On the Cryptographic Hardness of Local Search

Authors: Nir Bitansky, Idan Gerichter

Presenters: Shouqiao Wang[1], Yuriko Nishijima[2]

April, 2024

[1]Columbia University, New York. shwang27@gsb.columbia.edu

[2]Columbia University, New York. yn2411@columbia.edu

Recap: PLS

Definition (SINK-OF-DAG)

Given $V = \{0,1\}^n$, $S : V \to V$, cost $C : V \to \{0,1\}^m$. The edge $e = (u, v)$ exists $\iff$ $S(u) = v$, $C(v) > C(u)$.
Problem: Given a source s':

$$S(s') \neq s' \text{ and } C(S(s')) > C(s'),$$

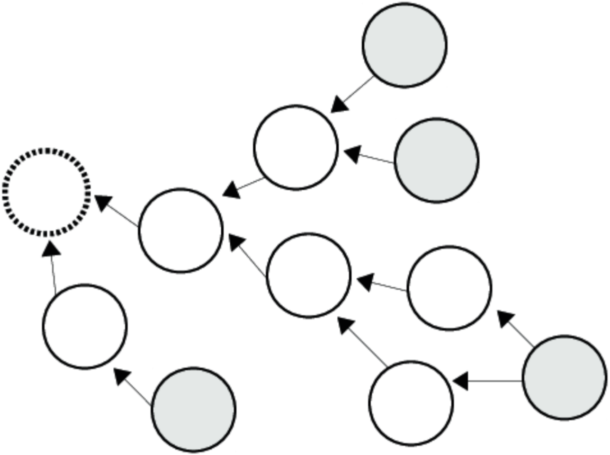
find a sink u with no out-edge, i.e.,

$$S(u) = u \text{ or } S(u) = v \text{ but } C(v) \leq C(u).$$

SINK-OF-DAG is PLS complete!

Recap: PLS

Each node can have multiple in-nodes, but only one out-node.



Recap: SVL

Definition (SVL: SINK-OF-VERIFIABLE-LINE)

Given a DAG on $U = \{0,1\}^n$ implicitly defined by $S : U \to U$, we also consider the promise $V : U \times [T] \to \{0,1\}$ given as

$$V(w, i) = 1 \iff w = S^{i-1}(x_s).$$

Problem: Given a x_s, find a w s.t. $V(w, T) = 1$.

From previous lecture, we have

- iO + OWFs $\Rightarrow$ SVL is hard
- SVL is hard $\Rightarrow$ PLS $\cap$ PPAD is hard

Recap: SVL

Why iO + OWFs $\Rightarrow$ SVL hardness?

Recap: SVL

Why iO + OWFs $\Rightarrow$ SVL hardness?

Main proof idea is:
A polynomial-space machine M solving hard search problem R on input x, with an incrementally-verifiable computation, given the proof is generated uniquely

$$(M_x^0, \pi_0) \xrightarrow{S} \cdots \xrightarrow{S} (M_x^T, \pi_T).$$

Recap: SVL

Why iO + OWFs $\Rightarrow$ SVL hardness?

Main proof idea is:
A polynomial-space machine M solving hard search problem R on input x, with an incrementally-verifiable computation, given the proof is generated uniquely

$$(M_x^0, \pi_0) \xrightarrow{S} \cdots \xrightarrow{S} (M_x^T, \pi_T).$$

SVL is not a total problem (it's a promise problem)
We can construct it in a way that it always has a solution

- PSPACE machine
- IVC (Incrementally-verifiable computation) + **Uniqueness**

$\Rightarrow$ Three properties:
verifiable, incrementally generateable proofs, **unique**

Main Question

Question:
What can we get if we relax the uniqueness property?

Verifiable + incrementally generateable proofs + unique
$\Rightarrow$ SVL hardness $\Rightarrow$ PLS $\cap$ PPAD hardness.

Main Question

Question:
What can we get if we relax the uniqueness property?

Verifiable + incrementally generateable proofs + unique
$\Rightarrow$ SVL hardness $\Rightarrow$ PLS $\cap$ PPAD hardness.

Verifiable + incrementally generateable proofs + computational unique (hard to find more than one proof)
$\Rightarrow$ rSVL hardness $\Rightarrow$ PLS $\cap$ PPAD hardness.

Main Question

Question:
What can we get if we relax the uniqueness property?

Verifiable $+$ incrementally generateable proofs $+$ unique
$\Rightarrow$ SVL hardness $\Rightarrow$ PLS $\cap$ PPAD hardness.

Verifiable $+$ incrementally generateable proofs $+$
computational unique (hard to find more than one proof)
$\Rightarrow$ rSVL hardness $\Rightarrow$ PLS $\cap$ PPAD hardness.

Verifiable $+$ incrementally generateable proofs
$\Rightarrow$ We cannot construct SVL instances, but **PLS hardness**!

Main Result

IVC with incremental completeness $\Rightarrow$ PLS hardness.

Definition (IVC with incremental completeness)

IVC=(IVC.G, IVC.P, IVC.V) for Turing machine M:

- IVC.G(x): Outputs public parameters pp (randomized)
- IVC.P(pp, t, C, π): Outputs a proof π' (deterministic)
- IVC.V(pp, t, C, π): Outputs ACC or REJ (deterministic)

with properties:

- Incremental completeness: Given π' =IVC.P(pp, t, C, π), IVC.V$(pp, t, M_x^t, \pi) = ACC$ $\Rightarrow$ IVC.V$(pp, t+1, M_x^{t+1}, \pi') = ACC$
- Soundness
- Efficiency

Intuitively, why IVC+IC $\Rightarrow$ PLS?

In PLS,

- one node can have multiple in-nodes
- but only one out-node
- all nodes must be ordered in some increasing way

In IVC with incremental completeness,

- multiple accepting proofs $\Rightarrow$ multiple in-nodes
- given $\forall(M_t, \pi_t)$, the incremental-proof procedure gives only one specific π' for M_{t+1}

IVC Reduction Theorem

Theorem

Let $R \in$ FNP, solvable by a polynomial-space Turing machine M. If there exists an IVC scheme with incremental completeness for M, there exists a computationally sound Karp reduction (χ, W) from R to LS (or SINK-OF-DAG).

Definition 4.2 (Computational Karp Reduction). *For $R, R' \in \mathsf{FNP}$, a computational Karp reduction from R to R' consists of a pair $(\mathcal{X}, \mathcal{W})$, where $\mathcal{X}(x)$ is a randomized efficient algorithm and $\mathcal{W}(w)$ is a deterministic efficient algorithm. We make the following requirement:*

- ***Computational Soundness:*** *For every efficient adversary $\mathcal{A}$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^\lambda$ for which R_x is non-empty:*

$$\Pr\left[\begin{array}{c} w' \in R'_{x'} \\ w \notin R_x \end{array} \middle| \begin{array}{c} x' \leftarrow \mathcal{X}(x) \\ w' \leftarrow \mathcal{A}(x') \\ w = \mathcal{W}(w') \end{array}\right] \leq \mu(\lambda) \ ,$$

where R_x and $R'_{x'}$ are the set of witnesses for x in R and x' in R' respectively.

Construct IVC with IC

Question:
Without uniqueness requirement for proofs, can we construct IVC with incremental completeness from better (less) assumptions than what we used previously for rSVL hardness (like SNARGs and so on)?

[3] CP stands for Cohen and Pietrzak, who introduced CP Graphs

Construct IVC with IC

Question:
Without uniqueness requirement for proofs, can we construct IVC with incremental completeness from better (less) assumptions than what we used previously for rSVL hardness (like SNARGs and so on)?

Answer:
Yes!

We only need the assumption of ROM (Random Oracle Model)

PSPACE-language-with-incrementable-non-unique-proofs exists in the ROM!

We can construct it using some techniques in blockchain, CP Graphs[3] and Proofs of Sequential Work (PoSW).

[3] CP stands for Cohen and Pietrzak, who introduced CP Graphs

PoSW

Definition (PoSW)

In a PoSW, the prover is given a statement χ and a time parameter T, and can generate a corresponding proof π by making T sequential steps.
The soundness requirement is that provers that make $\ll T$ sequential steps, will fail to generate a valid proof for χ.

PoSW

Definition (PoSW)

In a PoSW, the prover is given a statement χ and a time parameter T, and can generate a corresponding proof π by making T sequential steps.
The soundness requirement is that provers that make $\ll T$ sequential steps, will fail to generate a valid proof for χ.

Question: How to construct a PoSW?

PoSW

Definition (PoSW)

In a PoSW, the prover is given a statement χ and a time parameter T, and can generate a corresponding proof π by making T sequential steps.
The soundness requirement is that provers that make $\ll T$ sequential steps, will fail to generate a valid proof for χ.

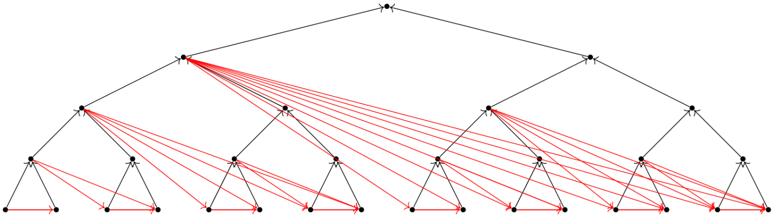
Question: How to construct a PoSW?

Answer: Use CP Graphs!

CP Graphs

Definition (CP Graphs)

CP Graphs is a complete binary tree with edges pointing from the leaves to the root with some added edges (red lines).



The added edges are the edge pointing from one node to all the leaves under its right sibling node.

Graph Labeling in CP Graphs

Labeling Principle:

In order to compute the label of one node, we must first compute the labels of its all in-nodes. Specifically, the label of any given node is obtained by applying a random oracle $H_\chi = H(\chi, \cdot)$ to the labels of its incoming nodes.

Graph Labeling in CP Graphs

Labeling Principle:

In order to compute the label of one node, we must first compute the labels of its all in-nodes. Specifically, the label of any given node is obtained by applying a random oracle $H_\chi = H(\chi, \cdot)$ to the labels of its incoming nodes.

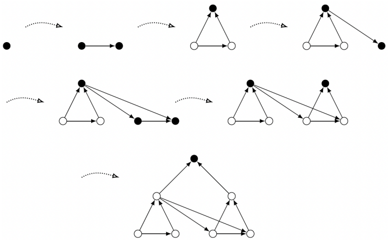
To compute the root, labeling must be done **sequentially**!

Graph Labeling in CP Graphs

Labeling Principle:
In order to compute the label of one node, we must first compute the labels of its all in-nodes. Specifically, the label of any given node is obtained by applying a random oracle $H_\chi = H(\chi, \cdot)$ to the labels of its incoming nodes.

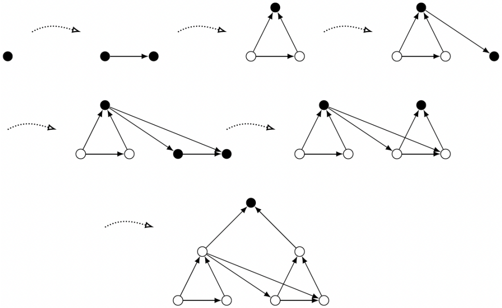
To compute the root, labeling must be done **sequentially**!

CP Labeling Procedure:



Properties of CP Labeling

CP Labeling Procedure:



Suppose $v_1, v_2, \cdots v_T$ is the sequence in which labels are being outputted, U_t is the set of nodes we need to store at round t.

Properties:

- Computing root needs exponentially many steps, but only polynomial space.
- Given t, computing U_t v_t can be done in poly(d) time.

How to Construct Proofs?

How to turn this into an actual proof of sequential work π?

How to Construct Proofs?

How to turn this into an actual proof of sequential work π?

A high-level design

1. Prover publishes the label of the root of the entire tree
2. Verifier responds with random challenge leaves
3. Prover answers by providing all the labels in the corresponding paths toward the root (Merkle proof)
4. Make it non-interactive by Fiat-Shamir

How to Construct Proofs?

How to turn this into an actual proof of sequential work π?

A high-level design

1. Prover publishes the label of the root of the entire tree
2. Verifier responds with random challenge leaves
3. Prover answers by providing all the labels in the corresponding paths toward the root (Merkle proof)
4. Make it non-interactive by Fiat-Shamir

Then, what do we have for now?

How to Construct Proofs?

How to turn this into an actual proof of sequential work π?

A high-level design

1. Prover publishes the label of the root of the entire tree
2. Verifier responds with random challenge leaves
3. Prover answers by providing all the labels in the corresponding paths toward the root (Merkle proof)
4. Make it non-interactive by Fiat-Shamir

Then, what do we have for now?

We can get an incrementally generateable nodes (think of M_x^t), by keeping track of U_t.

How to Construct Proofs?

How to turn this into an actual proof of sequential work π?

A high-level design

1. Prover publishes the label of the root of the entire tree
2. Verifier responds with random challenge leaves
3. Prover answers by providing all the labels in the corresponding paths toward the root (Merkle proof)
4. Make it non-interactive by Fiat-Shamir

Then, what do we have for now?

We can get an incrementally generateable nodes (think of M_x^t), by keeping track of U_t.

However, we still need to construct incrementally generateable proofs to guarantee the verifiability of the intermediate states!

Naive Idea

Question: Given a sequence of set $\{U_t\}$, how to contruct π_t?
A high-level design

1. Prover publishes the label of the root of the entire tree
2. Verifier responds with random challenge leaves
3. Prover answers by providing all the labels in the corresponding paths toward the root (Merkle proof)
4. Make it non-interactive by Fiat-Shamir

Naive Idea

Question: Given a sequence of set $\{U_t\}$, how to contruct π_t?
A high-level design

1. Prover publishes the label of the root of the entire tree
2. Verifier responds with random challenge leaves
3. Prover answers by providing all the labels in the corresponding paths toward the root (Merkle proof)
4. Make it non-interactive by Fiat-Shamir

Naive idea:

- Use Fiat-Shamir to get random challenges in step 2
- $\pi_t = \{\pi_t^i | \text{for } \forall u_i \in U_t, \pi_t^i \text{ is the proof for } u_i\}$

Not enough for IVC, since the intermediate proofs themselves cannot be computed **incrementally**, i.e., we cannot get π_{t+1} from (U_t, π_t), because we randomly select challenge leaves!

DLM[4] Construction

Rather than sampling fresh challenges for each round, we sample them randomly from previously computed challenges!

[4] DLM stands for Döttling, Lai, and Malavolta, who introduced DLM

DLM[4] Construction

Rather than sampling fresh challenges for each round, we sample them randomly from previously computed challenges!

Input:

1. String $\chi \in \{0,1\}^\lambda$.
2. Time $t \in [T]$.
3. Candidate proof $\pi = (\mathcal{L}, (u_i, S_{u_i}, \mathcal{P}_{u_i})_{i=1}^m)$.

Algorithm:

1. If $\{u_1, \ldots, u_m\} \neq \mathcal{U}_{t-1}$ output ε.
2. Set $\mathcal{L}[v_t] = \mathsf{H}_\chi(v_t, \mathcal{L}[p_1], ..., \mathcal{L}[p_r])$ where $(p_1, ..., p_r) = \mathsf{parents}(v_t)$ in lexicographic order.
3. If v_t is a leaf, let $S_{v_t} = \{v_t\}$ and $\mathcal{P}_{v_t} = \{(v_t)\}$.
4. Otherwise, v_t has two parents, $\ell, r \in \mathcal{U}_{t-1}$.
   (a) Sample a random subset S_{v_t} of size $\min(s, |\mathsf{leaves}(v_t)|)$ from $S_\ell \cup S_r$ using rand $\leftarrow \mathsf{H}'_\chi(v_t, \mathcal{L}[v_t])$ as random coins.
   (b) Let $\mathcal{P}_{v_t} = \emptyset$. For every $\mathsf{Path} \in \mathcal{P}_\ell \cup \mathcal{P}_r$ starting from a leaf in S_{v_t}, extend with v_t and append to $\mathcal{P}_{v_t}$.
5. Remove labels from $\mathcal{L}$ for all nodes but $\bigcup_{u \in \mathcal{U}_t} A_{\mathcal{P}_u}$.
6. Output $\pi' = (\mathcal{L}, (u, S_u, \mathcal{P}_u)_{u \in \mathcal{U}_t})$.

[4] DLM stands for Döttling, Lai, and Malavolta, who introduced DLM

ROM and IVC for DLM

Wait, where do we use ROM?

ROM and IVC for DLM

Wait, where do we use ROM?

We use random oracle machine H_χ in step 2 to label $\mathcal{L}[v_t]$ and H'_χ to sample random challenges.
H_χ and H'_χ could be different random oracle machines.

ROM and IVC for DLM

Wait, where do we use ROM?

We use random oracle machine H_χ in step 2 to label $\mathcal{L}[v_t]$ and H'_χ to sample random challenges.
H_χ and H'_χ could be different random oracle machines.

Why non-uniqueness?

ROM and IVC for DLM

Wait, where do we use ROM?

We use random oracle machine H_χ in step 2 to label $\mathcal{L}[v_t]$ and H'_χ to sample random challenges.
H_χ and H'_χ could be different random oracle machines.

Why non-uniqueness?

This is because the proofs for every randomly selected challenges are acceptable, even in the case of the naive idea rather than DLM.

ROM and IVC for DLM

Wait, where do we use ROM?

We use random oracle machine H_χ in step 2 to label $\mathcal{L}[v_t]$ and H'_χ to sample random challenges.
H_χ and H'_χ could be different random oracle machines.

Why non-uniqueness?

This is because the proofs for every randomly selected challenges are acceptable, even in the case of the naive idea rather than DLM.

Why IVC with incremental completeness?

ROM and IVC for DLM

Wait, where do we use ROM?

We use random oracle machine H_χ in step 2 to label $\mathcal{L}[v_t]$ and H'_χ to sample random challenges.
H_χ and H'_χ could be different random oracle machines.

Why non-uniqueness?

This is because the proofs for every randomly selected challenges are acceptable, even in the case of the naive idea rather than DLM.

Why IVC with incremental completeness?

The paper proves the IVC with incremental completeness, soundness and efficiency.

Main Takeaways

1. PSCPACE machine + IVC + **uniqueness**
$\Rightarrow$ PLS $\cap$ PPAD hardness

$\Big\downarrow$ get rid of the uniqueness property

PSCPACE machine + IVC $\Rightarrow$ PLS hardness

Main Takeaways

1. PSCPACE machine + IVC + **uniqueness**
   $\Rightarrow$ PLS $\cap$ PPAD hardness

   $\downarrow$ get rid of the uniqueness property

   PSCPACE machine + IVC $\Rightarrow$ PLS hardness

2. How to construct IVC with incremental completeness with less assumption?

   By CP Graphs and Proof of Sequential Work!

   - Exponential steps to compute the root, but only requires polynomial space
   - Incrementally generateable proofs by DLM.