Report on Simple Proofs of Sequential Work

Haikuo Yin and Sharon Zhen

March 11, 2019

1 Introduction

The main idea of this paper is a construction to prove sequential work. This was first done by Mahmoody, Moran and Vadhan at ITCS 2013 [MMV'13], and authors of the current paper–Cohen and Pietrzak [CP'18]–propose a new construction that's simpler and more efficient. This paper received the Best Paper Award at EuroCrypt 2018.

In this paper, proof of sequential work is roughly defined as a prover $\mathcal{P}$ proving to a verifier $\mathcal{V}$ that N sequential queries have been made to a Random Oracle H.

[MMV'13]'s construction is based on repeatedly calling H on the nodes of a directed acyclic graph (DAG) to generate labels for each node in a sequential manner. [CP'18] improves upon their work by changing the underlying DAG to both simplify the proof and achieve more efficient performance.

2 Fundamental Concepts

Here we will describe the fundamental concepts used to construct [CP'18].

2.1 Definition of Proof of Sequential Work (PoSW)

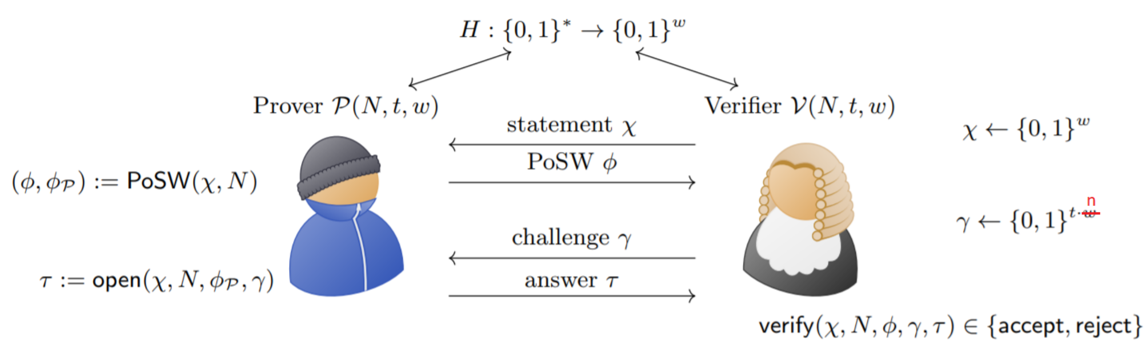

Figure 1: Illustration of PosW protocol given in the paper [1]

In this paper, PoSW is specified using a set of three algorithms, PoSW, open, and verify, roughly defined as:

- PoSW: $\mathcal{P}$ computes a proof and sends it to $\mathcal{V}$
- open: $\mathcal{P}$ receives a challenge from $\mathcal{V}$ and computes the answer
- verify: $\mathcal{V}$ receives answer from $\mathcal{P}$ and either accepts or rejects

Formally, the protocol is defined as:

1. **Common Inputs**: $\mathcal{P}$ and $\mathcal{V}$ receive as input statistical security parameters $w, t \in \mathbb{N}$, and a time parameter $N \in \mathbb{N}$. Both also has access to a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^w$.
2. **Statement**: $\mathcal{V}$ generates random string $\chi \leftarrow \{0,1\}^w$ and sends it to $\mathcal{P}$.
3. **Compute PoSW**: $\mathcal{P}$ computes a proof $(\phi, \phi_{\mathcal{P}}) := \mathsf{PoSW}^{\mathsf{H}}(\chi, N)$ (an honest $\mathcal{P}$ would make N sequential queries to H), keeping ϕ and sends $\phi_{\mathcal{P}}$ to $\mathcal{V}$.
4. **Generating Challenge**: $\mathcal{V}$ samples random challenge $\gamma \leftarrow \{0,1\}^{t \cdot w}$ (t strings of length w) and sends it to $\mathcal{P}$.
5. **Open**: $\mathcal{P}$ opens challenge γ and computes response $\tau := \mathsf{open}^{\mathsf{H}}(\chi, N, \phi_{\mathcal{P}}, \gamma)$, sending τ back to $\mathcal{V}$.
6. **Verify**: $\mathcal{V}$ verifies τ by computing $\mathsf{verify}^{\mathsf{H}}(\chi, N, \phi, \gamma, \tau) \in \{\mathsf{accept}, \mathsf{reject}\}$ and either accepts or rejects the proof.

The authors require:

- Perfect **Correctness**: $\mathcal{V}$ will accept honest $\mathcal{P}$ with probablility 1.
- **Soundness**: $\mathcal{V}$ will accept malicious prover $\widetilde{\mathcal{P}}$ with good probability ONLY IF $\widetilde{\mathcal{P}}$ has queried H "almost" N times.

2.2 Graph Definitions

Here we will define the graph properties used by [CP'18] in their construction.

Definition 1 (Graph Labelling): Given a DAG (directed acyclic graph) $G = (V, E)$, where the set of vertices $V = \{0, ..., N-1\}$, and a hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^w$, the label $\ell_i \in \{0,1\}^w$ for each vertex $i \in V$ is defined as $\ell_i = \mathsf{H}(i, \ell_{p_1}, \ldots, \ell_{p_d})$, where $(p_1, \ldots, p_d) = \mathsf{parents}(i)$. Parents of a vertex i are defined as any node with an edge to vertex i.

As defined here, the labels can be computed by making a query to H for each vertex in the DAG, in topological order, resulting in N sequential queries to H.

Definition 2 (Depth-Robust DAG): A DAG is e, d depth-robust if for any subset S of its vertices V (i.e. $S \subset V$), where $|S| \leq e$, the subgraph $V - S$ has a path of at least length d.

2.2.1 Graph Notations

Definition 3 ($(\hat{S}, S^*, D_S)$): Given a DAG $G = (V, E)$ and subset of vertices $S \subseteq V$, the authors defined $\hat{S}$ as the set of leaves under S i.e.

$$\hat{S} := \{v || u \in \{0,1\}^n \ : \ v \in S, u \in \{0,1\}^{n-|v|}\}$$

The authors defined S^* as the smallest set that share the same leaves of S, i.e.

$$S^* := \{S' \ : \ \hat{S'} = \hat{S} \ and \ |S'| \leq |K|, \forall \hat{K} = \hat{S}\}$$

The authors defined D_S as the nodes that are in S or below S, i.e.

$$D_S := \{v || v' \ : \ v \in S, v' \in \{0,1\}^{n-|v|}\}$$

2.3 Random Orable Properties (RO)

Salting the RO: In all three algorithms PoSW, open, and verify, random string χ is used to sample a RO H_χ. One example [CP'18] provided is adding χ as a prefix to every call to H, i.e. $\mathsf{H}_\chi(\cdot) \stackrel{\text{def}}{=} \mathsf{H}(\chi, \cdot)$.

Definition 3 (H-Sequence): An 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}.

3 Construction

3.1 Definition of Underlying DAG (G_n^{PoSW})

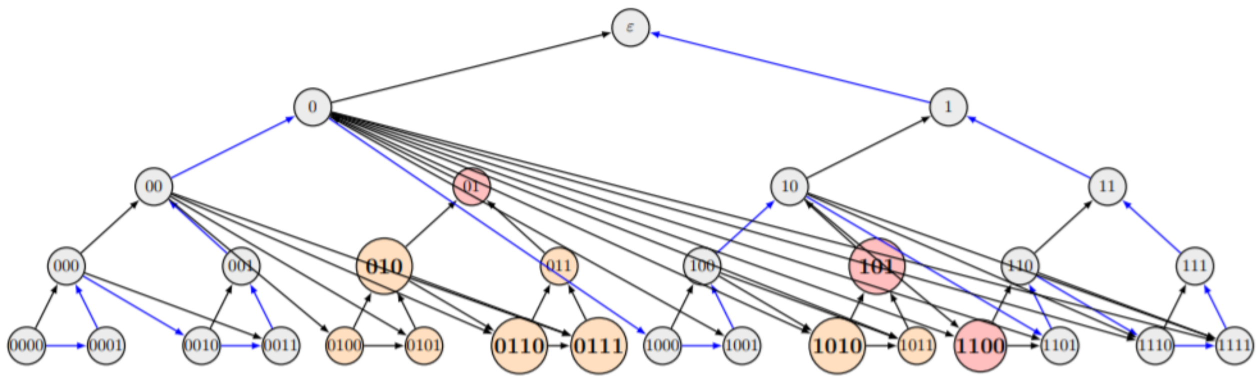

Figure 2: Example graph given in the paper [1]

First, the authors start with a complete binary tree of depth $n \in \mathbb{N}$, which they call $B_n = (V, E')$. This would mean that there are $N = 2^{n+1} - 1$ vertices in the graph, and each vertex is identified as a binary string where its length is equal to the depth of the vertex from the root. The root, at depth 0, is identified as the empty string ε, or $\{0,1\}^0$. More formally, the set of vertices V is equal to the set of strings $\{0,1\}^{\leq n}$.

The authors also define vertex v as *above* u if $u = v||a$ (where $||$ refers to concatenation) for some a. If v as *above* u, then u is *below* v.

All the edges in B_n go from the leaves to the root (which would mean that all the vertices below some vertex v are the ancestors of v. Formally, $E' = \{(x\|b, x) \ : \ b \in \{0,1\}, x \in \{0,1\}^i, i < n\}$.

Now the authors add edges to B_n to turn it into G_n^{PoSW} used for their construction. They add edges E'' which, for all leaves $u \in \{0,1\}^n$, are edges (v, u), where v is the left sibling of any vertex on the path from u to the root. Formally, $E'' = \{(v, u) \ : \ u \in \{0,1\}^n, u = a\|1\|a', v = a\|0\}$.

Thus formally, $G_n^{\text{PoSW}} = (V, E)$ where V are the vertices from B_n and $E = E' \cup E''$. This G_n^{PoSW} is what's used by the authors for PoSW.

3.2 Definition of PoSW

3.2.1 Parameters

To define these algorithms, there are 4 input parameters:

1. N: What the authors call the time parameter, which is the number of nodes in the DAG. $N = 2^{n+1} - 1$ for some depth $n \in \mathbb{N}$.
2. $\mathsf{H} : \{0,1\}^{\leq w(n+1)} \to \{0,1\}^w$ A hash function that takes inputs of length $w(n+1)$ and outputs strings of length w. For this proof, they are modeled as random oracles.
3. t: A statistical security parameter
4. M: Amount of memory available to prover $\mathcal{P}$, which is assumed to be $M = (t + n \cdot t + 1 + 2^{m+1})w$, where m is an integer $0 \leq m \leq n$.

3.2.2 Definition of PoSW, open, and verify

- $(\phi, \phi_{\mathcal{P}}) := \mathsf{PoSW}^{\mathsf{H}_\chi}(N)$: Algorithm used by prover $\mathcal{P}$ to generate labels $\{\ell_i\}_{i\in\{0,1\}^{\le n}}$ for every vertex in the DAG G_n^{PosW} using H_χ. $\mathcal{P}$ stores the labels of the m highest layers ($\{\ell_i\}_{i\in\{0,1\}^{\le m}}$) as $\phi_{\mathcal{P}}$, and sends root label (ℓ_ε) to $\mathcal{V}$ as ϕ, the proof.

- $\tau := \mathsf{open}^{\mathsf{H}_\chi}(N, \phi_{\mathcal{P}}, \gamma)$: Upon receiving challenge $\gamma = (\gamma_1 \ldots, \gamma_t)$ from verifier $\mathcal{V}$, where each $\gamma_i \in \{0,1\}^n$ is a leaf node, prover $\mathcal{P}$ generates response τ that contains, for every γ_i, the label ℓ_{γ_i} and the labels of the siblings of every node on the path from γ_i to the root, i.e. $\{\ell_k\}_{k\in\mathcal{S}_{\gamma_i}}$ where $\mathcal{S}_{\gamma_i} \stackrel{\text{def}}{=} \{\gamma_i[1\ldots j-1]\|(1-\gamma_i[j])\}_{j=1\ldots n}$.

  Formally, τ is defined as $\tau \stackrel{\text{def}}{=} \{\ell_{\gamma_i}, \{\ell_k\}_{k\in\mathcal{S}_{\gamma_i}}\}_{i=1\ldots t}$.

- $\mathsf{verify}^{\mathsf{H}_\chi}(N, \phi, \gamma, \tau)$: Algorithm used by verifier $\mathcal{V}$ to check that labels are correctly computed. Since $\mathcal{S}\gamma_i$ contains all the parents of γ_i, $\mathcal{V}$ first checks that ℓ_{γ_i} is correctly computed from its parents, i.e. that $\ell_{\gamma_i} \stackrel{?}{=} \mathsf{H}_\chi(i, \ell_{p_1}, \ldots, \ell_{p_d})$ where $(p_1, \ldots, p_d) = \mathsf{parents}(\gamma_i)$.

  $\mathcal{V}$ will then use that information to recursively compute the labels of all the vertices from on the path to the root. More formally, for $i = n-1, n-2, ..., 0$, $\mathcal{V}$ will compute $\ell_{\gamma_i[0\ldots i]} := \mathsf{H}_\chi(\gamma_i[0\ldots i], \ell_{\gamma_i[0\ldots i]\|0}, \ell_{\gamma_i[0\ldots i]\|1})$.

  Finally, $\mathcal{V}$ will verify that the computed label of the root equals the ϕ received from $\mathcal{P}$ earlier, i.e. that $\ell_{\gamma_i[0\ldots 0]} = \ell_\varepsilon$.

4 Proofs

4.1 Lemmas

Lemma 1: **Random Oracles are Collision Resistant**
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 $x \ne x', \mathsf{H}(x) = \mathsf{H}(x')$ is at most $\frac{q^2}{2^{w+1}}$

Proof. For individual queries $x_1 \le x_i \le x_q$, 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 $\sum_{i=1}^{q} \frac{i-1}{2^w} = \frac{q^2}{2^{w+1}}$. □

Lemma 2: **Random Oracles are Sequential**
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 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 $x_i \ne x_j, \mathsf{H}(x_i) = \mathsf{H}(x_j)$.

- Case 1: for some $0 \le i < s$, $\mathsf{H}(x_i)$ is a substring of x_{i+1}, but A did not query x_i. Since 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 $1 \le i \le j \le s-1$ 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^{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^{PoSW}.

1. First, separate G_n^{PoSW} into Right and Left subtrees. Each subtree is isomorphic to G_{n-1}^{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}^{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}^{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^{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^{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^{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^{PoSW}. G_0^{PoSW} is obviously true since it contains a single node. Suppose the lemma holds for G_i^{PoSW}. We now want to show it holds for G_{i+1}^{PoSW}. So pick some $G_{i+1}^{\mathsf{PoSW}} = (V, E)$. G_{i+1}^{PoSW} has a Left and Right subgraph, and root ϵ. The Left and Right subgraphs are isomorphic to G_i^{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^{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^{PoSW}) and find a directed path ending in node 0. Take the Right subgraph (equivalent to G_i^{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

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

□

4.2 Proof of Security

Theorem 1: Consider a PoSW defined using parameters N, H, t, and M as defined above, with an additional parameter $\alpha > 0$. α is what the authors call a "soundness gap", which is the percentage difference between N and how many queries to 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 $1-(1-\alpha)^t - \frac{2\cdot n\cdot w\cdot q^2}{2^w}$.

Proof: First let us consider $\frac{2\cdot n\cdot w\cdot q^2}{2^w}$, which is the probability that $\widetilde{\mathcal{P}}$ will find a collision in H (Lemma 1) and $\widetilde{\mathcal{P}}$ will find an H_χ sequence of length s making less than s queries to H_χ (Lemma 2). Note: $|x\_i| = w$ because x is the output of the H function. Since the maximum input length of the H is $w(n+1)$, Q, the total number of bits queried in q queries, is at most $q(n+1)w$. Since we assume that H is queried for $s-1$ rounds with arbitrarily many queries each round (Lemma 2), we can say that $s+1 \leq q$, the total number of queries.

$$\begin{aligned}
\frac{q^2}{2^{w+1}} + q\frac{Q+\sum_i^s |x_i|}{2^w} \quad &\leq q\frac{q\cdot w\cdot(n+1)+qw}{2^w} + \frac{q^2/2}{2^w} \\
&= \frac{q^2 w(n+1)+q^2 w+q^2/2}{2^w} \\
&= \frac{q^2(w(n+1)+w+1/2)}{2^w} \\
&< \frac{q^2(2wn)}{2^w}
\end{aligned}$$

Now that we have accounted for the probability that $\widetilde{\mathcal{P}}$ will break the sequentiality of H, let us consider the probability that $\mathcal{V}$ detects an inconsistent vertex (an inconsistent vertex being defined as a vertex with an incorrect label).

Let set $S \subseteq V = \{0,1\}^{\leq n}$ be the set of inconsistent vertices, and by Lemma 4 there is a path going through all the vertices of $V - D_{S^*}$, which are all consistent, so the path is an H_χ-sequence of length $N - |D_{S^*}|$. Now we can divide this into two cases:

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

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

$\mathcal{V}$ will reject if any γ_i of the t challenges $\gamma = (\gamma_1, \ldots, \gamma_t)$ it gives $\mathcal{P}$ has a vertex in S on the path from γ_i to the root, or $\gamma \cap D_{S^*} = \gamma \cap \hat{S}^* = \gamma \cap \hat{S} \neq \emptyset$.

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

And since all γ_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 $1-(1-\alpha)^t - \frac{2\cdot n\cdot w\cdot q^2}{2^w}$.

4.3 Proof of Efficiency

4.3.1 proof size

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

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

- $|\phi| = w$. ϕ and ϕ_p are proofs computed from PoSW. ϕ 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 γ. The answer will contain the w-bit label for each n-bit γ_i.

4.3.2 prover efficiency

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

- $\mathsf{PoSW}^{\mathsf{H}_\chi}(N)$ is computed using N sequential queries to H_χ. Each input has a length of at most $(n+1) \cdot w$ bits, by definition.

- $\mathsf{open}^{\mathsf{H}_\chi}(N, \phi, \gamma) = \tau$. open requires

  1. $(n+1)w$ bits to compute each label of the challenge,
  2. $2^{m+1}w$ labels to be stored in ϕ_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 ϕ_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 $n - m^{th}$ level which would require $(2^{n-m+1} - 1) \cdot t$ queries, for t challenges.

4.3.3 verifier efficiency

The verifier only needs to sample a random challenge of $|\gamma| = t \cdot n$, and computing $\mathsf{verify}(\chi, N, \phi, \gamma, \tau)$. verify makes $t \cdot n$ queries (for each γ) each of length $n \cdot w$ bits (n leaf nodes' length and w label lengths).

References

1. Cohen B., Pietrzak K. (2018) Simple Proofs of Sequential Work. In: Nielsen J., Rijmen V. (eds) Advances in Cryptology - EUROCRYPT 2018. EUROCRYPT 2018. Lecture Notes in Computer Science, vol 10821. Springer, Cham