A}_1^{(IPoW)}$). The list of table entries generated by $\mathcal{A}_2$ in a winning response to the challenge must be contained in S (except with negligible probability in the length of the Merkle oracle), otherwise either the IPoW verification would fail or the Merkle commitment verification would fail (in contradiction to the response being a "winning" response). We claim that this implies, in almost every instance in which $\mathcal{A}$ wins the PoST security game, that $|S| > \frac{9}{10}Tn'$.

To see why, assume in contradiction that $|S| \leq \frac{9}{10}Tn'$. Then for any set of n' tables, there must be at least one table that does not have $\frac{9}{10}T$ valid entries. In particular, since n' verifiers accepted, there must be at least one accepting PoST whose corresponding table had less than $\frac{9}{10}T$. Let S' be the entries belonging to that table. The probability (over the choice of the random oracle) that for

a given *nonce* all k entries $\mathrm{H}_1(nonce||ch||1), \ldots, \mathrm{H}_1(nonce||ch||k)$ are in S' is $(|S'|/T)^k \leq 2^{-k}$. Let $\mathbf{Succ}_{nonce}$ be the event that this test succeeded for *nonce*. Since entries of the random oracle are independent, the events $\mathbf{Succ}_{nonce}$ for different nonces are independent. Hence, the the probability that an adversary can succeed with less than $2^k/k$ oracle queries to H_1 is bounded by a negligible function in $n' \geq k$ (using the Chernoff bound).

Since $\mathcal{A}$ won the security game with a postponement attack, $\mathcal{A}_1$ made $q^{\#}_{\mathcal{A}_1} \leq \frac{1}{2}wn' = \frac{1}{2}w'Tn'$ queries to the work oracle H, hence $\mathcal{A}^{(IPoW)}$ made

$$q^{\#}_{\mathcal{A}^{(IPoW)}} = q^{\#}_{\mathcal{A}_1} + q^{\#}_{ver}Tn \leq \frac{1}{2}w' \cdot T \cdot n' + q^{\#}_{ver} \cdot T \cdot n$$

queries to the oracle (where $q^{\#}_{ver}$ is the number of oracle queries required to verify a single IPoW). When the amount of work per IPoW verification is $q^{\#}_{ver} \leq \frac{1}{10}\frac{n'}{n}$, this means $\mathcal{A}^{(IPoW)}$ output $\frac{9}{10}Tn'$ valid IPoWs, while using only $\frac{2}{3} \cdot (\frac{9}{10}Tn')$ work queries, contradicting the $\frac{2}{3}$ soundness of the IPoW.

Lemma 2. *If there exists an adversary $\mathcal{A}$ that wins with probability p_{win} in the PoST $\frac{1}{2}$-security game using a spacetime attack, then there exists an adversary for the IPoW scheme that can win the $(\frac{2}{3}, \frac{1}{2}Tn')$-compression game with probability $p_{win} - \eta(k)$ (where $\eta(\cdot)$ is a negligible function).*

Proof. We construct an adversary that runs $\mathcal{A}$ once, and wins (except with negligible probability) whenever $\mathcal{A}$ wins. The IPoW adversary, $\mathcal{A}^{(IPoW)} = (\mathcal{A}_1^{(IPoW)}, \mathcal{A}_2^{(IPow)})$, works as follows:

1. $\mathcal{A}_1^{(IPoW)}$ runs $\mathcal{A}_1$. For every Merkle oracle query corresponding to an invalid IPoW, $\mathcal{A}_1^{(IPoW)}$ intercepts the oracle call and replaces the answer with a random string (this will cause it to be invalid with high probability). If asked again, it answers consistently. ($\mathcal{A}_1^{(IPoW)}$ can ask the IPoW verification oracle, since in the space-time attack we don't care about the number of queries made by $\mathcal{A}_1^{(IPoW)}$).
2. $\mathcal{A}_2^{(IPoW)}$ runs $\mathcal{A}_2$, internally simulating H_2 (and recording all the queries to the oracles).
3. For each path-probe query (to H_2) made by $\mathcal{A}_2$, $\mathcal{A}_2^{(IPoW)}$ parses the query as a set of k table entries and their commitment openings, and verifies the commitment openings. Note that $\mathcal{A}_1^{(IPoW)}$ ensured that any validly opened entries are also valid IPoWs.
4. $\mathcal{A}_2^{(IPoW)}$ outputs the set of valid IPoWs collected during the execution of $\mathcal{A}_2$.

Since $\mathcal{A}$ wins the PoST security game with a space-time attack, $q^{\#}_{\mathcal{A}_2} + \gamma^*|\sigma|n' < \frac{1}{2} \cdot w \cdot n' = \frac{1}{2} \cdot w' \cdot T \cdot n'$, while managing to convince n' verifiers to accept. $\mathcal{A}_2^{(IPoW)}$ makes the same number of queries and uses the same amount of space, hence we must show that it can output $T \cdot n'$ valid IPoWs in order to win the compression challenge.

To do this, we require that for every PoST table for which $\mathcal{A}_2^{(IPoW)}$ reconstructed less half of the entries, $\mathcal{A}_2$ will fail to convince a verifier with high probability. Thus, there must be at least n' tables with $\frac{1}{2}T$ "good" entries.

Let S be the entries of one of the tables output by $\mathcal{A}_2$, such that $\mathcal{A}_2^{(IPoW)}$ reconstructed less than $\frac{1}{2}|S|$ of the entries. Since $\mathcal{A}_2^{(IPoW)}$ will reconstruct any entry that was a valid IPoW and was included in a path-query, this means $\mathcal{A}_2$ did not make path-queries about at least half of the entries in the table. However, in order to convince the verifier, $\mathcal{A}_2$ must output k "good" nonces—that is, paths for which $\mathrm{H}_2(\cdot)$ returned value less than $p^* = k^2/(8 \cdot T)$. Consider the array $nonce_1 \ldots, nonce_k$ of nonces output for this table by $\mathcal{A}_2$. For each nonce, the probability of finding a good path with less than $\frac{1}{2p^*}$ queries to $\mathrm{H}_2(\cdot)$ is at most $\frac{1}{2}$. Since $\mathrm{H}_2(\cdot)$ queries are independent for different nonces, if there are more than $\frac{1}{2}k$ of the nonces for which less than $\frac{1}{2p^*}$ queries were made, the probability that all of the nonces are good is at most $2^{-k/2}$. (Note that $\mathrm{H}_2(\cdot)$ is entirely independent of the view of $\mathcal{A}_1$ (and hence of σ, since we used an internally simulated oracle in $\mathcal{A}_2$. This is ok because the challenge was chosen with high min-entropy, so the probability that $\mathcal{A}_1$ could have queried $\mathrm{H}_2(\cdot)$ on the same challenge prefix is negligible.)

Hence, we can assume that at least $\frac{1}{2p^*} \cdot \frac{k}{2} = \frac{k}{4p^*}$ path-probes were made. Moreover, since the table indices are chosen i.i.d. for each different nonce and challenge, if the adversary has previously asked about less than $1/2$ of the table entries, with high probability at least $1/4$ of the indices in each path probe query will be new. Thus, each path-probe query will add at least $k/4$ new table entries to the set collected by $\mathcal{A}_2^{(IPoW)}$. Hence, except with negligible probability in k, the set S will contain at least $\frac{k}{4} \cdot \frac{k}{4p^*} = \frac{1}{2}T$ valid entries.

3 Hash-Preimage IPoW

One of the most popular proofs of work is the hash-preimage PoW: given a challenge $ch \in \{0,1\}^k$, interpret the random oracle's output as a binary fraction in $[0,1]$ and find $x \in \{0,1\}^k$ s.t.

$$H(ch||x) < p \tag{1}$$

p is a parameter that sets the difficulty of the proof. For any adversary, the expected number of oracle calls to generate a proof-of-work of this form is at least $1/p$.

At first glance, this might seem to be an incompressible PoW already—after all, the random oracle entries are uniformly distributed and independent, so compressing the output of a random oracle is information-theoretically impossible. Unfortunately, this intuition is misleading. The reason is that we need the proof to be incompressible *even with access to the random oracle*. However, given access to the oracle, it's enough to compress the *input* to the oracle. Indeed, the hash-preimage PoW is vulnerable to a very simple compression attack: Increment a counter x until the first valid solution is found, but don't store the zero

prefix of the counter. Since the expected number of oracle calls until finding a valid x is only $1/p$, on average that means only $\log \frac{1}{p}$ bits need to be stored (rather than the full length of an oracle entry).

We show that this is actually an optimal compression scheme. Therefore, to make this an *incompressible* PoW, we instruct the honest user to use this strategy, and store exactly the $\left\lceil \log \frac{1}{p} \right\rceil$ least significant bits of the counter. We note that $\frac{1}{p}$ is the *expected* number of attempts—in the worst case the prover may require more; thus, we allow the prover to search up to $\frac{k}{p}$ entries; the verifier will check k possible prefixes for the $\log \frac{1}{p}$ bits sent by the prover (with overwhelming probability, there will be a valid solution in this range). Thus, the verifier may have to make k oracle queries in the worst case in order to check a proof (however, in expectation it will be only slightly more than one).[4]

Formally,

Definition 8 (Hash-Preimage IPoW (k, p)). *The honest prover and verifier are defined as follows:*

Prover *Given a challenge y, calls H on the inputs $\{y||x\}_{x \in \{0,1\}^{\log \frac{k}{p}}}$ in lexicographic order, returning as the proof π the least significant $\log \frac{1}{p}$ bits of the first x for which $H(y||x) < p$.*

Verifier *Given challenge y and proof π, verifies that $|\pi| \leq \log \frac{1}{p}$ and that there exists a prefix z of length $\log k$ such that $H(y||z||\pi) < p$ (where π is zero-padded to the maximum length).*

Completeness

Claim. The hash-preimage protocol is a $\left(1/p, \left\lceil \log \frac{1}{p} \right\rceil\right)$-IPoW.

Proof. With all but negligible probability (in k), the honest prover succeeds in finding a valid proof. $\left\lceil \log \frac{k}{p} \right\rceil$ bits can express any number in the range $0, \ldots, \frac{k}{p}$. Since each call to the random oracle returns independent and uniformly random coins, the number of calls to the oracle will be geometrically distributed with mean $1/p$, hence the probability that the honest prover will make more than k/p calls is $(1-p)^{k/p} \leq e^{-k}$.

Soundness To prove soundness, we make use of a non-standard assumption about the adversary (in a similar vein to [8]): we assume that the adversary cannot make use of dependencies between different IPoW challenges (even if the adversary chooses the challenges). More formally:

Assumption 2 *For any set of independent random oracles $H_1, \ldots, H_n$, each corresponding to a different IPoW execution, we can partition the adversary's storage into n disjoint blocks $B_1, \ldots, B_n$ such that for all $i \neq j$ it holds that $I(H_i; B_j) = 0$.*

[4] We note that this computation can be performed by the prover instead, but it will simplify our analysis to assume the verifier performs the checks.

While this is certainly an unsatisfactory assumption, we conjecture that this is a limitation of the proof, and that the protocol itself is indeed sound (moreover, a more general version of assumption is implicitly assumed when claiming that "when using independent random oracles it is clear that amortization holds"). For readability, we defer some of the proofs to section A.

Theorem 3. *If $p < 2^{-13}$, the hash-preimage-based IPoW is $(\frac{2}{3}, n)$-sound for every $n > k$.*

Proof. Set $\varepsilon = \frac{2}{3}$. Suppose, in contradiction, that for some constant c there exists an adversary $\mathcal{A}$ and an infinite increasing sequence $k_1, k_2, \ldots$ such that $\mathcal{A}(k_i)$ makes k_i^c queries to the oracle and wins the $(\varepsilon/p, \varepsilon \cdot \ell_{k,p}, k)$-compressibility game with probability $\rho > k_i^{-c}$. Thus, $\mathcal{A}$ must succeed in satisfying either the first or the second winning condition of the compressibility game with probability $\rho/2$.

1. $q_1^{\#} + q_2^{\#} \leq (\varepsilon/p) \cdot k$. To win the game via the first condition, $\mathcal{A}$ must produce k distinct outputs satisfying the condition $H(z) < p$. Since each query to the random oracle produces an independent uniform response, in bounding the number of queries made we need only consider the number of distinct queries made to the oracle—we can ignore the query contents. Denote X_i the indicator variable for the event "the i^{th} query returned a successful response". Then $\Pr[X_i = 1] = p$, and the X_i's are all independent. Let $q \stackrel{def}{=} \varepsilon k/p$ be a bound on the number of queries made by the adversary and $X = \sum_{i=1}^{q} X_i$ denote the total number of successful responses seen by the adversary (w.l.o.g. we can assume the adversary makes all q queries). Then $E[X] = \varepsilon k$. Using Chernoff's bound (noting that $\frac{1}{2}k = \frac{1}{2}(1+(1/\varepsilon)-1)E[X]$),

$$\Pr\left[X \geq \frac{1}{2}k\right] \leq e^{-\frac{1}{2}k\frac{(1-\varepsilon)^2}{1+\varepsilon}}$$

   That is, with overwhelming probability the adversary did not see more than $\frac{1}{2}k$ successful responses. Conditioned on $X < \frac{1}{2}k$, there are at least $\frac{1}{2}k$ outputs of $\mathcal{A}$ that either have not been queried at all (in which case the probability that the corresponding verifier accepts is exactly p), or have been queried but are not successful (in which case $\mathcal{A}$ loses the game). So the probability that the adversary wins the game is bounded by

$$\begin{aligned}\Pr[\mathcal{A} \text{ wins}] &\leq \Pr\left[X \geq \frac{1}{2}k\right] + \Pr\left[\mathcal{A} \text{ wins} | X < \frac{1}{2}k\right] \\ &\leq e^{-\frac{1}{2}k\frac{(1-\varepsilon)^2}{1+\varepsilon}} + p^{\frac{1}{2}k}\end{aligned}$$

   For any constant ε and p, this is negligible (exponentially small) in k.
2. By Theorem 4, for any single IPoW in which the adversary successfully convinces a verifier with probability at least $\delta = \frac{10}{11}$, its total cost must be at least $\frac{11}{10}\varepsilon w'$. By Assumption 2, we can partition the adversary's storage into disjoint blocks for each IPoW instance; Thus, if more than $\frac{1}{11}n$ of its IPoW solutions used less than $\frac{11}{10}\varepsilon w'$ cost, the probability that it succeeds is at most $(\frac{1}{11})^{\frac{n}{11}} = 2^{-\Omega(k)}$. Hence we can assume that it used at least $\frac{10}{11} \cdot \frac{11}{10} \cdot \varepsilon \cdot w' \cdot n = \varepsilon \cdot w' \cdot n$.

4 Discussion and Open Questions

One of the apparent drawbacks of our IPoW construction is that the tradeoff between work and space is fixed, and exponential. However, this can be overcome by defining a "composite" IPoW made up of several IPoWs in sequence (e.g., with challenges formed by appending an index to the original challenge). This allows us to create a $(t \cdot w', t \cdot m)$-IPoW from a (w', m)-IPoW, giving much more flexibility in the space/time tradeoff, at the cost of increased verification time.

Constructing additional IPoW constructions using different techniques is also an interesting open question.

References

1. J. Alwen and V. Serbinenko. High parallel complexity graphs and memory-hard functions. In R. A. Servedio and R. Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 595–603. ACM, 2015.
2. G. Ateniese, I. Bonacina, A. Faonio, and N. Galesi. Proofs of space: When space is of the essence. In *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, pages 538–557, 2014.
3. G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. Song. Provable data possession at untrusted stores. *IACR Cryptology ePrint Archive*, 2007:202, 2007.
4. K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: theory and implementation. In R. Sion and D. Song, editors, *CCSW*, pages 43–54. ACM, 2009.
5. H. Buhrman, R. Cleve, M. Koucký, B. Loff, and F. Speelman. Computing with a full memory: catalytic space. In D. B. Shmoys, editor, *STOC 2014*, pages 857–866. ACM, 2014.
6. C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In D. Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 426–444. Springer, 2003.
7. C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In E. F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer, 1992.
8. S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. Proofs of space. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 585–605. Springer, 2015.
9. P. Golle, S. Jarecki, and I. Mironov. Cryptographic primitives enforcing communication and storage complexity. In M. Blaze, editor, *Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2002.
10. A. Juels and B. S. K. Jr. Pors: proofs of retrievability for large files. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 584–597. ACM, 2007.
11. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. webpage, 2008. `https://bitcoin.org/bitcoin.pdf`.

12. S. Park, K. Pietrzak, J. Alwen, G. Fuchsbauer, and P. Gazi. Spacecoin: A cryptocurrency based on proofs of space. IACR Cryptology ePrint Archive, Report 2015/528, 2015. http://eprint.iacr.org/2015/528.
13. C. Percival. Stronger key derivation via sequential memory-hard functions. BSDCan 2009, 2009.
14. R. D. Pietro, L. V. Mancini, Y. W. Law, S. E., and P. J. M. Havinga. Lkhw: A directed diffusion-based secure multicast scheme for wireless sensor networks. In *ICPP Workshops*, pages 397–. IEEE Computer Society, 2003.
15. B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for dos resistance. In V. Atluri, B. Pfitzmann, and P. D. McDaniel, editors, *ACM Conference on Computer and Communications Security*, pages 246–256. ACM, 2004.

A Information-Theoretic Proofs for Hash-Proof IPoW

Denote $w_{max} = k/p$. Let $O = O_1, \ldots, O_{w_{max}}$ be the oracle entries for a given challenge (chosen i.i.d. in $\{0,1\}^L$).

Let O' be a random variable computed from O in the following way: If $O_i < p$ then O'_i is chosen uniformly at random from the values in $[0, p)$, otherwise it is chosen uniformly at random from the values $[p, 1)$.

$X \in [w_{max}]$ is a random variable that represents the output of the prover A prover's output will be accepted by the verifier iff it holds that $O_X < p$. We define the indicator variable $\mathbf{Succ}_{X,O}$ to capture this event. Z represents the auxiliary information stored by the adversary. Z is an arbitrary random variable of size $|Z|$ bits. Z' represents any additional randomness used by the adversary (including additional random oracles) that is independent of O.

$Q \subseteq [w_{max}]$, $|Q| = q$ will denote the set of queries made by the adversary to O. That is, Q is a random variable consisting of a subset of indices $(i_1, \ldots, i_q)$, such that the adversary queries $O_{i_1}, \ldots, O_{i_q}$.

We want to show that any adversary that can compute a successful X must make a tradeoff between the number of queries to O and the size of its storage. More precisely,

Theorem 4 (Main Theorem). *For every adversary whose auxiliary input, query set and output are given by the variables Z, Q and X, respectively (and with access to an arbitrary random oracle Z' that is independent of O), if* $\Pr[\mathbf{Succ}_{X,O} = 1] = \delta \geq \frac{10}{11}$ *and* $p < 2^{-13}$ *then*

$$E(|Q|) + |Z| \frac{1}{p \log \frac{1}{p}} \geq \frac{11}{10} \cdot \frac{2}{3} \cdot \frac{1}{p}$$

The proof of the theorem appears below; first we prove several information-theoretic lemmas that we will require.

Lemma 3. *If* $\Pr[\mathbf{Succ}_{X,O} = 1] = \delta$ *then* $I(X; O') \geq \delta \log 1/p - 1$.

Proof. Define $Y = (\mathbf{Succ}_{X,O}, Y_1, Y_2)$ to be a random variable as follows: $Y_1 = O'|_{\overline{X}}$ (that is, Y_1 contains all the entries of O' in order, except for those pointed to by X).

$$Y_2 = \begin{cases} comp(O'_X) & \text{if } \mathbf{Succ}_{X,O} = 1 \\ O'_X & \text{otherwise} \end{cases}$$

where *comp* is a "compression" function that returns only the $L - \log 1/p$ least significant bits of its input. Since $\mathbf{Succ}_{X,O} = 1$ implies that $O'_X < p$, Y_2 suffices to reconstruct O'_X.

By the chain rule for mutual information, $I(X; O') = I(X, Y; O') - I(Y; O'|X)$. Since O' is a deterministic function of X, Y, it holds that $I(X, Y; O') = H(O')$. On the other hand,

$$I(Y; O'|X) \leq H(Y|X) \leq H(Y) \leq E[|Y|] \ ,$$

where $|Y|$ is the length in bits of Y's description. We can compute this exactly:

$$\begin{aligned} E[|Y|] &= 1 + E[|Y_1|] + E[|Y_2|] \\ &= 1 + E[|Y_1|] + \delta E[|Y_2| \mid \mathbf{Succ}_{X,O} = 1] + (1-\delta) E[|Y_2| \mid \mathbf{Succ}_{X,O} = 0] \\ &= 1 + |O'| - L + \left(\delta(L - \log \frac{1}{p}) + (1-\delta) \cdot L \right) \\ &= 1 + H(O') - \delta \log \frac{1}{p} \end{aligned}$$

Putting it together:

$$I(X; O') \geq H(O') - 1 - H(O') + \delta \log \frac{1}{p} = \delta \log \frac{1}{p} - 1$$

Lemma 4. *Let Z be an arbitrary random variable of size $|Z|$ bits, and Q the set of queries made to O. Let Z' be an arbitrary random variable* independent *of O and O' (i.e., $I(Z'; O, O') = 0$). Then*

$$I(O|_Q, Z', Z; O') \leq E(|Q|)H(p) + |Z|$$

Proof. By definition, $I(O|_Q, Z', Z; O') = H(O|_Q, Z', Z) + H(O') - H(O|_Q, Z', Z, O')$. Since $H(O|_Q, Z', Z) \leq H(O|_Q, Z') + H(Z) \leq H(O|_Q, Z') + |Z|$, and $H(O|_Q, Z', Z, O') \geq H(O|_Q, Z', O')$, it follows that

$$I(O|_Q, Z', Z; O') \leq I(O|_Q, Z'; O') + |Z| \ .$$

By the chain rule for mutual information, and the fact that Z' is independent of O, O',

$$I(O|_Q, Z'; O') = I(O|_Q; O') + I(Z'; O' \mid O|_Q) = I(O_Q; O') \ .$$

Finally, since the only connection between O and O' is that each for index i either both satisfy $O_i < p \wedge O'_i < p$ or both satisfy $O_i \geq p \wedge O'_i \geq p$, the mutual information between two entries is $H(p)$ if they have the same index and 0 otherwise. Hence,

$$I(O_Q; O') = E[|Q|]H(p) \ .$$

Proof (Proof of Theorem 4). Since X is a deterministic function of $O|_Q$, Z and Z', it follows that $I(X;O) \leq I(O|_Q, Z', Z; O)$. Combining Lemma 3 and Lemma 4, we have

$$\begin{aligned} E(|Q|)H(p) + |Z| &\geq I(O|_Q, Z', Z; O') \geq I(X; O') \\ &\geq \delta \log \frac{1}{p} - 1 \\ &\geq (\delta - \frac{1}{11}) \log \frac{1}{p} \end{aligned}$$

(the last inequality uses $p < 2^{-11}$.) Dividing both sides by $H(p)$ and using the fact that for $p < 2^{-13}$ it holds that $p \log \frac{1}{p} \leq H(p) \leq \frac{9}{8} p \log \frac{1}{p}$:

$$\begin{aligned} E(|Q|) + |Z| \frac{1}{p \log \frac{1}{p}} \geq E(|Q|) + |Z| \frac{1}{H(p)} &\geq (\delta - \frac{1}{11}) \frac{\log \frac{1}{p}}{H(p)} \\ &\geq (\delta - \frac{1}{11}) \frac{\log \frac{1}{p}}{\frac{9}{8} p \log \frac{1}{p}} \\ &= \frac{8}{9} \cdot (\delta - \frac{1}{11}) \cdot \frac{1}{p} \\ &= \frac{8}{9} \cdot (\frac{10}{11} - \frac{1}{11}) \cdot \frac{1}{p} \\ &= \frac{8}{9} \cdot \frac{9}{11} \cdot \frac{1}{p} \\ &\geq \frac{11}{10} \cdot \frac{2}{3} \cdot \frac{1}{p} \end{aligned}$$