

Derandomization in Cryptography

Paper of Boaz Barak, Shien Jin Ong, Salil Vadhan

CPSC-637

Yixin Cao

21. April 2008

Warming Up

Primality is in P.

Manindra Agrawal, Neeraj Kayal, and Nitin Saxena
derandomized this in 2004.

Random Algorithm and Derandomization

Probability algorithms are practical, when no deterministic algorithm found for a problem, or the deterministic is too slow.

But it is also important to find a deterministic version from a probabilistic algorithm.

Random Algorithm and Derandomization

Probability algorithms are practical, when no deterministic algorithm found for a problem, or the deterministic is too slow.

But it is also important to find a deterministic version from a probabilistic algorithm.

Naïve derandomization: go over all possible coin tosses an probabilistic algorithm uses and act according to the majority.

Random Algorithm and Derandomization

Probability algorithms are practical, when no deterministic algorithm found for a problem, or the deterministic is too slow.

But it is also important to find a deterministic version from a probabilistic algorithm.

Naïve derandomization: go over all possible coin tosses an probabilistic algorithm uses and act according to the majority.

The core problem in this is : $\mathcal{P} = \mathcal{BPP}$?

Pseudorandom Number Generator

Definition 1 (Blum-Micali-Yao Generators). A function $G = \bigcup_m G_m : \{0, 1\}^l \rightarrow \{0, 1\}^m$ is a *BMV-type pseudorandom generator* with seed length $l = l(m)$, if G is computable in time $\text{poly}(l)$, and for every constant c , G_m is a $(m^c, 1/m^c)$ -pseudorandom generator against circuits for all sufficiently large m .

Definition 2 (Nisan-Wigderson Generators). A function $G = \bigcup_m G_m : \{0, 1\}^l \rightarrow \{0, 1\}^m$ is a *NW-type pseudorandom generator* with seed length $l = l(m)$, if G is computable in time $2^{O(l)}$, and G_m is a $(m^2, 1/m^2)$ -pseudorandom generator against circuits for all sufficiently large m .

Pseudorandom Number Generator

Definition 1 (Blum-Micali-Yao Generators). A function $G = \bigcup_m G_m : \{0, 1\}^l \rightarrow \{0, 1\}^m$ is a *BM-Y-type pseudorandom generator* with seed length $l = l(m)$, if G is computable in time $\text{poly}(l)$, and for every constant c , G_m is a $(m^c, 1/m^c)$ -pseudorandom generator against circuits for all sufficiently large m .

Definition 2 (Nisan-Wigderson Generators). A function $G = \bigcup_m G_m : \{0, 1\}^l \rightarrow \{0, 1\}^m$ is a *NW-type pseudorandom generator* with seed length $l = l(m)$, if G is computable in time $2^{O(l)}$, and G_m is a $(m^2, 1/m^2)$ -pseudorandom generator against circuits for all sufficiently large m .

Difference

BMV-generators can fool even circuits with greater running time than the generator.

NW-generators are allowed greater running time than the adversarial circuit.

Difference

BMV-generators can fool even circuits with greater running time than the generator.

NW-generators are allowed greater running time than the adversarial circuit.

You do not know the time of your adversary, so it is ridiculous to say that you have more running time than it. Then it seems NW-generators are *of no use* in cryptography.

Hitting Set Generators

Definition 3 (Hitting Set Generators(HSG)). We say that H is an ε -hitting set generator against circuits if for every $m, s \in \mathbb{N}$, and circuit $C : \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s , the following holds:

$$\Pr[C(U_m) = 1] > \varepsilon \implies \exists y \in H(1^m, 1^s) \text{ such that } C(y) = 1.$$

A HSG is a deterministic algorithm $H(1^m, 1^s)$ that outputs a set of strings of length m .

HSGs is *weaker* notions of pseudorandom generators.

Properties of HSG

Definition 4. We say hitting set generator $H(1^m, 1^s)$ is *efficient* if its running time is polynomial in m and s .

HSG can only be directly used to derandomize algorithms with one-sided error(\mathcal{RP}). While pseudorandom generators are applicable to algorithms with two-sided error(\mathcal{BPP}).

Non-deterministic Adversary

However, it is not true. The NW-generators can fool *nondeterministic* circuits.

Non-deterministic Adversary

However, it is not true. The NW-generators can fool *nondeterministic* circuits.

The reason is: even if a non-deterministic circuit can guess the seed, it does not have enough time to evaluate the generator on it.

Interactive Proofs

An *interactive proof* is an interactive protocol in which a prover P (with unlimited computational powers) tries to convince a probabilistic polynomial-time verifier V the validity of a certain statement.

Interactive Proofs

An *interactive proof* is an interactive protocol in which a prover P (with unlimited computational powers) tries to convince a probabilistic polynomial-time verifier V the validity of a certain statement.

It is probabilistic!

Interactive Proofs

An *interactive proof* is an interactive protocol in which a prover P (with unlimited computational powers) tries to convince a probabilistic polynomial-time verifier V the validity of a certain statement.

It is probabilistic!

PS: The class of languages L possessing interactive proofs is denoted as \mathcal{IP} .

Properties of Interactive Proofs

(Efficiency) On common input x , the number and total length of messages exchanged between P and V are bounded by a polynomial in $|x|$, and V is a probabilistic polynomial-time machine.

(Completeness) If $x \in L$, then $\Pr[(P, V)(x) = 1] \geq \frac{2}{3}$

(Soundness) If $x \notin L$, then for any P^* , $\Pr[(P^*, V)(x) = 1] \leq \frac{1}{3}$

Properties of Interactive Proofs

(Efficiency) On common input x , the number and total length of messages exchanged between P and V are bounded by a polynomial in $|x|$, and V is a probabilistic polynomial-time machine.

(Completeness) If $x \in L$, then $\Pr[(P, V)(x) = 1] \geq \frac{2}{3}$

(Soundness) If $x \notin L$, then for any P^* , $\Pr[(P^*, V)(x) = 1] \leq \frac{1}{3}$

An interactive proof system has *perfect completeness* if the probability is 1; has *perfect soundness* if the probability is 0.

Zero-Knowledge Proof & Witness-Indistinguishable Proof

For a language $L \in \mathcal{NP}$

† zero-knowledge proofs leak no knowledge at all.

‡ witness-indistinguishable proof system leaks no knowledge about which witness is being used by the prover.

Zero-Knowledge Proof & Witness-Indistinguishable Proof

For a language $L \in \mathcal{NP}$

† zero-knowledge proofs leak no knowledge at all.

‡ witness-indistinguishable proof system leaks no knowledge about which witness is being used by the prover.

Therefore, Their relation is:

witness indistinguishable proof \prec zero knowledge proof.

Interaction in the Proofs

Both *interaction* and *randomization* are necessary for the existence of *zero-knowledge proofs*.

Interaction in the Proofs

Both *interaction* and *randomization* are necessary for the existence of *zero-knowledge proofs*.

However, the interaction can be reduced in *witness-indistinguishable proof*, and 3-message and 2-message ones are found. While the 2-message one is proved *impossible* in zero-knowledge proofs.

Interaction in the Proofs

Both *interaction* and *randomization* are necessary for the existence of *zero-knowledge proofs*.

However, the interaction can be reduced in *witness-indistinguishable proof*, and 3-message and 2-message ones are found. While the 2-message one is proved *impossible* in zero-knowledge proofs.

Interaction in the Proofs

Both *interaction* and *randomization* are necessary for the existence of *zero-knowledge proofs*.

However, the interaction can be reduced in *witness-indistinguishable proof*, and 3-message and 2-message ones are found. While the 2-message one is proved *impossible* in zero-knowledge proofs.

No the question: how about 1-message, i.e., can the interaction be completely removed in *witness-indistinguishable*?

Basic Idea

By derandomizing the verifier in the ZAPs via an NW-generator, a standard \mathcal{NP} proof system with witness indistinguishability is obtained.

The key point here is the preservation of the witness indistinguishability property in the building of \mathcal{NP} proof systems.

The Protocol -Prover

On common input $x \in \{0, 1\}^n$ and auxiliary input w for the prover, such that $(x, w) \in W_L$, do:

Prover's message:

1. Compute $(r_1, \dots, r_m) \stackrel{\text{def}}{=} H(1^{l(n)}, 1^{p(n)})$.
2. Using the auxiliary input (witness) w and the ZAP prover algorithm, compute for every $i \in [1, m]$, a string π_i that is the prover's response.
3. Send to verifier (π_1, \dots, π_m) .

The Protocol - Verifier

continued from last page.

Verifier's test:

1. Compute $(r_1, \dots, r_m) \stackrel{\text{def}}{=} H(1^{l(n)}, 1^{p(n)})$.
2. Given prover's message (π_1, \dots, π_m) , run the ZAP verifier on the transcript (x, r_i, π_i) , for every $i \in [1, m]$.
3. Accept if the ZAP verifier accepts *all* these transcripts.

Conclusion

Lemma 5. *This protocol is a perfectly sound proof system for L .*

Lemma 6. *This protocol is a witness indistinguishable (WI) proof system for L .*

The End!