Finding a Nash Equilibrium Is No Easier Than Breaking Fiat-Shamir

Arka Rai Choudhuri, Pavel Hubacek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, Guy N. Rothblum

Presented by Jiaqian Li, Kashvi Gupta, adapted from Guy Rothblum's slides

March 21, 2024

Columbia University

Recap

Last lecture, we saw

Cryptographic hardness in $\mathsf{PPAD} \cap \mathsf{PLS}$:

- "supper strong" iOWF + "supper strong" iO $\Longrightarrow$ SVL hard
- OWP + iO $\Longrightarrow$ SVL hard

Last lecture, we saw

Cryptographic hardness in $\mathsf{PPAD} \cap \mathsf{PLS}$:

- "supper strong" iOWF + "supper strong" iO $\implies$ SVL hard
- OWP + iO $\implies$ SVL hard

In fact, the reduction in the latter work can also work with keyed iOWF (instead of OWP), and (as we will hopefully see later in the class) [BPW] showed iO+ OWF implies keyed iOWF, so overall we can get:

$$\text{OWF} + \text{iO} \implies \text{SVL hard}$$

However, the notion of iO still lies within the domain of speculation: many candidate schemes have been broken, and surviving ones are yet to undergo extensive evaluation.

The Sink-of-Verifiable-Line Problem

Definition (Sink-of-Verifiable-Line, SVL)

A Sink-of-Verifiable-Line instance (S, V, T, v_0) consists of

- $T \in \{1, 2, \ldots, 2^M\}$,
- $v_0 \in \{0, 1\}^M$,
- $S : \{0, 1\}^M \to \{0, 1\}^M$,
- $V : \{0, 1\}^M \times \{1, 2, \ldots, T\} \to \{0, 1\}$
  with the guarantee that $V(v, i) = 1$ if and only if $v = S^i(v_0)$.

The goal is to find a vertex v such that $V(v, T) = 1$ (i.e., the sink).

The Sink-of-Verifiable-Line Problem

Definition (Sink-of-Verifiable-Line, SVL)

A Sink-of-Verifiable-Line instance (S, V, T, v_0) consists of

- $T \in \{1, 2, \ldots, 2^M\}$,
- $v_0 \in \{0, 1\}^M$,
- $S : \{0, 1\}^M \to \{0, 1\}^M$,
- $V : \{0, 1\}^M \times \{1, 2, \ldots, T\} \to \{0, 1\}$
  with the guarantee that $V(v, i) = 1$ if and only if $v = S^i(v_0)$.

The goal is to find a vertex v such that $V(v, T) = 1$ (i.e., the sink).

Lemma

SVL is reducible to UEOPL (Unique-End-of-Potential-Line, which is known to lie in $\mathsf{PPAD} \cap \mathsf{PLS}$, *but not known to be complete).*

Definition (relaxed-Sink-of-Verifiable-Line, rSVL)

A relaxed-Sink-of-Verifiable-Line instance (S, V, T, v_0) consists of

- ...
- $V : \{0,1\}^M \times \{1, 2, \ldots, T\} \to \{0,1\}$
  with the guarantee that for every (v, i) such that $v = S^i(v_0)$, $V(v, i) = 1$.

The goal is to find one of the following:

(i) **The sink**: a vertex v such that $V(v, T) = 1$, or

(ii) **False positive**: a pair (v, i) such that $v \neq S^i(v_0)$ and $V(v, i) = 1$.

Definition (relaxed-Sink-of-Verifiable-Line, rSVL)

A relaxed-Sink-of-Verifiable-Line instance (S, V, T, v_0) consists of

- ...
- $V : \{0,1\}^M \times \{1, 2, \ldots, T\} \to \{0,1\}$
  with the guarantee that for every (v, i) such that $v = S^i(v_0)$, $V(v, i) = 1$.

The goal is to find one of the following:

(i) **The sink**: a vertex v such that $V(v, T) = 1$, or

(ii) **False positive**: a pair (v, i) such that $v \neq S^i(v_0)$ and $V(v, i) = 1$.

New Lemma

rSVL is also reducible to UEOPL.

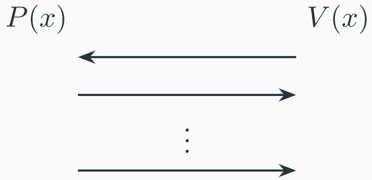
Today, we will see

Cryptographic hardness of rSVL – and therefore of $\mathsf{PPAD} \cap \mathsf{PLS}$ – based on

- hardness of counting # of satisfying assignments of a boolean formula (#SAT), and
- soundness of Fiat-Shamir transformation applying to some interactive protocol (the sumcheck protocol)

Preliminaries: IP, the sumcheck protocol

Unambiguous IPs



An interactive protocol (P, V) is a δ-sound interactive proof (IP) for L if:

- **Completeness:** For every $x \in L$, if V interacts with P on common input x, then V accepts with probability 1.
- **Soundness:** For every $x \notin L$ and every (computationally unbounded) cheating prover strategy $\widetilde{P}$, the verier V accepts when interacting with $\widetilde{P}$ with probability less than $\delta(|x|)$ for some function δ.

Unambiguous IPs

An interactive protocol (P, V) is a δ-sound interactive proof (IP) for L if:

- **Completeness:** For every $x \in L$, if V interacts with P on common input x, then V accepts with probability 1.
- **Soundness:** For every $x \notin L$ and every (computationally unbounded) cheating prover strategy $\widetilde{P}$, the verier V accepts when interacting with $\widetilde{P}$ with probability less than $\delta(|x|)$ for some function δ.

Remark

$\mathsf{NP} \subseteq \mathsf{IP}$ *as the prover can send the certificate to the verifier in one round.*

In fact, $\mathsf{IP} = \mathsf{PSPACE}$*, where* PSPACE *contains all languages that can be computed by a program (Turing machine) using polynomial space.*

Unambiguous IPs

An interactive protocol (P, V) is a (ϵ, δ)-unambiguosly sound interactive proof (IP) for L if:

- **Prescribed Completeness:** For every $x \in \{0,1\}^*$, if V interacts with P on common input x, then V outputs $L(x)$ with probability 1.
- **Soundness:** For every $x \notin L$ and every (computationally unbounded) cheating prover strategy $\widetilde{P}$, the verier V accepts when interacting with $\widetilde{P}$ with probability less than $\delta(|x|)$ for some function δ.
- **Unambiguity:** For every $x \in L$ and every (computationally unbounded) cheating prover strategy $\widetilde{P}$, if $\widetilde{P}$ deviates from P at some point, then at the end of the protocol V accepts with probability at most $\epsilon(|x|)$.

Unambiguous IPs

An interactive protocol (P, V) is a (ϵ, δ)-unambiguosly sound interactive proof (IP) for L if:

- **Prescribed Completeness:** For every $x \in \{0,1\}^*$, if V interacts with P on common input x, then V outputs $L(x)$ with probability 1.
- **Soundness:** For every $x \notin L$ and every (computationally unbounded) cheating prover strategy $\widetilde{P}$, the verier V accepts when interacting with $\widetilde{P}$ with probability less than $\delta(|x|)$ for some function δ.
- **Unambiguity:** For every $x \in L$ and every (computationally unbounded) cheating prover strategy $\widetilde{P}$, if $\widetilde{P}$ deviates from P at some point, then at the end of the protocol V accepts with probability at most $\epsilon(|x|)$.

Remark

Unambiguous IP *is more restrictive than* IP. *But in particular it will be good enough for an important problem...*

Interactive Sumcheck Protocol

- Fix a finite field $\mathbb{F}$ and a subset $\mathbb{H} \subseteq \mathbb{F}$ (usually $\mathbb{H} = \{0, 1\}$).
- The (not necessarily effcient) prover takes as input an n-variate low-degree polynomial $f : \mathbb{F}^n \to \mathbb{F}$.
  - Degree at most d in each variable; think of d as a constant, significantly smaller than $|\mathbb{F}|$
  - The verifier only has oracle access to f, and is given the constant $y = f(\mathbf{z}) \in \mathbb{F}$ for an oracle query $\mathbf{z} \in \mathbb{F}^n$. Each single oracle query runs in time $\text{poly}(n, d, \log(|\mathbb{F}|))$.
- The prover's goal is to convince a verifier that

$$\sum_{\mathbf{z} \in \mathbb{H}^n} f(\mathbf{z}) = y$$

for some value $y \in \mathbb{F}$.

Interactive Sumcheck Protocol: $(P_{SC}(y, f), V_{SC}^f(y))$

For $i \leftarrow 1$ to n:
At the beginning of round i, both P_{SC} and V_{SC} know y_{i-1} and $\beta_1, \ldots, \beta_{i-1} \in \mathbb{F}$, $y_0 = y$

(a) P_{SC} computes the degree-d univariate polynomial $g_i(x) = \sum_{z_{i+1},\ldots,z_n \in \mathbb{H}} f(\beta_1, \ldots, \beta_{i-1}, x, z_{i+1}, \ldots, z_n)$

$$\xrightarrow{\{\alpha_{i,\gamma} = g_i(\gamma)\}_{\gamma=0}^{d}}$$

(b) V_{SC} receives $d + 1$ field elements $\alpha_{i,\gamma}$ and interpolates the (unique) degree-d polynomial $\hat{g}_i$ such that $\hat{g}_i(\gamma) = \alpha_{i,\gamma}$.
V_{SC} then checks that $\sum_{x \in \mathbb{H}} \hat{g}_i(x) = y_{i-1}$. If not, then V_{SC} rejects.
(c) V_{SC} chooses a random element $\beta_i \in \mathbb{F}$, sets $y_i = g_i(\beta_i)$, and sends β_i to P_{SC}.

$$\xleftarrow[\text{Sends } \beta_i]{}$$

(*) At the last round, V_{SC} uses a single oracle call to f to check that $y_n = f(\beta_1, \ldots, \beta_n)$

Interactive Sumcheck Protocol: $(P_{SC}(y, f), V_{SC}^{f}(y))$

Remark

The sumcheck protocol is public-coin (fresh coins chosen at each round). If all randomnesses are in sky at the beginning of time, P could just send one message and get something that is almost NP *(except the randomnesses is in the sky).*

Remark

We can extend the sumcheck protocol for instances f with a partial assignment $(\beta_1, \ldots, \beta_j)$.

Interactive Sumcheck Protocol: $(P_{SC}(y, f), V_{SC}^{f}(y))$

Theorem

The sumcheck protocol is a $(d(n-j)/|\mathbb{F}|)$-unambiguously sound interactive proof system for prefixed L_{SC}, i.e., given a partial assignment $(\beta_1, \ldots, \beta_j)$.

Interactive Sumcheck Protocol: $(P_{SC}(y, f), V_{SC}^f(y))$

Theorem

The sumcheck protocol is a $(d(n-j)/|\mathbb{F}|)$-unambiguously sound interactive proof system for prefixed L_{SC}, i.e., given a partial assignment $(\beta_1, \ldots, \beta_j)$.

Theorem

The sumcheck protocol can be used to count/verify the number of satisfying assignments of a SAT formula (#P-complete, believed to be hard):

SAT formula $\Rightarrow$ 3SAT-4 formula $\Rightarrow$ low-degree polynomial

Example

How to get low-degree polynomial? Arithmetization!

Interactive Sumcheck Protocol: $(P_{SC}(y, f), V_{SC}^{f}(y))$

Theorem

The sumcheck protocol is a $(d(n-j)/|\mathbb{F}|)$-unambiguously sound interactive proof system for prefixed L_{SC}, i.e., given a partial assignment $(\beta_1, \ldots, \beta_j)$.

Theorem

The sumcheck protocol can be used to count/verify the number of satisfying assignments of a SAT formula (#P-complete, believed to be hard):

SAT formula $\Rightarrow$ 3SAT-4 formula $\Rightarrow$ low-degree polynomial

Example

How to get low-degree polynomial? Arithmetization!

$\neg p \Rightarrow (1-p)$ $\qquad$ $p_1 \wedge p_2 \Rightarrow p_1 \cdot p_2$

$p_1 \vee p_2 \Rightarrow 1-(1-p_1)(1-p_2)$

Non-interactive Proof Systems

A non-interactive proof system involves the prover sending a single message to the verifier

To give this proof system additional power, we assume that both prover and verifier have access to a common reference string (CRS):

$$\pi \leftarrow P(x, R)$$
$$V(x, R, \pi)$$

Non-interactive Proof Systems

A non-interactive proof system involves the prover sending a single message to the verifier

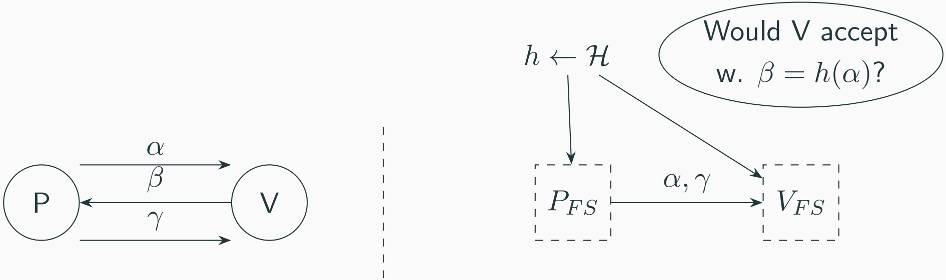
To give this proof system additional power, we assume that both prover and verifier have access to a common reference string (CRS):

$$\pi \leftarrow P(x, R)$$
$$V(x, R, \pi)$$

Remark

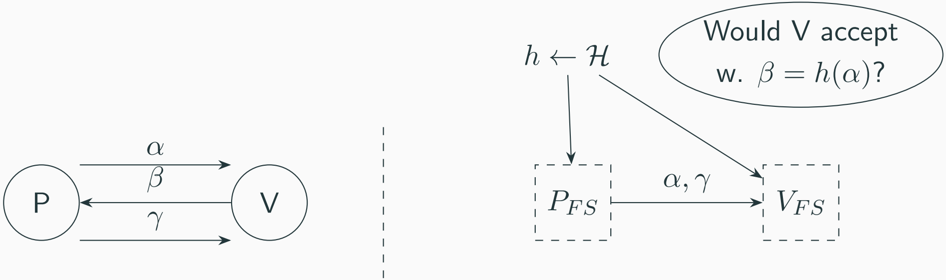
A non-interactive proof system is called an <u>argument</u> if the soundness and unambiguity properties hold only against computationally-bounded (i.e., poly(n)) cheating prover strategy P.

Fiat-Shamir Transformation



- Many-message protocol $\Rightarrow$ single-message protocol

Fiat-Shamir Transformation



- Many-message protocol $\Rightarrow$ single-message protocol
- Big open problem: Is the Fiat-Shamir Transformation sound?
  - Hash functions "looks like" random functions; generate "random bits" in a mutually agreed way
  - Negative results for some contrived protocols; don't know if the transformation is insecure when applying to a natural protocol

Applying Fiat-Shamir Transformation to Sumcheck

- We assume the Fiat-Shamir heuristic is unambiguously sound for the sumcheck protocol (this is true relative to a random oracle).
- Main result of this paper: Assuming there exists a hash function for which Fiat-Shamir Transformation of the sumcheck protocol is unambiguously sound and #SAT is hard, then rSVL is hard.

Corollary

If you show cryptographic assumption A implies that Fiat-Shamir transformation of the sumcheck protocol is unambiguously sound, then rSVL is hard

Applying Fiat-Shamir Transformation to Sumcheck

- We assume the Fiat-Shamir heuristic is unambiguously sound for the sumcheck protocol (this is true relative to a random oracle).
- Main result of this paper: Assuming there exists a hash function for which Fiat-Shamir Transformation of the sumcheck protocol is unambiguously sound and #SAT is hard, then rSVL is hard.

Corollary

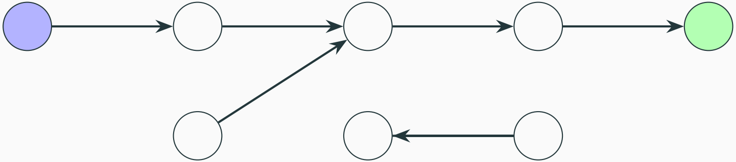
If you show cryptographic assumption A implies that Fiat-Shamir transformation of the sumcheck protocol is unambiguously sound, then rSVL is hard

Corollary

Relative to a random oracle, if #SAT is hard, then rSVL is hard.

Proof systems and PPAD

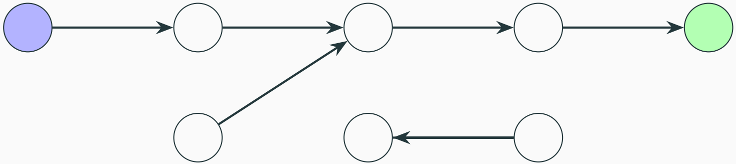
PSPACE and PPAD (EOL)?



PSPACE computation is a exponential graph, and the solution is a sink...

- Problem 1: finding predecessors
- Problem 2: many solutions

PSPACE and PPAD (EOL)?



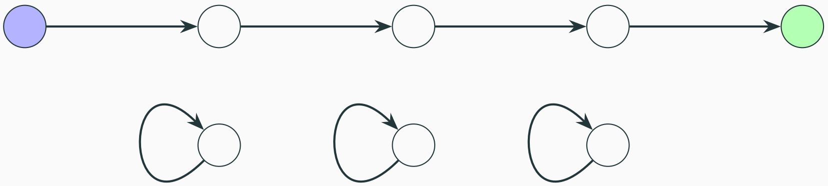
PSPACE computation is a exponential graph, and the solution is a sink...

- Problem 1: finding predecessors
- Problem 2: many solutions

We don't think there is a reduction from a PSPACE-complete problem to PPAD.

Idea: Associate each state with a proof, and a verifier circuit that outputs 1 if the state is a valid state in the line of computation

Proof systems and PPAD



Add proof that node is on "correct" path of the computation; nodes without proof become self-loops

- Computationally sound proofs suffice
- Need incremental unambiguous (to ensure the unique successor) proofs

Incrementally Verifiable Computation [Valiant '04]

Long computation (length L), performed via sequence of polynomial time steps:

- after step i, state is $\sigma_i = (y_i, \pi_i)$, where π_i is the proof
- step function $S(i, \sigma_i) = \sigma_{i+1} = (y_{i+1}, \pi_{i+1})$
- verifier V: Accept/reject given (i, y_i, π_i)

Completeness: $S^L(1, \sigma_1)$ gives correct output

Soundness: hard to find accepting $(i, \widetilde{\sigma_i})$

Incrementally Verifiable Computation [Valiant '04]

Long computation (length L), performed via sequence of polynomial time steps:

- after step i, state is $\sigma_i = (y_i, \pi_i)$, where π_i is the proof
- step function $S(i, \sigma_i) = \sigma_{i+1} = (y_{i+1}, \pi_{i+1})$
- verifier V: Accept/reject given (i, y_i, π_i)

Completeness: $S^L(1, \sigma_1)$ gives correct output

Soundness: hard to find accepting $(i, \widetilde{\sigma_i})$

Incremental unambiguous verifiable computation procedure for a hard-on-average problem $\implies$ hardness-on-average of rSVL

Incrementally Verifiable Computation [Valiant '04]

Long computation (length L), performed via sequence of polynomial time steps:

- after step i, state is $\sigma_i = (y_i, \pi_i)$, where π_i is the proof
- step function $S(i, \sigma_i) = \sigma_{i+1} = (y_{i+1}, \pi_{i+1})$
- verifier V: Accept/reject given (i, y_i, π_i)

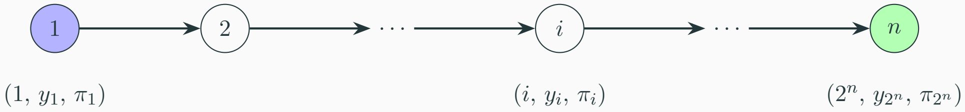
Completeness: $S^L(1, \sigma_1)$ gives correct output

Soundness: hard to find accepting $(i, \widetilde{\sigma_i})$

Incremental unambiguous verifiable computation procedure for a hard-on-average problem $\implies$ hardness-on-average of rSVL – the adversary has to either find the sink (solve the instance) or some cheating proof (break the soundness).

The Reduction

Incrementally Verifiable Counter for Satisfying Assignments



For a SAT formula $\varphi(z_1, z_2, \ldots, z_n)$, we build a graph such that:

- y_i – # of satisfying assignment $\in [0^n, i]$
- π_i – proof that y_i is correct
- Goal: construct successor and verifier circuits:
  - $S(i, y_i, \pi_i) \to (y_{i+1}, \pi_{i+1})$
  - $V(i, y_i, \pi_i)$ accepts if π_i proves that # of satisfying assignments $\in [0, i] = y_i$

Incrementally Verifiable Counter for Satisfying Assignments

$$1 \longrightarrow 2 \longrightarrow \cdots \longrightarrow i \longrightarrow \cdots \longrightarrow n$$

$(1,\, y_1,\, \pi_1)$ $\qquad$ $(i,\, y_i,\, \pi_i)$ $\qquad$ $(2^n,\, y_{2^n},\, \pi_{2^n})$

- Challenge: getting π_i to be of size poly(n)
  Solution: use the sumcheck protocol
- Challenge: protocol is interactive
  Solution: use Fiat-Shamir transformation
- Challenge: computing $S(i, y_i, \pi_i) = (y_{i+1}, \pi_{i+1})$
  Solution: recursive approach, incremental proof update

Incrementally Verifiable Counter for Satisfying Assignments

- In particular, given the counts for $\{y^\gamma\}_{\gamma=0}^{d}$ for the $(d+1)$ prefix sums with prefixes (β, γ) (sums of size 2^{n-j}), computing a proof for the count $y = (y_0 + y_1)$ of the sum with prefix β (a sum of size 2^{n-j+1}) reduces to computing a single additional prefix sum of size 2^{n-j} (by the sumcheck protocol).

Incrementally Verifiable Counter for Satisfying Assignments

- In particular, given the counts for $\{y^\gamma\}_{\gamma=0}^d$ for the $(d+1)$ prefix sums with prefixes (β, γ) (sums of size 2^{n-j}), computing a proof for the count $y = (y_0 + y_1)$ of the sum with prefix β (a sum of size 2^{n-j+1}) reduces to computing a single additional prefix sum of size 2^{n-j} (by the sumcheck protocol).
- Then, can merge the $(d+2)$ proofs into one by the sumcheck protocol.

Incrementally Verifiable Counter for Satisfying Assignments

- In particular, given the counts for $\{y^\gamma\}_{\gamma=0}^d$ for the $(d+1)$ prefix sums with prefixes (β,γ) (sums of size 2^{n-j}), computing a proof for the count $y = (y_0 + y_1)$ of the sum with prefix β (a sum of size 2^{n-j+1}) reduces to computing a single additional prefix sum of size 2^{n-j} (by the sumcheck protocol).
- Then, can merge the $(d+2)$ proofs into one by the sumcheck protocol.
- Construct the proof recursively, the overall process looks like a depth-first-search.
- The depth of the tree is at most n (at each level, reduce the # of variables by 1); each level contains at most $(d+2)$ proofs.
- Overall, size of the proof at each step is still polynomial.

Conclusion

Takeaways

- If we can build an incremental unambiguous computation procedure for a hard-on-average problem, then rSVL is hard (which implies in turn $\mathsf{PPAD} \cap \mathsf{PLS}$ is hard).
- Assuming Fiat-Shamir Transformation is unambiguously sound for the sumcheck protocol, we construct such procedure for #SAT.
- So –

  Finding a Nash Equilibrium (a PPAD-complete problem) Is No Easier Than Breaking Fiat-Shamir

Thanks for listening!