## 0.1 intro

Our presentation is on a simple proof of sequential work. This paper won the best paper award, and builds upon a paper about **publicly verifiable proofs of sequential work** by Mahmoody, Moran and Vadhan in 2013.

So a proof of sequential work is a protocol for proving that one did sequential computational work related to some statement.

- Ensuring someone did sequential computational work is useful for time-stamping, and the main motivation of this proof is its application in blockchain designs and applicable to cryptocurrencies.
- The proof of sequential work is done with the assumption of a random oracle model. So the prover and verifier, that both have access to a random oracle, which is basically a hash function (will show some properties later)
- so my partner will construct and define a Proof of Sequential Work
- then I will prove two lemmas on the properties of random oracles, 3 graph lemmas and prove the security and efficiency of the protocol

## 0.2 Lemmas

**Lemma 1**: **Random Oracles are Collision Resistant**

- we are given an adversary and a random function. WTS that if A makes at most $q$ queries, the probability that two queries collide is bounded by $\frac{q^2}{2^{w+1}}$
- to prove this, we find the probability of collision between two consecutive queries and add then use the union bound to bound them
- each individual query has length w

Consider an adversary $\mathcal{A}^{\mathsf{H}}$ which is given access to a random function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^w$. If $\mathcal{A}$ makes at most $q$ queries, the probability of two colliding queries $x \neq x', \mathsf{H}(x) = \mathsf{H}(x')$ is at most $\frac{q^2}{2^{w+1}}$

*Proof.* For individual queries $x_1 \leq x_i \leq x_q$, the probability of collision between the $i^{th}$ query with a previous query , (i.e. $P(H(x_i) = H(x_{i-1})))$ is bounded by $\frac{i-1}{2^w}$. So the probability of collision for all $q$ queries is bounded by $\sum_{i=1}^{q} \frac{i-1}{2^w} = \frac{(q-1)(q-1+1)}{2 \cdot 2^w} = \frac{q^2}{2^{w+1}}$. □

**Lemma 2**: **Random Oracles are Sequential**

- an $\mathsf{H}$-sequence of length $s$ is a sequence of $s$ strings $x_0, \ldots, x_s \in \{0,1\}^*$, where for each $i < s$, $H(x_i)$ is a substring of $x_{i+1}$.

Consider an adversary $\mathcal{A}^{\mathsf{H}}$ which is given access to a random function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^w$ that it can query for at most $s-1$ rounds. Each round, $\mathcal{A}^{\mathsf{H}}$ can make arbitrarily many parallel queries. If $A$ makes at most $q$ queries of total length $Q$ bits, then the probability that it outputs an $\mathsf{H}$-sequence $x_0, ..., x_n$ is at most $q \cdot \frac{Q+\sum_{i=1}^{s}|x_i|}{2^w}$

*Proof.* We can divide this into two cases, where (1) $A$ "gets lucky" with one $x_i$, or (2) there's collision, i.e. for some $x_i \neq x_j, \mathsf{H}(x_i) = \mathsf{H}(x_j)$.

- Case 1: for some $0 \leq i < s$, $\mathsf{H}(x_i)$ is a substring of $x_{i+1}$, but $A$ did not query $x_i$. Since $\mathsf{H}$ is a uniformly random function, the probability that $\mathsf{H}(x_i) \subseteq_{i+1}$ for some $i$ and some $a, b$ would at most $q \cdot \frac{|x_i|}{2^w}$. Thus the probability for any $i$ is at most $q \cdot \frac{\sum_{i=0}^{s}|x_i|}{2^w}$, by union bound.
- Case 2: for some $1 \leq i \leq j \leq s-1$ and some queries $x_i$, $x_j$, the probability of collision, i.e. that $x_i \supseteq \mathsf{H}(\mathsf{x_j})$ is bounded by $q \cdot \frac{Q}{2^w}$

Adding the two cases, we get $q \cdot \frac{Q+\sum_{i=1}^{s}|x_i|}{2^w}$. □

**Lemma 3**: The labels of $G_n^{\mathsf{PoSW}}$ can be computed in topological order using only $w \cdot (n+1)$ bits of memory

*Proof.* $n$ is the depth of the graph and $w$ is the output range of the hash function. The proof is a backward induction on the depth of $G_n^{\mathsf{PoSW}}$.

1. First, separate $G_n^{\mathsf{PoSW}}$ into Right and Left subtrees. Each subtree is isomorphic to $G_{n-1}^{\mathsf{PoSW}}$, if we don't take into account the edges going from $label_0$ to leaves on the Right subtree.
2. We calculate $label_0$ on the Left subtree using the space it would take to calculate $G_{n-1}^{\mathsf{PoSW}}$, and keep $label_0$.
3. Then, to calculate $label_1$ on the Right subtree, we need the space it takes to calculate $G_{n-1}^{\mathsf{PoSW}}$, plus $w$ bits to store $label_1$.
4. Then, using only $label_0$ and $label_1$, we calculate the label of the root $label_\epsilon = \mathsf{H}(\epsilon, label_0, label_1)$.

Thus, the memory required to compute $G_n^{\mathsf{PoSW}}$ is the memory it takes to compute $w + G_{n-1}^{\mathsf{PoSW}} = w + w + G_{n-2}^{\mathsf{PoSW}} = k \cdot w + G_{n-k}^{\mathsf{PoSW}}$.

For base case $G_0^{\mathsf{PoSW}}$, theres only 1 node, meaning the root can be can be computed in $w$ bits. So we get $w + G_{n-1}^{\mathsf{PoSW}} = k \cdot w + G_{n-k}^{\mathsf{PoSW}} = n \cdot w + G_0^{\mathsf{PoSW}} = n \cdot w + w = w(n+1)$. □

**Lemma 4:** Take a graph $G_n^{\mathsf{PoSW}} = (V, E)$. For any $S \subseteq V$, the subgraph of $G_n^{\mathsf{PoSW}}$ consisting of nodes $V - D_{S^*}$ has a directed path going through all the leftover nodes (there are $|V| - |D_{S^*}| = N - |D_{S^*}|$ leftover nodes).

*Proof.* This proof is an induction on $n$ for $G_n^{\mathsf{PoSW}}$. $G_0^{\mathsf{PoSW}}$ is obviously true since it contains a single node. Suppose the lemma holds for $G_i^{\mathsf{PoSW}}$. We now want to show it holds for $G_{i+1}^{\mathsf{PoSW}}$. So pick some $G_{i+1}^{\mathsf{PoSW}} = (V, E)$. $G_{i+1}^{\mathsf{PoSW}}$ has a Left and Right subgraph, and root $\epsilon$. The Left and Right subgraphs are isomorphic to $G_i^{\mathsf{PoSW}}$, except for extra edges from from $label_0$ to leaves of the Right subgraph. Consider an arbitrary $S \subseteq V$, and these four cases:

- **case 1:** If node $\epsilon \in S^*$ then we are done because $D_{S^*}$ would be the whole graph and it is vacuously true that $V - D_{S^*}$ has a directed path.
- **case 2:** Suppose nodes $0 \in S^*$, $1 \notin S^*$ then the whole Left subtree would be in $D_{S^*}$. The Right subtree would become equivalent to $G_i^{\mathsf{PoSW}}$ and by assumption the subgraph on $V - D_{S^*}$ has a direct path to 1. Add an edge $1 \to \epsilon$ and we are done.
- **case 3:** Suppose $0 \notin S^*$, $1 \in S^*$. By the same argument as case 2, we can find a direct path going through the leftover nodes.
- **case 4:** Suppose $0 \notin S^*$, $1 \notin S^*$ Then, take the Left subgraph (equivalent to $G_i^{\mathsf{PoSW}}$) and find a directed path ending in node 0. Take the Right subgraph (equivalent to $G_i^{\mathsf{PoSW}}$) and find a directed path starting at leaf $v$. Then, link the Left and Right subgraph by adding edges to $0 \to v$ and $1 \to \epsilon$.

□

**Lemma 5:** For any $S^*, S \subset V$, $D_{S^*}$ contains $|\{0,1\}^n \cap D_{S^*}| = \frac{|D_{S^*}|+|S^*|}{2}$ many leaves

*Proof.* Suppose $S^* = \{v_1, ..., v_k\}$. Then, $D_{v_i} \cap D_{v_j} =$ for $i \neq j$ because $S^*$ is a minimal set. Thus, to find the total number of leaves in $D_{S^*}$, we can sum the number of leaves in each $D_{v_i}$, which is easier since each $D_{v_i}$ is a full binary tree with $\frac{|D_{v_i}|+1}{2}$ leaves. So

$$
\begin{aligned}
|\{0,1\}^n \cap D_{S^*}| &= \textstyle\sum_{i=1}^k |\{0,1\}^n \cap D_{v_i}| \\
&= \textstyle\sum_{i=1}^k \frac{|D_{v_i}|+1}{2} \\
&= \frac{|D_{S^*}|+|S^*|}{2}
\end{aligned}
$$

□

## 0.3 Proof of Security

honest prover will make N queries to prove the statement, and we want to show a dishonest prover must make close to N queries or otherwise get rejected.

**Theorem 1**: Consider a PoSW defined using parameters $N$, $\mathsf{H}$, $t$, and $M$ as defined above, with an additional parameter $\alpha > 0$. $\alpha$ is what the authors call a "soundness gap", which is the percentage difference between $N$ and how many queries to $\mathsf{H}$ a cheating prover $\widetilde{\mathcal{P}}$ actually makes, i.e. a cheating prover $\widetilde{\mathcal{P}}$ will make at most $(1-\alpha)N$ queries. For such a PoSW, the verifier $\mathcal{V}$ will reject with probability $1 - (1-\alpha)^t - \frac{2 \cdot n \cdot w \cdot q^2}{2^w}$.

the first guy (1-a) represents the answer the cheater gave is within soundness gap
the second guy (2nwq...) is the event where there exists a collision or P breaks sequentiality so the cheater gets away

- So $\frac{2 \cdot n \cdot w \cdot q^2}{2^w}$ accounts for the assumptions we make bounded by lemma 1 and lemma 2, that there will be a collision and that $P$ will break sequentiality of $H$: $\frac{q^2}{2^{w+1}} + q\frac{Q+\sum_i^s |x_i|}{2^w}$
- Q is the total number of bits queries, $Q \leq q \cdot w \cdot (n+1)$, where $w \cdot (n+1)$ is the largest possible query. And $q$ is the total number of queries.
- $|x_i|$ in this case is the sequence of labels wich is bounded by $w$ bits, and $s$ is the number of rounds queried and is bounded by $q$ (the total number of queries).
- so $\leq q\frac{q \cdot w \cdot (n+1) + qw}{2^w} + \frac{q^2/2}{2^w} = \frac{q^2 w(n+1) + q^2 w + q^2/2}{2^w} = \frac{q^2(w(n+1)+w+1/2)}{2^w} < \frac{q^2(2wn)}{2^w}$

**Proof**: Consider the probability that $\mathcal{V}$ detects an inconsistent answer to the challenge (challenge is a set of nodes and the answer should be their corresponding labels).

- Let set $S \subseteq V = \{0,1\}^{\leq n}$ be the set of inconsistent vertices. By Lemma 4 there is a path going through all the vertices of $V - D_{S^*}$ (which are all labeled correctly)
- so the path is an $\mathsf{H}_\chi$-sequence of the leftover nodes $N - |D_{S^*}|$. (Where N is the number of all nodes.)

Now we can divide this into two cases:

- the case where the answer is within the soundness gap, then the verifier considers that close enough to $N$ and wont reject it
- the answer is not within the soundness gap, so the verifier will reject with high probability

**Case 1 ($|D_{S^*}| \leq \alpha N$):** $\widetilde{\mathcal{P}}$ must have made at least $(1-\alpha)N$ sequential queries to $\mathsf{H}_\chi$, to compute the $\mathsf{H}_\chi$-sequence of length $N - |D_{S^*}|$. verifier accepts.

**Case 2 ($|D_{S^*}| > \alpha N$):** By definition, the number of nodess is $N = (2^{n+1} - 1)$, so $\alpha N = \alpha(2^{n+1} - 1)$.
By Lemma 5, $D_{S^*}$ contains $\frac{|D_{S^*}|+|S^*|}{2}$ leaves. Substituting this into $|D_{S^*}| > \alpha N$, and looking at the leaves $\alpha N = \alpha 2^{n+1} - \alpha \frac{|D_S|+|S*|}{2} > \frac{\alpha(2^{n+1}+|S*|)}{2} > \frac{\alpha 2^{n+1}}{2} = \alpha 2^n$ we get the number of leaves $|\{0,1\}^n \cap D_{S^*}| = \frac{|D_{S^*}|+|S^*|}{2} > \alpha 2^n$.

So the case is not within soundness gap.

So the genera idea, $\mathcal{V}$ will reject $\tau$ (the answer to the challenge $\gamma_i$, $\gamma = (\gamma_1, \ldots, \gamma_t)$, if there exists a node $u \in S$ that's on the path from $\gamma_i$ to the root, i.e.

V rejects if $\gamma \cap D_{S^*} = \gamma \cap \hat{S}^* = \gamma \cap \hat{S} \neq \emptyset$.

From the previous inequality on the number of leaves and assuming that all $\gamma_i$'s are sampled uniformly, we get $\Pr[\gamma_i \notin D_{S^*}] = 1 - |\{0,1\}^n \cap D_{S^*}|/2^n < 1 - \alpha$.

(probability of challenged nodes arent in the path of inconsistent answers$\hat{)}$
And since all $\gamma_i$'s are sampled independently, $\Pr[\gamma \cap D_{S^*} = \emptyset] = \prod_{i=1}^{t} \Pr[\gamma_i \notin D_{S^*}] < (1-\alpha)^t$.

Combined all together, a cheating prover $\widetilde{\mathcal{P}}$ will have its proof rejected with probability $1 - (1-\alpha)^t - \frac{2 \cdot n \cdot w \cdot q^2}{2^w}$. 1-(inconsistent leaves)

## 0.4 Proof of Efficiency

### 0.4.1 proof size

$w$ bits specify a label and $n$ bits specify a node. Thus, the exchanged messages and their lengths are as follows:

- $|\chi| = w$. $\chi$ is the initial statement that is a uniformly random $w$-bit string. It is initially communicated from Verifier $\to$ Prover.

- $|\phi| = w$. $\phi$ and $\phi_p$ are proofs computed from $\mathsf{PoSW}$. $\phi$ is the root label sent from Prover $\rightarrow$ Verifier. It is a $w$-bit string, since labels are calculated using $\mathsf{H} : \{0,1\}^{\leq w(n+1)} \rightarrow \{0,1\}^w$.
- $|\gamma| = t \cdot n$. $\gamma = (\gamma_1, ...\gamma_t)$ is a challenge sent from Verifier $\rightarrow$ Prover. It consists of $t$ leaf nodes of $n$ bits.
- $|\tau| \leq t \cdot w \cdot n$. $\tau := \mathsf{open}(\chi, N, \phi_p, \gamma)$ is the answer sent from Prover $\rightarrow$ Verifier, which answers the challenge $\gamma$. The answer will contain the $w$-bit label for each $n$-bit $\gamma_i$.

### 0.4.2 prover efficiency

The prover $\mathcal{P}$'s efficiency depends queries made while computing the $\mathsf{PoSW}$ and $\mathsf{open}$.

- $\mathsf{PoSW}^{\mathsf{H}_\chi}(N)$ is computed using $N$ sequential queries to $\mathsf{H}_\chi$. Each input has a length of at most $(n+1) \cdot w$ bits, by definition.
- $\mathsf{open}^{\mathsf{H}_\chi}(N, \phi, \gamma) = \tau$. $\mathsf{open}$ requires
  1. $(n+1)w$ bits to compute each label of the challenge,
  2. $2^{m+1}w$ labels to be stored in $\phi_p$, and
  3. $|\tau| \leq t \cdot w \cdot n$ bits to send back.

  Adding these, we need $(n+1+n \cdot t + 2^{m+1})w$ bits of memory. We examine the different cases depending on $m$, i.e. how many levels are used to store $\phi_p$:

  **Case** $m = n$ $\mathcal{P}$ stores all the labels computed by $\mathsf{PoSW}^{\mathsf{H}_\chi}(N)$, so no additional queries are needed

  **Case** $m = 0$, $\mathcal{P}$ does not store any label computed by $\mathsf{PoSW}^{\mathsf{H}_\chi}(N)$, and needs to recompute all $N$ queries

  **Case** $0 < m < n$ Since $\mathcal{P}$ stored the top $m$ levels, it needs to recalculate any query between level $n$ to $m$. This would require calculating the leaves starting from the $m - n^{th}$ level which would require $(2^{m-n+1} - 1) \cdot t$ queries, for $t$ challenges.

### 0.4.3 verifier efficiency

The verifier only needs to sample a random challenge of $|\gamma| = t \cdot n$, and computing $\mathsf{verify}(\chi, N, \phi, \gamma, \tau)$. $\mathsf{verify}$ makes $t \cdot n$ queries (for each $\gamma$) each of length $n \cdot w$ bits ($n$ leaf nodes' length and $w$ label lengths).
and we want to know if $A$ makes at most $q$ queries what's the probability of two queries to collide

# 1 previous papers

**MMV's paper defined** their proof of sequential work mainly in the context of time-lock puzzles

it's also possible to use modular exponentiation as a proof of sequential work in the context of CPU benchmarks and time-lock puzzles