On the Cryptographic Hardness of Finding a Nash Equilibrium

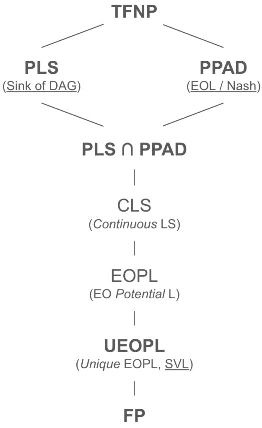
Nir Bitansky, Omer Paneth & Alon Rosen

Presented by Mark Chen & Yizhi Huang

2024-03-07

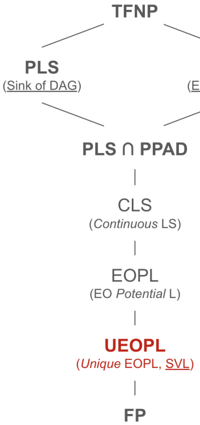
What Dan Showed Last Time

SVL, UEOPL, Pebbling Game, SVL $\subseteq$ PLS $\cap$ PPAD



What Dan Showed Last Time

SVL, UEOPL, Pebbling Game, SVL $\subseteq$ PLS $\cap$ PPAD



Definition (SVL : (S, V, x_s, T))

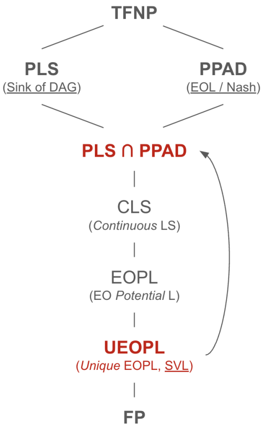
Given a **DAG** on $U = \{0,1\}^n$ implicitly defined by $S : U \to U$, we also consider the promise $V : U \times [T] \to \{0,1\}$ given as

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

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

What Dan Showed Last Time

SVL, UEOPL, Pebbling Game, SVL $\subseteq$ PLS $\cap$ PPAD



Lemma (SVL hardness* $\implies$ hardness in PPAD and PLS)

Recall pebbling game, **PG***:*

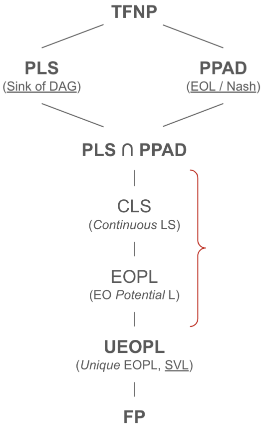
$$\mathbf{PG} : (S, V, x_s, T) \to (S', P', \widetilde{x_s}, C)$$

(Recall how **PG** *works as well as this reduction from Dan's lecture last time). The point is, $(S', P', \widetilde{x_s})$ gives an instance of* **SVL** *(***PPAD***-complete) and $(S', C, \widetilde{x_s})$ gives an instance of* **DAG on SVL** *(***PLS***-complete).*

* In this talk, "hardness" means "hard on average" – i.e. $\exists$ an efficient sampler of hard instances.

What Dan Showed Last Time

SVL, UEOPL, Pebbling Game, SVL $\subseteq$ PLS $\cap$ PPAD



Remark (Why is **SVL** important?)

SVL *as a hard instance of* **UEOPL** *implies hard instances for both* **PLS** *and* **PPAD** *through the aforementioned reduction. But,* **UEOPL** *is actually much lower in the* **TFNP** *hierarchy!!*

Remark (Next Steps in the Journey)

1. **OWF** $+$ **VBB** $\implies$ **SVL** *is hard*
2. *"Super strong"* **iOWF** $+$ *"Super strong"* $i\mathcal{O}$ $\implies$ **SVL** *is hard*
3. *(Next Talk by Ashvin)* **OWP** $+ i\mathcal{O} \implies$ **SVL** *hard.*
4. *Other stuff* $\implies$ **SVL** *hard (after spring break)*

Agenda

Theorem (Stage ①; Idea of the Construction, Gate Way to **SVL**)

$\exists$**OWF** + **VBB** $\implies$ **SVL** *hard.*

Agenda

Theorem (Stage ①; Idea of the Construction, Gate Way to SVL)

$\exists$**OWF** + **VBB** $\implies$ **SVL** *hard.*

Theorem (Stage ②, BPR Main Theorem; Outdated Analysis)

$\exists$ *"super strong"* **iOWF** *and "super strong"* $i\mathcal{O}$ $\implies$ **SVL** *hard.*

- First "super strong" to mean **sub-exponentially-hard**
- Second "super strong" to mean **quasi-polynomially-hard**

Agenda

Theorem (Stage ①; Idea of the Construction, Gate Way to SVL)

$\exists$**OWF** + **VBB** $\implies$ **SVL** *hard.*

Theorem (Stage ②, BPR Main Theorem; Outdated Analysis)

$\exists$ *"super strong"* **iOWF** *and "super strong"* $i\mathcal{O}$ $\implies$ **SVL** *hard.*

- First "super strong" to mean **sub-exponentially-hard**
- Second "super strong" to mean **quasi-polynomially-hard**

Theorem (Stage ③, Ashvin's Talk Right After; Compare to ②)

(Next Talk by Ashvin) **OWP** + $i\mathcal{O}$ $\implies$ **SVL** *hard.*

Agenda

Theorem (Stage ①; Idea of the Construction, Gate Way to **SVL**)

$\exists$**OWF** + **VBB** $\implies$ **SVL** *hard.*

Why **OWF**?

Theorem (Recall)

OWF $\implies$ **PRG**

Why **OWF**?

Theorem (Recall)

OWF $\implies$ **PRG** $\implies$ **PRF**

Definition (Pseudo-Random Function (PRF), Informal)

PRF$_k(X)$ is **deterministically** and **efficiently** computable given k (the secret key), but someone without k **cannot efficiently** distinguish it from a **truly random function** [Goldreich-Goldwasser-Micali'86].

Definition (Pseudo-Random Function (PRF), Formal)

A function $f : \underbrace{\{0,1\}^n}_{X} \times \underbrace{\{0,1\}^s}_{k} \to \{0,1\}^m$ is $\underbrace{\text{a } (t, \epsilon, q)\text{-}\mathbf{PRF}}_{\text{default: poly } t, q, \text{ \& neg } \epsilon}$ if:

- Given k and X, $F_k(X)$ is efficiently computable.
- For any t-time oracle algorithm A making at most q queries,

$$\left| \Pr_{k \leftarrow \{0,1\}^s} \left[A^{f_k} = 1\right] - \Pr_{f \in \mathcal{F}} \left[A^{f} = 1\right] \right| < \epsilon$$

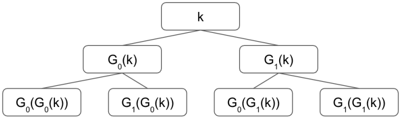
Why **OWF**?

Theorem (Recall)

OWF $\implies$ **PRG** $\implies$ **PRF**

Proof of the Second Implication.

Recall the Goldreich-Goldwasser-Micali (GGM) construction of **PRF** using **PRG**. Let $G : \{0,1\}^s \to \{0,1\}^n, n = 2s$ be a **PRG**. Then, we can define G_0 an G_1 to respectively be the left and right halves of G, s.t. $G = G_0 || G_1$.



(only two layers of a $(n+1)$-layer binary tree shown)

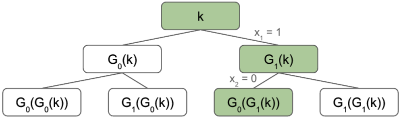
Why **OWF**?

Theorem (Recall)

OWF $\implies$ **PRG** $\implies$ **PRF**

Proof of the Second Implication.

Recall the Goldreich-Goldwasser-Micali (GGM) construction of **PRF** using **PRG**. Let $G : \{0,1\}^s \to \{0,1\}^n, n = 2s$ be a **PRG**. Then, we can define G_0 an G_1 to respectively be the left and right halves of G, s.t. $G = G_0 || G_1$.



(only two layers of a $(n+1)$-layer binary tree shown)

Define the **PRF** as

$$F_k(x_1x_2\dots x_n) = G_{x_n}(G_{x_{n-1}}(\dots(G_{x_1}(k))\dots))$$

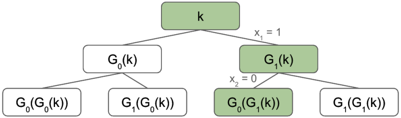
Why **OWF**?

Theorem (Recall)

OWF $\implies$ **PRG** $\implies$ **PRF**

Proof of the Second Implication.

Recall the Goldreich-Goldwasser-Micali (GGM) construction of **PRF** using **PRG**. Let $G : \{0,1\}^s \to \{0,1\}^n, n = 2s$ be a **PRG**. Then, we can define G_0 an G_1 to respectively be the left and right halves of G, s.t. $G = G_0 || G_1$.



(only two layers of a $(n+1)$-layer binary tree shown)

Define the **PRF** as

$$F_k(x_1 x_2 \ldots x_n) = G_{x_n}(G_{x_{n-1}}(\ldots(G_{x_1}(k))\ldots))$$

Then, hybrid argument follows. ∎

What is a Virtual Black Box (**VBB**)?

We First Introduce Program Obfuscator

Definition (Obfuscator, informal)

Obfuscator, not unlike a compiler, alters the look of a program such that the program becomes **unintelligible** (i.e. you won't know the program fully if it has been obfuscated) while keeping its **functionalities**.

In summary, general obfuscators require:

- **Functionality:** For any $C \in \mathcal{C}$,

$$\Pr_x[\mathcal{O}(C)(x) = C(x)] = 1.$$

- **Indistinguishability:** $O(C)$ should be unintelligible beyond just the input / output, serving practically as an oracle / blackbox (and this obfuscation should be done efficiently, with at most a polynomial blow-up).

What is a **VBB**?

Define Virtual Black Box

Definition (Virtual Black Box, informal)

An **ideal obfusctor** is so powerful that the obfuscated program would practically become a **(virtual) black box**, i.e. you would know nothing about it other than its input and output.

- **Functionality:** For any $C \in \mathcal{C}$,

$$\Pr_x[\mathcal{O}(C)(x) = C(x)] = 1.$$

- **Security:** For any PPT D, there exists a PPT S_D such that

$$\left|\Pr\left[D(\mathcal{O}(C)) = 1\right] - \Pr\left[S_D^C\left(1^\lambda\right) = 1\right]\right| \leq \varepsilon.$$

What is a **VBB**?

What We Already Know

Theorem ([Barak et al 2001]; VBB cannot exist for all circuits)

Constructive proof: https://en.wikipedia.org/wiki/Black-box_obfuscation

$*$ **VBB** is a very strong assumption. This theorem is to say that **VBB** may be too strong as an assumption.

What is a **VBB**?

What We Already Know

Theorem ([Barak et al 2001]; VBB cannot exist for all circuits)

Constructive proof: https://en.wikipedia.org/wiki/Black-box_obfuscation

∗ **VBB** is a very strong assumption. This theorem is to say that **VBB** may be too strong as an assumption. However, this is a general disproof, it is (probably) not known if **VBB** is constructible for the circuits used in the **SVL** construction.

So, reduction time!

What is a **VBB**?

What We Already Know

Theorem ([Barak et al 2001]; **VBB** cannot exist for all circuits)

Constructive proof: https://en.wikipedia.org/wiki/Black-box_obfuscation

$*$ **VBB** is a very strong assumption. This theorem is to say that **VBB** may be too strong as an assumption. However, this is a general disproof, it is (probably) not known if **VBB** is constructible for the circuits used in the **SVL** construction.

So, reduction time!
Well, a different kind of reduction...

Agenda

Theorem (Stage ①; Idea of the Construction, Gate Way to **SVL**)

$\exists$**OWF** + **VBB** $\implies$ **SVL** *hard.*

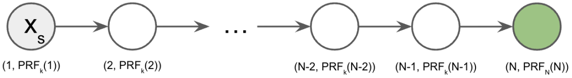
Reduce **VBB** to a hard instance of **SVL** assuming **PRF**

Definition (SVL)

Recall that **SVL** can be defined by a 4-tuple: (S, V, x_s, T) [we take $x_s = 0^n$ from now on].

Remark (When is SVL hard?)

A hard instance of **SVL** *is one with S (successor circuit) s.t. it is hard to fast forward. Particularly, finding σ_N should take $2^{\Omega(n)}$ steps, where $N = 2^n$ (exponentially-sized DAG).*



Reduce **VBB** to a hard instance of **SVL** assuming **PRF**

Construction of the **SVL** hard problem

We apply $f_k \in \mathbf{PRF}$, $\boxed{\sigma_i = f_k(i)}$, to define S_k, V_k:

$$\begin{aligned} S_k(i,\sigma) &= \begin{cases} \text{“sink” if } (i,\sigma) = (N, \sigma_N) \\ (i+1, \sigma_{i+1}) \text{ if } (i,\sigma) = (i, \sigma_i) \\ \perp \text{ o.w. [i.e. making it junk]} \end{cases} \\ V_k(i,(j,x)) &= \begin{cases} 1 \text{ if } i = j \text{ and } x = f_k(i) \\ 0 \text{ o.w.} \end{cases} \end{aligned} \tag{1}$$

Then, we obfuscate (1) to get

$$S = VBB(S_k); x_s = (1, f_k(1)), V = VBB(V_k)$$

So, we already have our **SVL** instance, $(S, VBB(V), x_s, N)$ ($T = N = 2^n$, and N was defined as the number of nodes in the DAG in the last slide).

Reduce **VBB** to a hard instance of **SVL** assuming **PRF**

Show the constructed **SVL** instance is indeed hard (security analysis)

Assume to the contrary that (the constructed **SVL** is not hard) $\exists A$ which is a PPT solver for our obfuscated instance $(S = VBB(S_k), V = VBB(V_k), x_s, N)$. Then, recalling security definition of **VBB**:

$$\left|\Pr\left[D(\mathcal{O}(C)) = 1\right] - \Pr\left[S_D^C\left(1^\lambda\right) = 1\right]\right| \leq \varepsilon,$$

we must have A' that solves the **SVL** instance (S_k, V_k, x_s, N) with only <u>oracle access</u> to S_k and V_k (non-neg $\pm$ neg $\implies$ non-neg). For example, we have

$$\left|\Pr\left[D(\mathcal{O}(S_k)) = 1\right] - \Pr\left[A_D'^{S_k}\left(1^\lambda\right) = 1\right]\right| \leq \varepsilon,$$

and analogously for V_k.

Next, the goal is to find a distinguisher D that necessarily breaks the security requirement of **PRF** using A'.

Reduce **VBB** to a hard instance of **SVL** assuming **PRF**

Show the constructed **SVL** instance is indeed hard (security analysis)

Since D can fully simulate A' (both are PPTs) on **SVL** which can in turn simulate S_k, V_k entirely by doing this (WLOG, say we want to simulate S_k): We let S'_k do the same thing as S_k, other than whenever S_k computes f_k, in which case we query the $f(i)$ oracle instead, where $f(i)$ is the function which we want to decide is truly random or a **PRF**.

Since A solves $VBB(S_k)$ and $VBB(V_k)$, we find the first instance where

- A hasn't queried $S(j-1, x)$ or $V(j-1, x)$.
- But A has a valid response for $S(j, y)$.

Finally, D decides that $f(i)$ is $\begin{cases} \text{a } \textbf{PRF}\text{, if } f_k(j) = y \\ \text{a truly random function, if } f_k(j) \neq y \end{cases}$

[$*$Note: Here, we need to argue with the fact that $f_k(j) = y$ for negligible probability when it's truly random, given how much bigger $\mathcal{F}$ is.]

Since N is greater than the runtime of A', there must be an $i > 1$ such that A' outputs or queries an oracle on $(i, PRF(i))$ but never queries $(i-1, PRF(i-1))$. This violates the security of **PRF**. (More formally, we can construct an adversary B^f for PRF that simulates A' up to the point that A' outputs or queries an oracle on (i, x), and decide whether f is a **PRF** according to whether $x = f(i)$.)

Agenda

Theorem (Stage ①; Idea of the Construction, Gate Way to SVL)
$\exists$**OWF** + **VBB** $\Longrightarrow$ **SVL** *hard.*

Theorem (Stage ②, BPR Main Theorem; Outdated Analysis)
$\exists$ *"super strong"* **iOWF** *and "super strong"* $i\mathcal{O}$ $\Longrightarrow$ **SVL** *hard.*

- First "super strong" to mean **sub-exponentially-hard**
- Second "super strong" to mean **quasi-polynomially-hard**

Theorem ([Barak et al 2001]; **VBB** cannot exist for all circuits)

Constructive proof: https://en.wikipedia.org/wiki/Black-box_obfuscation

$*$ **VBB** is a very strong assumption. This theorem is to say that **VBB** may be too strong as an assumption.

Indistinguishable Obfuscator ($i\mathcal{O}$)

Remark

Showing

$$\text{“super strong” } \mathbf{iOWF} + \text{“super strong” } i\mathcal{O} \implies \mathbf{SVL} \text{ hardness}$$

is **quite good***!*
There was a construction of $i\mathcal{O}$ in [Jain-Lin-Sahai’21,22] based on three “well-founded” assumptions. Plus, $i\mathcal{O}$ implies deniable encryption, functional encryption, multi-party key exchange, time-lock puzzles, trapdoor permutations, non-interactive ZK, verifiable computation, etc.

Indistinguishable Obfuscator ($i\mathcal{O}$)

Remark

Showing

$$\text{“super strong” } \mathbf{iOWF} + \text{“super strong” } i\mathcal{O} \implies \mathbf{SVL} \text{ hardness}$$

is **quite good***!*
There was a construction of $i\mathcal{O}$ in [Jain-Lin-Sahai'21,22] based on three "well-founded" assumptions. Plus, $i\mathcal{O}$ implies deniable encryption, functional encryption, multi-party key exchange, time-lock puzzles, trapdoor permutations, non-interactive ZK, verifiable computation, etc.

Remark

However, the BPR method used is quite evolved and it was soon superseded by a better result which Ashvin will present right away, so this presentation is leaving the whole proof in "Appendix A", and only focusing on relevant parts of the proof here.

Relevant Definition: Puncturable **PRF**

Recall the definition of **PRF**

Definition (Pseudo-Random Function (PRF), Formal)

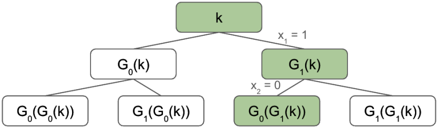
A function $f : \underbrace{\{0,1\}^n}_{X} \times \underbrace{\{0,1\}^s}_{k} \to \{0,1\}^m$ is a (t, ϵ, q)-**PRF** if:

- Given k and X, $F_k(X)$ is efficiently computable.
- For any t-time oracle algorithm A making at most q queries,

$$\left| \Pr_{k \leftarrow \{0,1\}^s} \left[A^{f_k} \right] - \Pr_{f \in \mathcal{F}} \left[A^{f} \right] \right| < \epsilon$$

Relevant Definition: Puncturable **PRF**

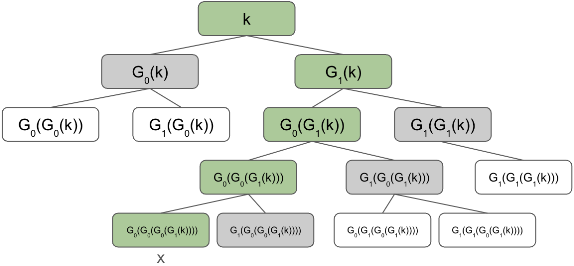
and the GGM construction of **PRF** from **PRG**



(only two layers of a $(n+1)$-layer binary tree shown)

Relevant Definition: Puncturable **PRF**

Then, a puncturable **PRF** is a **PRF** that can be evaluated everywhere but at x, which has the following GGM construction:



(only four layers of a $(n+1)$-layer binary tree shown)
*Notice how only n neighbors needed to specify

Relevant Lemma

$i\mathcal{O}$ of two circuits different for only one output is indistinguishable

Lemma (10)

Let $A(x)$ be a program, and $B_{r,z}(x) = \begin{cases} z \text{ if } x = r \\ A(x) \text{ otherwise} \end{cases}$. Then, for any random r and $\forall z$, $i\mathcal{O}(A) \approx i\mathcal{O}(B_{r,z})$ ["indistinguishable under $i\mathcal{O}$"].

Rest of the Proof

The rest of the proof for this stage uses the same construction as the first stage, but, since we need to obfuscate S_k, V_k using $i\mathcal{O}$ now, instead of **VBB**, we need to alter the security analysis, which is where puncturable **PRF** and lemma 11 come in, along with other things (we formulated puncturable **PRF** and lemma 11 because they are relevant later).

For details about the rest of the proof (where it's different to stage 1 proof), please see "Appendix A" (it's a hybrid argument).

For instance, "**super-polynomial**" actually comes from this analysis. The proof involves a "walk" between different hybrids, where in each hybrid another point is punctured, and there is a total of super polynomially many hybrids necessary for the proof to go through.

Agenda

Theorem (Stage ①; Idea of the Construction, Gate Way to SVL)

$\exists$**OWF** + **VBB** $\Longrightarrow$ **SVL** *hard.*

Theorem (Stage ②, BPR Main Theorem; Outdated Analysis)

$\exists$ *"super strong"* **iOWF** *and "super strong"* $i\mathcal{O}$ $\Longrightarrow$ **SVL** *hard.*

- First "super strong" to mean **sub-exponentially-hard**
- Second "super strong" to mean **quasi-polynomially-hard**

Theorem (Stage ③, Ashvin's Talk Right After; Compare to ②)

(Next Talk by Ashvin) **OWP** + $i\mathcal{O}$ $\Longrightarrow$ **SVL** *hard.*

Outro

This is it for what the presentation has to say about BPR. THANKS!

Special shout-outs to the teaching staff, Yizhi, Ashvin for their input, comments, intuitions.

Outro

This is it for what the presentation has to say about BPR. THANKS!

Special shout-outs to the teaching staff, Yizhi, Ashvin for their input, comments, intuitions.

Appendix A: iOWF and $i\mathcal{O} \implies$ hard instance of SVL

Idea

Reduce $i\mathcal{O}$ to a hard instance of **SVL**, *defined by* (S, V, x_s, T).

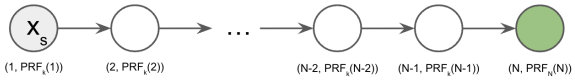
Appendix A: iOWF and $i\mathcal{O} \implies$ hard instance of SVL

Idea

Reduce $i\mathcal{O}$ to a hard instance of **SVL**, *defined by* (S, V, x_s, T).

Remark (When is **SVL** hard?)

A hard instance of **SVL** *is one with S (successor circuit) s.t. it is hard to fast forward. Particularly, finding σ_N should take $2^{\Omega(n)}$ steps, where $N = 2^n$ (exponentially-sized DAG).*



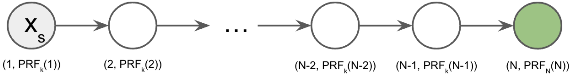
Appendix A: iOWF and $i\mathcal{O} \implies$ hard instance of SVL

Idea

Reduce $i\mathcal{O}$ to a hard instance of **SVL**, *defined by* (S, V, x_s, T).

Remark (When is **SVL** hard?)

A hard instance of **SVL** *is one with S (successor circuit) s.t. it is hard to fast forward. Particularly, finding σ_N should take $2^{\Omega(n)}$ steps, where $N = 2^n$ (exponentially-sized DAG).*



Idea

So, to reduce $i\mathcal{O}$ to a hard instance of **SVL** *is to construct S, V that make finding σ_N a $2^{\Omega(n)}$-time problem.*

Appendix A: Reduce $i\mathcal{O}$ to a hard instance of **SVL**

Definition (Recall, **PRF**)

$\mathbf{PRF}_k(X)$ is **deterministically** and **efficiently** computable given k (the secret key), but someone without k **cannot efficiently** distinguish it from a **truly random function** [Goldreich-Goldwasser-Micali'86].

Reduce $i\mathcal{O}$ to a hard instance of **SVL**

We apply $f_k \in$ **PRF**, $\boxed{\sigma_i = f_k(i)}$, to define S_k, V_k:

$$S_k(i,\sigma) = \begin{cases} \text{“sink” if } (i,\sigma) = (N,\sigma_N) \\ (i+1,\sigma_{i+1}) \text{ if } (i,\sigma) = (i,\sigma_i) \\ \perp \text{ o.w. [i.e. making it junk]} \end{cases} \tag{2}$$
$$V_k(i,(j,x)) = \begin{cases} 1 \text{ if } i = j \text{ and } x = f_k(i) \\ 0 \text{ o.w.} \end{cases}$$

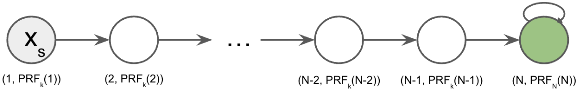
Then, we obfuscate (2) to get

$$S = i\mathcal{O}(S_k);\, V = i\mathcal{O}(V_k)$$

so that we won't have a way to know k for S, V to be efficiently computable and the only way to get to the end of the graph is by computing S super-polynomially many times.

Appendix A: Reduce $i\mathcal{O}$ to a hard instance of **SVL**

Now, suppose we have $S' = O(S'_k)$ and $V' = O(V'_k)$ that compute a similar graph, except that graph has a **self-loop at the end** instead of a **sink**:



Idea

If there is an efficient way to get to σ_N, we can simply find σ_N and check whether it is a self-loop or a sink, which will make the two graphs **not indistinguishable**. *That is, the only way to make it indistinguishable is for getting to σ_N to be hard. So, it suffices to show that the programs described by S, V and by S', V' are indistinguishable in order* **to show the desired hardness**.

Appendix A: Reduce $i\mathcal{O}$ to a hard instance of **SVL**

Idea

Show the programs described by S, V and by S', V' are **indistinguishable**.

Lemma (14)

Let $A(x)$ be a program, and $B_{r,z}(x) = \begin{cases} z \text{ if } x = r \\ A(x) \text{ otherwise} \end{cases}$. Then, for any random r and $\forall z$, $i\mathcal{O}(A) \approx i\mathcal{O}(B_{r,z})$ ["indistinguishable under $i\mathcal{O}$"].

Example

We can use it by planting r as a unique solution for a hard problem. We take a injective **PRG**, f, then $f(r)$ is a unique solution of r. Since the **PRG** is injective and an expanding when, when sampling from the image of the **PRG**, you will get something without no preimage, except for $\epsilon(\cdot)$ probability.

Appendix A: Reduce $i\mathcal{O}$ to a hard instance of **SVL**

First, we only show S and S', i.e. the way going forward:

- (Step 1): Pick a random edge and remove.
- (Step 2): Pick a random node w/ in-degree 0 and make it a self-loop.
- Repeat step 2 until we reach the end of the graph.

It has a runtime of $\boxed{t(\text{step 2}) \cdot O(N = 2^n)}$, assuming a sub-exponentially secure $i\mathcal{O}$. Note that this is called a "**hybrid argument**."

Corollary (Step 1 change is indistinguishable)

Direct result of lemma 11, since it is equivalent of changing

$$S_k(i, \sigma)$$

to

$$S'_{k,r}(i, \sigma) = \begin{cases} \perp & \text{if } i = r \\ S_k(i, \sigma) & \text{otherwise} \end{cases}$$

Appendix A: Reduce $i\mathcal{O}$ to a hard instance of SVL

Definition (puncturable **PRFs**)

Let n, k be polynomially bounded functions, then

$$\mathcal{PRF} = \left\{ \mathbf{PRF}_S : \{0,1\}^{n(|x|)} \to \{0,1\}^{|x|} \mid S \in \{0,1\}^{k(|x|)}, |x| \in \mathbb{N} \right\}$$

associated with an efficient key sampler $\mathcal{K}_{\mathcal{PRF}}$ is puncturable if $\exists$ a poly-time puncturing algorithm Punc that takes as in put a key S, and a point x^*, and output a punctured key $S\{x^*\}$, so that the following conditions are satisfied:

1. (Functionality preserved) For every $x^* \in \{0,1\}^{n(|x|)}$,

$$\Pr_{S \leftarrow \mathcal{K}_{\mathcal{PRF}}\left(1^{|x|}\right)} \left[\forall x \neq x^* : \mathbf{PRF}_S(x) = \mathbf{PRF}_{S\{x^*\}}(x) \mid S\{x^*\} = \mathrm{Punc}(S, x^*) \right] = 1$$

2. (Indistinguishability at punctured point) For any poly-size distinguisher $\mathcal{D}$, $\exists$ negligible $\epsilon(\cdot)$, s.t. $\forall |x| \in \mathbb{N}$, and $x^* \in \{0,1\}^{n(|x|)}$:

$$\left| \Pr[\mathcal{D}(x^*, S\{x^*\}, \mathbf{PRF}_S(x^*)) = 1] - \Pr[\mathcal{D}(x^*, S\{x^*\}, u) = 1] \right| \leq \epsilon(|x|)$$

Appendix A: Reduce $i\mathcal{O}$ to a hard instance of **SVL**

Corollary (19, Step 2 change is indistinguishable)

The choice of in-degree 0 *nodes is not truly random, as it can only be the ones that have already been made into a loop. Thus, we use* **puncturable pseudo-random functions** *for defining* $\sigma_i = f_k(i)$*, so that on some position* i *that has been punctured, it will still appear random even given the key* k*. Since now we have the true randomness after puncturing (*r *in place of* σ_i*, independent from rest of the program), we can just apply the lemma directly again similar to the previously corollary.*

Now that we showed S and S' are indistinguishable, we consider V and V': the hardness of computing V or V' is not affected by the end node being a sink or a self-loop, so the fact that key, k, is obfuscated gives the hardness for both, and they should agree everywhere given how we defined their respective graphs.

Appendix A: Reduce $i\mathcal{O}$ to a hard instance of **SVL**

Summary

In summary, we have used $i\mathcal{O}$ *to construct an instance of* **SVL**, (S, V, x_s, T) $(x_s, T$ *follow conveniently from the set-up, so we focused on constructing* $S, V)$. *We have shown that* S, V *are hard after applying sub-exponentially secure* $i\mathcal{O}$ *on* S_k, V_k *which assume the existence of* **PRF***s. In this case, it takes a super-polynomial runtime to solve this constructed instance of* **SVL**, *making it a hard instance of* **SVL**. *In other words,*

"super strong" **iOWF** *and "super strong"* $i\mathcal{O} \implies$ *a hard instance of* **SVL**