Limits on revocable proof systems, with applications to stateless blockchains

Miranda Christ[1,3] and Joseph Bonneau[2,3]

[1] Columbia University
[2] New York University
[3] a16z Crypto Research

Abstract. Motivated by the goal of building a cryptocurrency with succinct global state, we introduce the abstract notion of a revocable proof system. We prove an information-theoretic result on the relation between global state size and the required number of local proof updates as statements are revoked (e.g., coins are spent). We apply our result to conclude that there is no useful trade-off point when building a stateless cryptocurrency: the system must either have a linear-sized global state (in the number of accounts in the system) or require a near-linear rate of local proof updates. The notion of a revocable proof system is quite general and also provides new lower bounds for set commitments, vector commitments and authenticated dictionaries.

1 Introduction

Modern cryptocurrencies prevent double-spending attacks using a public, append-only log called a blockchain. Classically, a blockchain records *all* transactions, and validating a new transaction requires checking that it doesn't conflict with any prior transaction. This approach was first successfully deployed by Bitcoin [17] though it was proposed earlier [11].

A challenge of the blockchain paradigm is each validator traditionally must store the entire state of the system. In Bitcoin, this state consists of a set of unspent transaction outputs (UTXOs), which has consistently grown and now contains over 85 million elements, requiring several GB to store. The state is even larger for Ethereum [26], with over 200 million accounts and 35 GB of state.

The requirement that validators store this large (and growing) state raises concerns about centralization if the state grows so large that only well-funded organizations can afford to store it. As a result, many blockchain systems impose limits on state growth, which in turn limits transaction throughput severely. Famously, Bitcoin originally imposed a maximum size of 1MB per block, limiting throughput to about three transactions per second in Bitcoin.

The tension between throughput and state growth leads to a natural question: can we achieve high throughput with a small (perhaps even constant-sized) global state? This led to the proposal of *stateless* blockchain designs [22], although this term is a misnomer: they typically assume validators store a store a small commitment to the global state of the system (e.g., a Merkle root committing

to the set of unspent coins). Users wishing to make a transaction must publish a *witness* that their transaction is valid given the current state commitment (e.g., a Merkle proof that a coin is included in the valid set). Validators can then accept transactions without knowing the full state of the system. Since Todd's original proposal using Merkle trees, several other designs have been proposed using Merkle-tree-based accumulators [4,13], RSA accumulators [3], and vector commitments [23,21,25,9,15].

Unfortunately, all known stateless blockchains introduce a new problem: users' witnesses become invalid as other transactions update the global state, requiring users to monitor transactions on the network and periodically refresh their witnesses. This is a departure from the traditional blockchain model, in which users can stay offline for long periods of time and then successfully create and broadcast a transaction. This is not simply a matter of convenience; there are important security benefits of offline participation by end users, as private keys can be kept in an air-gapped machine such as a hardware wallet.

In this work we show that, regrettably, the trade-off between a large global state and requiring frequent witness changes is fundamental. More specifically, in Theorem 1 we show a lower bound on the global state size as a function of the number of revoked statements and the desired maximum number of witness changes. In fact, in Corollary 1, we show that there is no trade-off which does not require either an (asymptotically) linear-sized global state or an (asymptotically) near-linear number of witness updates as a constant fraction of coins are spent.

To analyze the efficiency of stateless blockchains and similar authenticated data structures, we introduce a new cryptographic notion: a *revocable proof system* (RPS, Section 2). A revocable proof system is a simple abstraction capturing a class of schemes that involve a *global state* V encapsulating a set S of *valid statements*. Correctness ensures that each valid statement $s_i \in S$ has a corresponding *proof* π_i which can be efficiently verified given s_i, the public parameters, and the global state. A subset $T \subseteq S$ of the initial set of valid statements may later be *revoked*, yielding an updated global state V'. Security requires that these revoked statements' proofs no longer verify. This functionality is quite natural and captures a wide range of useful cryptographic notions, including accumulators and vector commitments. We discuss these connections in Section 4.

Using our revocable proof system definition we prove a trade-off between the size of the global state and the frequency with which proofs must be updated (Theorem 1). We do so using a compression argument: if the global state is small and with constant probability there are few ($\leq k$ for some k) proof updates, an adversary can use the global state and a small amount of additional information to encode the revoked set. We apply Shannon's Coding Theorem to show a lower bound on the size of the global state given the number of proof updates k.

As a corollary, we observe that there is no asymptotically useful trade-off between the size of the global state and the number of proof changes: either the state size is linear, or a (nearly) linear number of proofs must change (Section 3.1). As a second corollary, we show that a useful notion of *persistence* (proofs of certain statements never change) requires linear storage for these

persistent statements. In Section 4, we show that accumulators, vector commitments, and authenticated dictionaries fit the framework of a revocable proof system and thus our lower bound applies to them, giving results of independent interest in other domains. Finally, we discuss the implications of our result on stateless blockchain proposals (Section 5).

2 Model

Notation. We use λ to denote the security parameter. We use lg to denote a logarithm with the base 2. Right $\rightarrow$ and left $\leftarrow$ arrows denote the output of a (possibly randomized) algorithm.

A *revocable proof system* (RPS) maintains a *global state* V, a *valid set* S, and a set of proofs π_i for each element $s_i \in S$ which we'll also call a *statement*. The global state commits to the valid set, such that proofs of elements s_i in S can be verified. More formally, a *revocable proof system* is a tuple of algorithms $(\mathsf{Setup}, \mathsf{ComputeState}, \mathsf{Revoke}, \mathsf{Verify})$ where:

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}$ is a randomized algorithm that takes as input 1^λ, where λ is the security parameter, and outputs public parameters pp.

$\mathsf{ComputeState}(\mathsf{pp}, S) \rightarrow V, (\pi_1, \pi_2, \ldots, \pi_n)$ is a deterministic algorithm that takes as input the public parameters pp and a valid set S of size n. It outputs the corresponding global state V and a list of proofs $(\pi_1, \pi_2, \ldots, \pi_n)$ where π_i is the proof for $s_i \in S$.

$\mathsf{Revoke}(\mathsf{pp}, S, T, V, (\pi_1, \pi_2, \ldots, \pi_n)) \rightarrow V', (\pi'_1, \pi'_2, \ldots, \pi'_n)$ is a deterministic algorithm that takes as input the public parameters pp, the initial valid set S, a revoked set $T \subseteq S$, an initial global state V, and a list of proofs for elements in S. It outputs an updated global state V' and updated proofs.[4]

$\mathsf{Verify}(\mathsf{pp}, V, s_i, \pi_i) \rightarrow \{\mathsf{true}, \mathsf{false}\}$ is a deterministic algorithm that takes as input the public parameters pp, a global state V, a statement s_i, and a proof π_i. It outputs true or false.

A revocable proof system must be correct and secure. By correct, we mean that genuine proofs for valid elements should verify against the corresponding global state. By secure, we mean that it should be difficult to find a proof for a revoked element; that is, it should be computationally hard for an adversary to produce a revoked set such that a proof for a revoked element still verifies against the updated global state. More formally, correctness and security are defined as follows:

[4] Although for ease of notation the output includes n proofs, security dictates that proofs for elements in T should not verify.

Definition 1 (Correctness[5]). *A revocable proof system is* correct *if for every set S, every set $T \subseteq S$, and every $s_i \in S \setminus T$,*

$$\Pr\left[\begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ V, (\pi_1, \pi_2, \ldots, \pi_n) \leftarrow \mathsf{ComputeState}(\mathsf{pp}, S) \\ V', (\pi'_1, \pi'_2, \ldots, \pi'_n) \leftarrow \mathsf{Revoke}(\mathsf{pp}, S, T, V, (\pi_1, \pi_2, \ldots, \pi_n)) \\ \mathsf{Verify}(\mathsf{pp}, V, s_i, \pi_i) = \mathsf{true} \\ \mathsf{Verify}(\mathsf{pp}, V', s_i, \pi'_i) = \mathsf{true} \end{array}\right] = 1$$

Definition 2 (Security). *A revocable proof system is* secure *if for every p.p.t. adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{rcl} \mathsf{pp} & \leftarrow & \mathsf{Setup}(1^\lambda) \\ S, T, s^*, \pi^* & \leftarrow & \mathcal{A}(1^\lambda, \mathsf{pp}) \\ V, (\pi_1, \pi_2, \ldots, \pi_n) & \leftarrow & \mathsf{ComputeState}(\mathsf{pp}, S) \\ V', (\pi'_1, \pi'_2, \ldots, \pi'_n) & \leftarrow & \mathsf{Revoke}(\mathsf{pp}, S, T, V, (\pi_1, \pi_2, \ldots, \pi_n)) \\ s^* \in T & & \\ \mathsf{Verify}(\mathsf{pp}, V', s^*, \pi^*) & = & \mathsf{true} \end{array}\right] \leq \mathsf{negl}(\lambda)$$

3 Main result

Our main result is an inequality describing the relationship between the size of the global state and the number of proofs of valid statements which must be updated. We first introduce the notion of a k-good revoked set; that is, a set of statements such that when these statements are revoked, none of their proofs still verify, and at most k still-valid statements' proofs must be changed.

Definition 3 (k-good revoked set). *We say a revoked set T is* k-good *given a revocable proof system* RPS*, an initial valid set S, size parameter $n = |S|$, and public parameters* pp *if, for $V, (\pi_1, \pi_2, \ldots, \pi_n) \leftarrow \mathsf{ComputeState}(\mathsf{pp}, S)$ and $V', (\pi'_1, \pi'_2, \ldots, \pi'_n) \leftarrow \mathsf{Revoke}(\mathsf{pp}, V, T)$ if:*

1. *For every revoked statement $s_i \in T$, $\mathsf{Verify}(\mathsf{pp}, V', s_i, \pi_i) = \mathsf{false}$*
2. *There are at most k non-revoked statements $s_j \in (S \setminus T)$ such that $\mathsf{Verify}(\mathsf{pp}, V', s_j, \pi_j) = \mathsf{false}$*

Condition (1), that the original proof of a revoked statement no longer verifies, is a consequence of the security requirement that should hold for most revoked sets. Condition (2) is that few ($\leq k$) proofs of non-revoked statements

[5] While correctness with probability 1 is standard for the schemes we consider, our main result (Theorem 1) still holds for a relaxed notion of correctness which holds with overwhelming probability. In fact, it does not rely on correctness at all and rather on the prevalence of a notion of a k-good revoked set. A revocable proof system that is secure, correct with overwhelming probability, and has few witness changes should have many k-good revoked sets.

no longer verify (i.e., need to be updated). Suppose that we want to have a secure revocable proof system such that most of the time, at most k non-revoked statements change when a set of size m is revoked. This is equivalent to having many k-good revoked sets of size m. By security (regardless of $|V|$), an overwhelming fraction of sets of size m must satisfy condition (1); otherwise an adversary could choose a set of size m at random, revoke it, and succeed in finding a proof for a revoked element. Condition (2) is exactly the other property we want: that at most k non-revoked statements change when our set is revoked. Therefore, our desired revocable proof system must have many k-good revoked sets of size m.

We now show that if any public parameters yield many k-good revoked sets, the size of the global state must be large. In other words, if the size of the global state is small, many proofs of non-revoked statements must change when an average set is revoked.

Theorem 1. *Let* $\mathsf{RPS} = (\mathsf{Setup}, \mathsf{ComputeState}, \mathsf{Revoke}, \mathsf{Verify})$ *be a revocable proof system satisfying correctness, and let* pp *be any public parameters occurring with nonzero probability over* $\mathsf{Setup}(1^\lambda)$*. Let* S *be a set of size* n*, and let* $\mathcal{X}_k^*$ *denote the set of subsets* $T \subseteq S$ *that are* k*-good given* RPS, S*, and public parameters* pp*. Then* $|V|$*, the size of the global state in bits, satisfies*

$$|V| \geq \lg |\mathcal{X}_k^*| - \lceil k \lg n \rceil$$

Proof. We show that if $|V|$ is any smaller, there exists an efficient encoding of $\mathcal{X}_k^*$ using fewer than $\lg |\mathcal{X}_k^*|$ bits, contradicting Shannon's Coding Theorem [20]. Note that $\mathcal{X}_k^*$ can be computed by trying every revoked set $T \subseteq S$ and determining whether it fits the conditions of a k-good revoked set. While this algorithm is not efficient, it does not need to be as the contradiction we derive is compression beyond information theoretic limits, which poses no computational bounds on the communicating parties.

Consider two parties $\mathcal{A}$ and $\mathcal{B}$ interacting with a challenger in a game given as input S and pp. The goal is for $\mathcal{A}$ to succinctly encode a uniformly chosen revoked set $T \subseteq S$ for $\mathcal{B}$ to decode. The challenger computes the initial global state and proofs $V_0, (\pi_1, \pi_2, \ldots, \pi_n) \leftarrow \mathsf{ComputeState}(\mathsf{pp}, S)$. The challenger passes $V_0, (\pi_1, \pi_2, \ldots, \pi_n)$ to both $\mathcal{A}$ and $\mathcal{B}$. $\mathcal{A}$ chooses T uniformly at random from $\mathcal{X}_k^*$ and computes the updated global state $V \leftarrow \mathsf{Revoke}(\mathsf{pp}, V_0, T)$. Then, for each $s_i \in (S \backslash T)$, $\mathcal{A}$ checks whether its proof verifies; i.e., whether $\mathsf{Verify}(\mathsf{pp}, V, s_i, \pi_i) = \mathsf{true}$. If *not*, $\mathcal{A}$ adds s_i to a list L of still-valid statements with changed proofs. $\mathcal{A}$ sends V and L to $\mathcal{B}$.

We now show that given V and L, $\mathcal{B}$ can decode T exactly. Let $\mathcal{B}$'s decoding be the set T' consisting of all statements s_i such that $\mathsf{Verify}(\mathsf{pp}, V, s_i, \pi_i) = \mathsf{false}$ and $s_i \notin L$. First, any statement $s_i \in T$ must be in T', since by definition of a k-good revoked set[6], no proofs for revoked statements verify. Therefore, $T \subseteq T'$. Next, $\mathcal{B}$'s decoding algorithm ensures any statement $s_i \in T'$ is not in L, and

[6] It is tempting to instead cite security of a revocable proof system here, but security guarantees only that for *most* revoked sets T, proofs of revoked statements do not verify. Our definition of k-good gives us exactly what we need.

$\mathsf{Verify}(\mathsf{pp}, V, s_i, \pi_i) = \mathsf{false}$. All elements s_j that were not revoked (i.e., not in T) and whose proofs no longer verify ($\mathsf{Verify}(\mathsf{pp}, V, s_i, \pi_i) = \mathsf{false}$) are included in L. Since s_i is not in L, it must in fact have been revoked, so $s_i \in T$. Therefore, $T' \subseteq T$, which implies that $T' = T$.

Finally, we observe that $\mathcal{A}$ can encode L by listing a $(\lg n)$-bit representation of each of its elements. Since $|L| \leq k$ by definition of $\mathcal{X}_k^*$, this encoding takes at most $\lceil k \lg n \rceil$ bits, and $\mathcal{A}$ sends $|V| + \lceil k \lg n \rceil$ bits in total after choosing T. Since T was chosen uniformly from $\mathcal{X}_k^*$, and the entropy of the uniform distribution over $\mathcal{X}_k^*$ is $\lg |\mathcal{X}_k^*|$, we have by Shannon's Coding Theorem that $|V| + \lceil k \lg n \rceil \geq \lg |\mathcal{X}_k^*|$. □

3.1 No useful trade-offs for sublinear state size

We show that under certain regimes (when $|\mathcal{X}_k^*|$ includes at least a constant fraction of subsets of size m for $\lg n \leq m \leq \frac{n}{2}$), there is no useful trade-off between the global state size and the frequency of proof changes when m elements are deleted. That is, the global state size is either linear in the size of the stored set, or $\Omega\left(\frac{m}{\lg n}\right)$ proofs must be updated.

Corollary 1 (No useful trade-offs). *Let n be the size of the initial valid set S and $m \leq \frac{n}{2}$ be the number of deleted elements.*[7] *If $|\mathcal{X}_k^*|$ includes at least a constant fraction of subsets $T \subseteq S$ of size m, and the global state size is $|V| = o(\lg \binom{n}{m})$, then $k = \Omega\left(\frac{m}{\lg n}\right)$.*

Proof. This holds by a straightforward application of Theorem 1. First, observe that the number of possible $T \subseteq S$ of size m is $\binom{n}{m} \geq \frac{n^m}{m^m} \geq 2^m$. $\mathcal{X}_k^*$ includes at least a constant fraction of these subsets T of size m, so $\lg |\mathcal{X}_k^*| = \Omega(\lg \binom{n}{m}) = \Omega(m)$. In order for the inequality from Theorem 1 to hold, we must have $k \lg n \geq \Omega(\lg \binom{n}{m}) - o(\lg \binom{n}{m})$, or $k = \frac{\Omega(\lg \binom{n}{m})}{\lg n} = \Omega\left(\frac{m}{\lg n}\right)$.

This bound on k holds for *any* global state size $|V|$ that is sublinear in $\lg \binom{n}{m}$. Once the global state size becomes $\Omega(\lg \binom{n}{m})$, we can (asymptotically) store the full list of deleted elements and require no witness updates. One especially interesting regime for this bound is when $m = \Theta(n)$ and $|V| = o(n)$. Then Corollary 1 implies that $k = \Omega\left(\frac{n}{\lg n}\right)$. In other words, if we want to avoid a near-constant fraction of proof updates, we need a linear global state size, at which point we can (asymptotically) store the full state naively and require no witness updates. This suggests that there is no asymptotically useful trade-off between global state size and number of proof changes in this regime; at least one of the two must be (nearly) linear.

[7] If more than $\frac{n}{2}$ elements are deleted in sequence, as in stateless blockchains, we can set $m = \frac{n}{2}$ since there must be an intermediate point where $\frac{n}{2}$ elements were deleted, and this bound still applies.

3.2 Persistence requires linear storage

We now show that another desirable property, which we call *persistence*, is also not possible without linear global state. Suppose that we want proofs of certain statements to always verify as long as those statements remain true. This guarantee would be very useful in cryptocurrencies, allowing a user to stay offline until she is ready to make a transaction, without fear of her proof becoming stale. We call this notion *persistence* and formalize it below.

Definition 4 (Persistence). *A statement $s_i \in S$ is* persistent *given initial valid set S of size n and public parameters* pp *if for all $T \subseteq S$ such that $s_i \notin T$,*

- $V, (\pi_1, \pi_2, \ldots, \pi_n) \leftarrow \mathsf{ComputeState}(\mathsf{pp}, S)$
- $V', (\pi'_1, \pi'_2, \ldots, \pi'_n) \leftarrow \mathsf{Revoke}(\mathsf{pp}, S, T, V, (\pi_1, \pi_2, \ldots, \pi_n))$
- $\mathsf{Verify}(\mathsf{pp}, V', s_i, \pi_i) = \mathsf{true}$

A corollary of Theorem 1 shows that there can be very few persistent statements:

Corollary 2 (Persistence requires linear storage). *Let* RPS *be a secure and correct revocable proof system such that there exists an initial set S and a set $S^* \subseteq S$ such that*

$$\Pr_{\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)} [\textit{every } s \in S^* \textit{ is persistent}] > \frac{1}{2}$$

Then the the global state of RPS *has size at least $|S^*| - 1$.*

Proof. Let S be any initial set and S^* be any subset of S. We wish to show that with high probability, $\mathcal{X}_0^*$ is large, where $\mathcal{X}_0^*$ is the family of revoked sets that require no witness changes and for which proofs of revoked statements do not verify. We first argue that by security, few revoked sets $T \subseteq S^*$ yield proofs of revoked statements that still verify. Then it follows that $\mathcal{X}_0^*$ contains all other revoked subsets of S^*, since by definition of persistence they require no witness changes.

Suppose for the sake of contradiction that for *all* parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ that occur with nonzero probability and for which every $s \in S^*$ is persistent, more than half of the revoked sets $T \subseteq S^*$ yield a global state such that the proof of a revoked statement verifies. Then the following adversary $\mathcal{A}$ forges a proof with non-negligible probability, breaking security. Let $\mathcal{A}$ compute $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and $V, (\pi_1, \pi_2, \ldots, \pi_n) \leftarrow \mathsf{ComputeState}(\mathsf{pp}, S)$. $\mathcal{A}$ then chooses $T \subseteq S^*$ uniformly at random, computes $V', (\pi'_1, \pi'_2, \ldots, \pi'_n) \leftarrow \mathsf{Revoke}(\mathsf{pp}, V, T)$, and checks whether $\mathsf{Verify}(\mathsf{pp}, V', s_i, \pi_i)$ for each $s_i \in T$. If $\mathcal{A}$ finds such an s_i, it outputs S, T, s_i, π_i. Independently, $\mathcal{A}$ chooses pp such that all of S^* is persistent with probability at least $\frac{1}{2}$ and T such that the proof of a revoked statement verifies with probability at least $\frac{1}{2}$. Thus, $\mathcal{A}$ is efficient and succeeds with probability $\frac{1}{4}$, contradicting security. Therefore, there must be some parameters pp occurring with nonzero probability such that all of S^* is persistent and at least

half of the revoked sets $T \subseteq S^*$ have no revoked statements whose original proofs verify. The family of these sets is exactly $\mathcal{X}_0^*$, whose size is at least $\frac{1}{2} \cdot 2^{|S^*|}$.

Thus, there exist public parameters pp occurring with nonzero probability such that $|\mathcal{X}_0^*| \geq \frac{1}{2} \cdot 2^{|S^*|}$. Applying Theorem 1 for $k = 0$, we have that the size of the global state is at least $\lg |\mathcal{X}_k^*| = |S^*| - 1$.

4 Implications for authenticated data structures

We show that cryptographic accumulators, vector commitments, and authenticated dictionaries, are instances of revocable proof systems. Thus, our lower bound from Theorem 1 applies. This result is of interest since these data structures are frequently used in distributed settings, in which users maintain proofs of portions of the committed data that are verified against a global state. Our bound dictates that these users must update their proofs frequently as the global state changes.

4.1 Cryptographic accumulators

A cryptographic *accumulator* [2,7], also called a *set commitment*, commits to an accumulated set X via a succinct digest A. Different accumulator schemes support efficiently proving various properties about the accumulated set X, such as membership or non-membership of elements. Some schemes may also allow X to be modified and the corresponding proofs updated. A typical accumulator supports additions, deletions, and membership proofs. That is, given a set X there is a function computing a digest A and a membership proof (also called a *witness*) w_i for each $x_i \in X$, corresponding to the $\mathsf{ComputeState}$ function of a revocable proof system. When a new element x is added to X, a new global state A' can be computed using x and A. Furthermore, each membership proof w_i can be updated given x and A. When an element $x_i \in X$ is deleted, a new global state A' can be computed using x_i, A, and the membership proof w_i for x_i. The membership proofs of the other elements of X can be updated using the same information. Some accumulator schemes also allow batch updates, where multiple elements can be efficiently added and/or deleted at once [3].

Constructing a revocable proof system using an accumulator. We show how an RPS can be constructed using an accumulator scheme Acc supporting addition, deletion, and membership proofs. Addition is only necessary for the initial set S. The Setup function for our RPS calls the Setup function for Acc. The $\mathsf{ComputeState}$ function for our RPS, given public parameters pp and a valid set S, adds S to our accumulator given pp to obtain a digest A and a membership witness w_i for each $s_i \in S$. We let the proof π_i for s_i be this membership witness w_i. We implement Revoke for our RPS by, given a set $T \subseteq S$, removing T from the accumulated set and updating all witnesses according to the accumulator scheme. The resulting global state is the resulting accumulator value A', and the

resulting proofs π'_i are the updated witnesses w'_i. We let the Verify function for our RPS be the same as the Verify function for the accumulator scheme.

Accumulator schemes have correctness and security definitions that are analogous to those of a revocable proof system; full definitions can be found in [6]. By correctness of the accumulator, membership witnesses for elements of the accumulated set (equivalently, valid statements) verify. By security of the accumulator, it is hard for an adversary to find verifying membership witnesses for elements not in the accumulated set (equivalently, revoked or invalid statements). Thus, this construction is indeed a revocable proof system, and our lower bound from Theorem 1 applies.

We note that we can also construct a revocable proof system using an accumulator that supports only addition and non-membership witnesses (but not deletion). Given a finite data universe U and a set $X \subseteq U$, a delete/membership accumulator storing X can be implemented using an add/non-membership accumulator storing $U \setminus X$.

Camacho-Hevia result Our accumulator lower bound is reminiscent of a lower bound proved by Camacho and Hevia [6]. They consider a dynamic accumulator supporting addition, deletion, and membership proofs. Their model allows batch updates: if $w_1, \ldots, w_n$ are witnesses for an initial accumulated set X, after deletion of a set T the state-update function outputs a string $Upd_{X,X\setminus T}$ that can be used to update all witnesses to $w'_1, \ldots, w'_n$ to reflect the updated state. They show that if there are $|T| = m$ deletions, $Upd_{X,X\setminus T}$ must have length $\Omega(m)$. Baldimtsi et al. show an analogous result for a universal accumulator supporting addition, deletion, and *non-membership* proofs, using the same proof style [1]. While these results are similar in spirit to ours, they do not address how this string $Upd_{X,X\setminus T}$ is incorporated into the new witnesses or how many witnesses must change. It is possible in this model that some elements require very long witness changes, while nearly all other witnesses can remain the same. Our result addresses the separate question of how many witness changes are required.

We note a small gap in the Camacho-Hevia proof (and similarly in the Baldimtsi et al. proof) in Section A of the appendix. In our proof, we address this issue by defining the notion of a k-good revoked set.

4.2 Vector commitments

A *vector commitment (VC)* [8] stores a vector $\mathbf{v} = [v_1, \ldots, v_k]$ in the form of a succinct digest C. For each index i and corresponding component v_i, the scheme produces a proof π_i that can be used alongside C to verify that $\mathbf{v}_i = v_i$. When a component is changed, the digest and proofs of some or all components may change. Correctness dictates that properly generated proofs of true components verify with their corresponding digests. Security dictates that it is hard to find a proof for an incorrect component. Recently several vector commitment schemes have been constructed with cryptocurrency applications in mind; see [23,21,25,9,15].

Constructing a revocable proof system using a vector commitment scheme Our construction commits to a vector storing valid statements. In describing our construction, we use the syntax for a VC scheme from [21]. Let q be an upper bound on the total number of valid statements. Let $\perp$ be some special value used to denote that there is no statement stored at that vector position. The Setup function for our RPS calls the KeyGen function of the VC scheme with security parameter λ and vector length n (the size of our initial valid set) to obtain public parameters pp. The $\mathsf{ComputeState}$ function, given pp and an initial valid set S of size n, first calls the commitment function of the VC scheme on the vector $[s_1, s_2, \ldots, s_n, \perp, \ldots, \perp]$ (using some arbitrary ordering of S). This outputs a commitment C that is the global state, along with auxiliary information aux. To generate the proof w_i for each s_i, $\mathsf{ComputeState}$ then calls the Open function of the VC given i, s_i, and aux. It outputs the commitment C and a proof w_i for each s_i. The Revoke function of our RPS, given $T \subseteq S$, revokes each statement $s_i \in T$ by setting the corresponding position of the committed vector to $\perp$. We describe how to do so assuming no batch updates, updating the state and all proofs for each revocation before moving onto the next. More precisely, for each $s_i \in T$, it calls $\mathsf{VC.Update}(C, s_i, \perp, i)$ to obtain an updated state C' and update information U. It then updates the proof w_j for each other component s_j using $\mathsf{VC.ProofUpdate}$ given C, w_j, s_i, i, U. After all updates have been made, it outputs all proofs and the resulting commitment. Finally, the Verify function of our RPS, given C, s_i, w_i, calls $\mathsf{VC.Ver}(C, s_i, j, w_i)$ for each vector component j. Verify outputs true if and only if there exists a j such that $\mathsf{VC.Ver}$ outputs true.

We give an overview of how correctness and security for a VC scheme relate to the corresponding definitions for a revocable proof system; full definitions of correctness and security for a VC scheme are given in [8]. VC schemes offer correctness with overwhelming probability, ensuring that properly generated proofs for committed components verify. See footnote 5 for a discussion of how this compares to correctness with probability 1 for a revocable proof system. The security definition for a VC guarantees that it's hard for an adversary to find two valid proofs for different values s_i and s_i' of the i^{th} component. This implies security of our constructed revocable proof system: if an adversary finds a proof w^* of a statement s^* that is *not* in the valid set, it has succeeded in finding a proof that the value of the vector at some index i is s^*. Since s^* is not in the valid set, the actual value at i must be $\perp$ or some other s'. The proof w of this other value yields a pair of proofs that verify for different values at index i. Thus, our constructed scheme is an RPS.

4.3 Authenticated dictionary

A related notion is an *authenticated dictionary* [14,18], which produces a commitment to a set of key-value pairs, such that proofs of these stored pairs can be generated and verified against the commitment. Throughout time, more key-value pairs can be added, and existing pairs can be modified. When the dictionary is updated, a new shared commitment is generated, potentially invalidating

old proofs. The existence of these proofs both for the original data and the updated data corresponds to correctness for a revocable proof system. Security of an authenticated dictionary guarantees that it is difficult to generate proofs of a key-value pair that is not in the stored set. The argument that we can construct a revocable proof system from an authenticated dictionary is along the same lines as the arguments from vector commitments and accumulators. One way to see this is to observe that we can construct a vector commitment scheme using an authenticated dictionary, by storing the vector index-value pairs as key-value pairs in the dictionary. Authenticated dictionaries therefore fit the framework of a revocable proof system, and thus our lower bound holds, implying that proofs must be updated often.

Aardvark [16], a recently proposed distributed authenticated dictionary with applications to stateless blockchains, proposes an interesting versioning scheme to overcome the need to change witnesses enough to accommodate many users making transactions concurrently. We discuss this idea further in Section 5.2.

5 Implications for blockchains

Blockchains typically operate in one of two models: the *unspent transaction output (UTXO)* model or the *account-based* model. A stateless blockchain functions slightly differently in each of these models. We describe the models below and argue that each requires the functionality of a revocable proof system, meaning that our lower bound from Theorem 1 holds.

UTXO model. In the UTXO model, the global state stores the set of unspent coins. When a user wants to make a transaction, they must specify the coin(s) (UTXOs) they wish to spend and submit a proof that these coins are unspent. A stateless blockchain needs to satisfy correctness: a proof for an unspent coin should verify against the corresponding global state. If the transaction is successful, the global state is updated, and the spent coins' proofs should no longer verify. In order to prevent users from double spending, it should be computationally hard to produce a proof for a spent coin—this is equivalent to the definition of security for a revocable proof system. A stateless blockchain in the UTXO model is commonly constructed using a dynamic accumulator, where the accumulated set is the set of valid UTXOs. Such accumulators include RSA accumulators [3], Merkle-tree-based accumulators [13,5], and Verkle trees [4].

Account-based model. In the account-based model, the global state stores a list of account-balance pairs. Each account owner, or user, maintains a proof of their account balance. When a user u wants to make a transaction, they submit a proof π that their account-balance pair is included in the global state. The validator verifies the user's account balance using this proof, and they check that the balance is high enough to make the desired transaction. The amount spent is then deducted from the user's balance, and the global state is updated accordingly.

In the context of a revocable proof system, the valid set is the set of account-balance pairs. An account-balance pair is revoked when the corresponding user makes a transaction, changing their account balance. Security ensures that it is hard to generate a proof for an incorrect account-balance pair. Correctness ensures that every user can prove that their true account balance is valid. An account-based blockchain is often constructed using a vector commitment or authenticated dictionary, where each index of the vector represents an account and the value is that account's balance (e.g., [21]).

5.1 Interpreting our bound in practice

An interesting question is exactly what implications Theorem 1 has for practical stateless blockchains. Toward answering this, we graph the number of witness (or proof) changes for various parameter values.

We first apply Theorem 1 to obtain a lower bound on the number of witness changes required after some number m deletions, given an initial valid set of size n. The number of possible deleted sets of size m is $\binom{n}{m} \geq \frac{n^m}{m^m}$ (by, e.g., [10]). Ideally, we would like at least half of these sets to (1) require few ($\leq k$ for some k) witness changes, and (2) allow no deleted elements to be double spent. These are exactly the conditions for a k-good revoked set; thus, in our application of Theorem 1 we can set $|\mathcal{X}_k^*| = \frac{1}{2}\binom{n}{m} \geq \frac{n^m}{2m^m}$. Our next step is to obtain a lower bound for k, showing that many witnesses must change.

Rearranging, we have $\lceil k \lg n \rceil \geq \lg \frac{n^m}{2m^m} - |V|$. Simplifying further,

$$k \lg n \geq m \lg n - m \lg m - |V| - 2$$

$$k \geq m - \frac{|V| + m \lg m}{\lg n} - \frac{2}{\lg n} \tag{1}$$

Let $f(m, n, |V|)$ denote the right hand side of Equation 1. We graph f, showing that if at least half of the sets of size m are k-good revoked sets, k must be at least $f(m, n, |V|)$. In our graphs, we use two natural values of n. The first is 2^{33}, approximately the world's current human population. The second is 2^{26}, approximately the current number of UTXOs in Bitcoin [12]; these graphs are included in Section B of the appendix.

In Figure 1, we can see that the relationship between f and m is approximately linear, with the $\frac{m \lg m}{\lg n}$ term having little impact since m is small relative to n in our ranges. Furthermore, increasing the size of the global state V results in a horizontal shift of the curve and has little benefit until it becomes very large.

Like Figure 1, Figure 2 shows that there is not a useful trade-off between the global state size and the number of witness changes per day. The global state size must become very large, at least 2^{22} for most throughput values, before there is much impact on the number of witness changes. This concrete effect mirrors the asymptotic relation of by Corollary 1.

While the number of witness changes may seem small in comparison to the number of UTXOs or accounts in the system, without some additional recovery

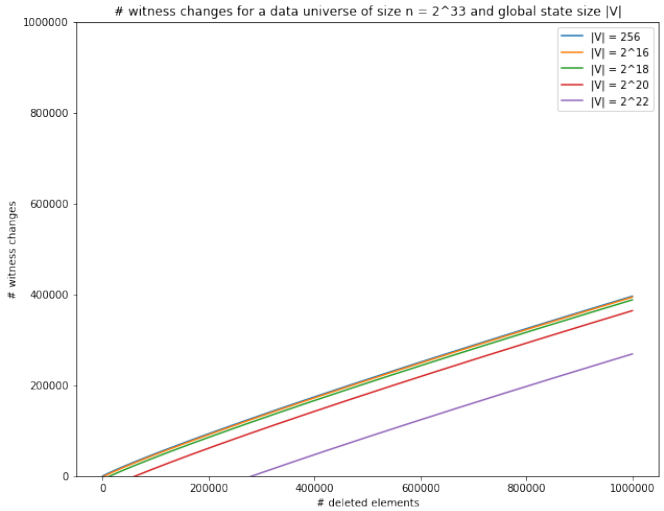

Fig. 1. Number of witness changes $f(m, n, |V|)$ given $0 \leq m \leq 10^6$ deleted elements, a data universe of size $n = 2^{33}$, and varying global state size $|V|$.

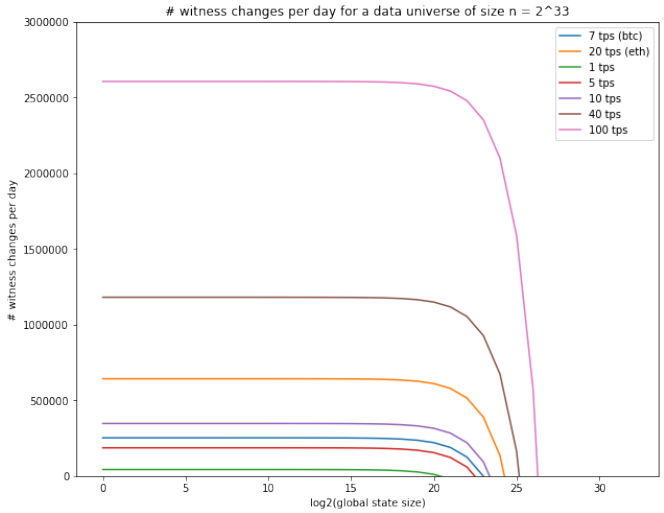

Fig. 2. Number of witness changes per day for a data universe of size $n = 2^{33}$ and varying global state size, for blockchains with various throughput. In particular, Bitcoin and Ethereum support roughly 7 and 20 transactions per second respectively.

mechanism, the consequences of a user missing their witness update are severe as they will no longer be able to make transactions. Furthermore, if the system has enough throughput to adequately serve the data universe, there will be many more witness changes: for 24,000 transactions per second (the maximum throughput supported by Visa [24]) the number of witness changes per day for $n = 2^{33}$ becomes roughly 1.25×10^8, as shown in Figure 5. Our graphs show that if most users are not willing to refresh their witnesses continually, hundreds of thousands of these users will lose their coins per day. As a result, most stateless blockchain proposals have included a way for lazy users to obtain updated proofs, at the cost of more storage for certain parties; the most prominent such solution uses proof-serving nodes (PSNs). Below, we discuss two more limited solutions (a versioning model and a partially persistent model), then conclude with a discussion of PSNs and potential future work relating to them.

5.2 Versioning model

An issue arises when at some time t, many users simultaneously provide a proof of their account balance (an element in the authenticated dictionary) and a transaction that they wish to make (an update of their element in the dictionary). If the transactions are executed in sequence, each user's transaction requires updating the dictionary, invalidating the other users' proofs. One solution is to store this set of transactions temporarily, so we can verify each user's proof against the global state at time t and then check manually that none of the subsequent transactions changed that user's account balance. We call this a versioning system.

Aardvark [16], an authenticated dictionary designed with cryptocurrency applications in mind, does essentially this: it stores all transactions that happen in the next τ time, for some tunable time parameter τ. The current state commitment at time t is also stored. At a future time up to $t + \tau$, any proof at least as recent as time t can still be verified by checking it against the state commitment at time t, then naively checking that it does not conflict with the cached transactions. This approach essentially ensures that proofs do not need to change for k transactions by storing k additional state, where k is the number of transactions happening in time τ. This matches our lower bound from Corollary 2 (up to constants), which when translated to this setting says that if we want no proof changes when deleting k elements, we must store at least k state. Thus, this versioning scheme is essentially the best one can hope to achieve without introducing parties such as PSNs storing more state (see, e.g., [19,23,21]).

5.3 Partially persistent model

A desirable feature of a stateless blockchain is that users know in advance when their proofs will need to change, so they can go online only at that time. Perhaps users could pay a fee for the guarantee that their proofs will remain valid for some number of transactions in the future. A natural question is how much additional state is necessary to accommodate these special requests.

This property is exactly our notion of *persistence*: the persistent set S^* corresponds to the set of proofs that are guaranteed to remain valid. Unfortunately, Corollary 2 says that any secure and correct revocable proof system with a persistent set S^* must have global state size at least $|S^*| - 1$. If any significant portion of the user base wants persistent proofs, the stateless blockchain model does essentially no better than storing the full state.

We can achieve persistence if users are willing to lock up their coins for a set period of time. That is, a user wanting their proof to remain valid for at least a day would sacrifice their ability to spend their coin during that day. We could then separate the blockchain into two state commitments: one state S_1 storing the set of locked coins and another state S_2 storing all other (liquid) coins. Since locked coins can only be spent at the end of the day, S_1 remains the same and no proofs of locked coins change throughout the day. At the end of the day, users may unlock their coins and move them from S_1 to S_2. We could extend this scheme to support other time ranges, incurring the cost of extra storage as more time ranges are supported.

While potentially helpful in limited settings, this model has serious drawbacks for general use. The most obvious is the fact that users cannot spend their locked coins. Furthermore, the benefits are all-or-nothing in the following way: If a user wants to maintain *any* liquid coins, they must continually update these liquid coins' witnesses, at which point updating their locked coins' witnesses would require minimal additional effort.

5.4 Proof-serving node model

Prior work proposes offloading witness updates to a *proof-serving node (PSN)* [19,23,21]. Instead of maintaining its proof itself, a user can delegate this task to a PSN and come online only when it wishes to make a transaction. In any revocable proof system, the PSN can update a user's proof simply by using the Revoke algorithm. The storage required for this simple approach scales with the number of users: the PSN can serve k users by storing only these users' proofs and constantly checking for updates. This property that PSNs can use storage proportional to the number of proofs they maintain somewhat mitigates the centralization issues posed by requiring storing a large state, allowing anyone to operate a small PSN. The PSN model is especially promising in light of our result that there is no holy grail revocable proof system achieving few witness updates on its own.

However, centralization is still a major concern with PSNs, and the PSN model raises interesting questions regarding incentives. PSNs must be incentivized in some way to do this work. Hyperproofs [21] suggests a PSN model where users pay PSNs to maintain their proofs for them. This payment model seems to have an interesting relationship with batch updates, which hyperproofs also allow. That is, while it takes a user time t to update a single proof, a PSN can update the proofs of all n users in the system in time $t \cdot f(n)$ (for some sublinear function f). PSNs that serve enough users to take advantage of batch updates can offer much cheaper prices than small PSNs. There can only be a

few PSNs that serve this many users. The resulting system will have a few PSNs storing the full state, and the users they serve will store nothing. This is a significant risk: an adversary that attacks these PSNs can compromise the entire blockchain, preventing many users from spending their coins.

References

1. Baldimtsi, F., Camenisch, J., Dubovitskaya, M., Lysyanskaya, A., Reyzin, L., Samelin, K., Yakoubov, S.: Accumulators with applications to anonymity-preserving revocation. In: IEEE Euro S&P (2017)
2. Benaloh, J., Mare, M.d.: One-way accumulators: A decentralized alternative to digital signatures. In: Eurocrypt (1993)
3. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: Annual International Cryptology Conference (2019)
4. Buterin, V.: A state expiry and statelessness roadmap. `https://notes.ethereum.org/@vbuterin/verkle_and_state_expiry_proposal`
5. Buterin, V.: The stateless client concept. `https://ethresear.ch/t/the-stateless-client-concept/172` (2017)
6. Camacho, P., Hevia, A.: On the impossibility of batch update for cryptographic accumulators. In: LatinCrypt (2010)
7. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: CRYPTO (2002)
8. Catalano, D., Fiore, D.: Vector commitments and their applications. In: PKC (2013)
9. Chepurnoy, A., Papamanthou, C., Srinivasan, S., Zhang, Y.: Edrax: A Cryptocurrency with Stateless Transaction Validation. Cryptology ePrint Archive, Paper 2018/968 (2018)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press (2022)
11. Dai, W.: b-money. `http://www.weidai.com/bmoney.txt` (1998)
12. Delgado-Segura, S., Pérez-Sola, C., Navarro-Arribas, G., Herrera-Joancomartí, J.: Analysis of the Bitcoin UTXO set. In: Financial Crypto (2018)
13. Dryja, T.: Utreexo: A dynamic hash-based accumulator optimized for the bitcoin utxo set. Cryptology ePrint Archive, Paper 2019/611 (2019)
14. Goodrich, M.T., Shin, M., Tamassia, R., Winsborough, W.H.: Authenticated dictionaries for fresh attribute credentials. In: Trust Management (2003)
15. Gorbunov, S., Reyzin, L., Wee, H., Zhang, Z.: Pointproofs: Aggregating proofs for multiple vector commitments. In: ACCM CCS (2020)
16. Leung, D., Gilad, Y., Gorbunov, S., Reyzin, L., Zeldovich, N.: Aardvark: An Asynchronous Authenticated Dictionary with Applications to Account-based Cryptocurrencies. In: USENIX Security (2022)
17. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
18. Papamanthou, C., Tamassia, R.: Time and space efficient algorithms for two-party authenticated data structures. In: Information and Communications Security (2007)
19. Reyzin, L., Meshkov, D., Chepurnoy, A., Ivanov, S.: Improving Authenticated Dynamic Dictionaries, with Applications to Cryptocurrencies. Cryptology ePrint Archive, Paper 2016/994 (2016)
20. Shannon, C.E.: A mathematical theory of communication. The Bell system technical journal **27**(3), 379–423 (1948)
21. Srinivasan, S., Chepurnoy, A., Papamanthou, C., Tomescu, A., Zhang, Y.: Hyperproofs: Aggregating and maintaining proofs in vector commitments. IACR Cryptol. ePrint Arch. **2021**, 599 (2021)

22. Todd, P.: Making UTXO Set Growth Irrelevant With Low-Latency Delayed TXO Commitments. https://petertodd.org/2016/delayed-txo-commitments (2016)
23. Tomescu, A., Abraham, I., Buterin, V., Drake, J., Feist, D., Khovratovich, D.: Aggregatable subvector commitments for stateless cryptocurrencies. Cryptology ePrint Archive, Paper 2020/527 (2020)
24. Visa: Visa acceptance for retailers. https://usa.visa.com/run-your-business/small-business-tools/retail.html
25. Wang, W., Ulichney, A., Papamanthou, C.: BalanceProofs: Maintainable Vector Commitments with Fast Aggregation. Cryptology ePrint Archive, Paper 2022/864 (2022)
26. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger (2014)

A Camacho-Hevia proof

Like our proof of Theorem 1, the Camacho-Hevia [6] proof uses a compression argument, considering an adversary that computes witnesses w_i for each element x_i of an accumulated set X, then uses this information to reconstruct a random deleted set $T \subseteq X$ of size m. The adversary receives the accumulator value A' for a set $X \setminus T$, along with a string of information used to update each witnesses w_i to a proof w_i' for the new accumulator value A'. The adversary then determines whether each x_i is in T by checking whether w_i' verifies with A'. The authors conclude that the adversary can correctly reconstruct T, and therefore the global state and the witness update information must together be at least $\Omega(m)$ bits long. In part of their proof that the adversary succeeds in learning T, the authors argue that by security, if the proof w_i' for x_i does verify with with A', x_i must not have been deleted.

However, there is a gap in this argument: accumulators have computational security with probability less than 1, meaning that there *exist* proofs w_i' that verify for deleted elements x_i, but they should be hard to find. In fact, since there may be exponentially many (in the security parameter) possible deleted sets T, it is likely that some sets T will yield verifying proofs w_i' for deleted elements. Fortunately, this issue has no impact on their bound asymptotically, since computational security still requires that nearly all sets T lack verifying proofs for deleted elements. We include a condition to capture this issue in our definition of a k-good revoked set, requiring that no proofs of deleted elements verify. The Camacho-Hevia argument works for deleted sets T with this property, and one can show that security implies that there are many such sets T.

B Additional figures

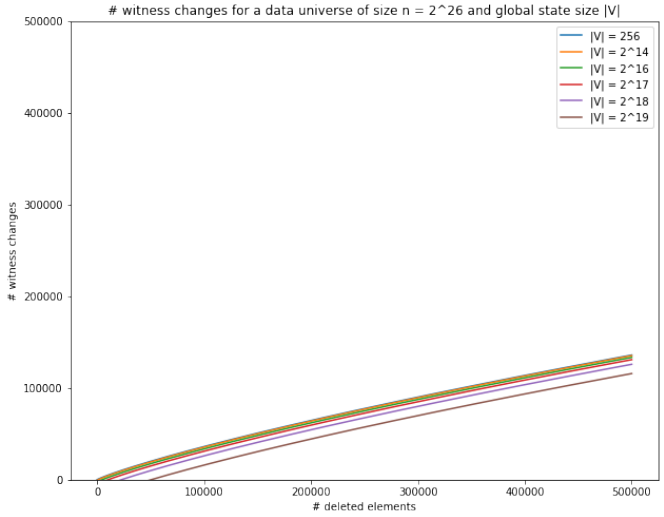

Fig. 3. Number of witness changes $f(m, n, |V|)$ for number of deleted elements m, a data universe of size $n = 2^{26}$, and varying global state size $|V|$.

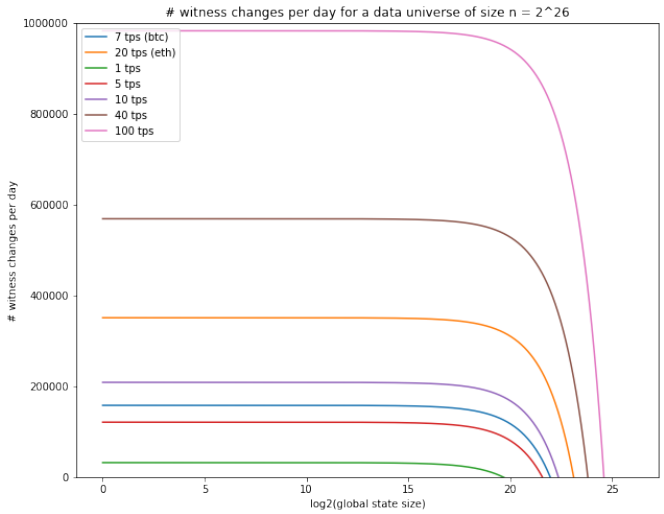

Fig. 4. Number of witness changes per day for a data universe of size $n = 2^{26}$ and varying global state size and throughput. In particular, Bitcoin and Ethereum support roughly 7 and 20 transactions per second respectively.

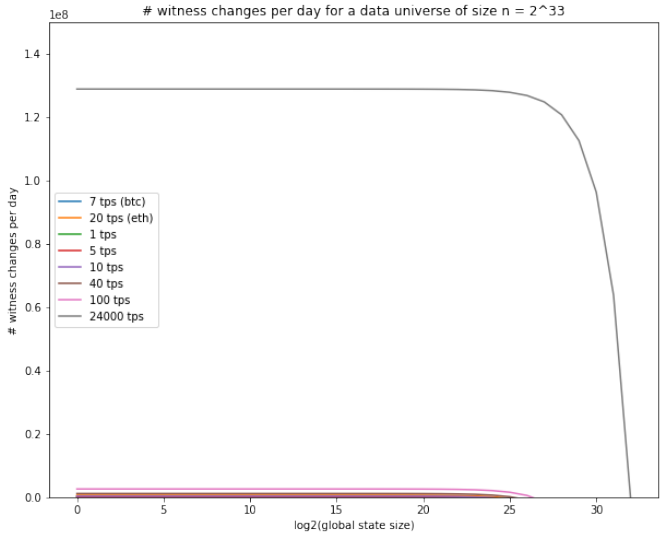

Fig. 5. Number of witness changes per day for a data universe of size $n = 2^{33}$ and varying global state size, for blockchains with throughput up to 24,000 transactions per second.