**CS 355: Topics in Cryptography** **Spring 2019**

# Lecture 17: Succinct Non-interactive Arguments

Dima Kogan (based on notes by David Wu)

## 1 Introduction

We revisit the topic of proof systems. Recall that in a proof system, a prover wants to convince a verifier of some statement $x \in \mathcal{L}$. We consider the case of Boolean-circuit satisfiability: $\mathcal{L}_{\mathcal{C}} = \{x \in \{0,1\}^n \mid \exists w \in \{0,1\}^\ell \text{ s.t. } C(x,w) = 1\}$ where $\mathcal{C}$ is some Boolean circuit.[1] We have seen two proof systems for circuit satisfiability:

- The straightforward proof system that consists of the prover sending the verifier the witness $w$.
- In Homework 2 we have seen a zero-knowledge Sigma protocol for circuit satisfiability. It could also be made non-interactive in the random oracle model using the Fiat-Shamir heuristic.

One downside of both of those proof system is that the communication complexity (or length of the non-interactive proof) was $\Omega(|\mathcal{C}|)$.

In general, we can consider the following design parameters:

- Type of soundness (statistical/computational): if the proof is only sound against computationally bounded malicious provers, the proof is called an *argument*.
- Knowledge complexity
- Time complexity (of the prover/verifier)
- Interaction (or lack of thereof)
- Communication complexity

Consider for example the two following applications:

1. **Verifiable computation**: a client holds $x$ and wants to delegate to a server some computation $f(x)$. Can the server prove to the client that the computation has been performed correctly? One trivial solution would be for the client to compute $f(x)$ on its own, but this beats the purpose. Could we obtain verification time that is sublinear in the time it takes to evaluate the circuit for $f$. Verifiable computation could be useful for example for a mobile device delegating some computation to the cloud, or a slow processor checking the computation of another, potentially faster yet untrusted, processor in the same system. Note that this application does not necessarily require zero-knowledge. For more information on verifiable computation see [WB15].
2. **Blockchain applications**: confidential transactions where the amount value of the transaction or the identity of the sender and recipient are kept private from 3rd parties. However since space on the Blockchain is expensive, we would like the proof to be as short as possible.

[1] As always, we actually implicitly refer to a circuit family $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$.

Both of the above applications motivate SNARGs: **S**uccinct **N**on-interative **AR**guments. We say that a proof $\pi$ is succinct if:

1. $|\pi| = \text{poly}(\lambda, \log|\mathcal{C}|)$ and
2. the verification time is $\text{poly}(\lambda, |x|, \log|\mathcal{C}|)$.

Note that both of the above are much smaller than the size of the circuit $\mathcal{C}$. As mentioned above, an argument means it is only sound against computationally bounded provers.

One can then extend SNARGs further to obtain SNARKs which are **S**uccinct **N**on-interactive **A**rguments of **K**nowledge, meaning they are also a proof of knowledge. Furthermore, one can also get zkSNARKs that additionally provide zero-knowledge. Such zkSNARKs form the basis for the zCash cryptocurrency [BCTV14, Gab17].

## 2 SNARGs from PCPs

Many of the SNARGs constructions follow the next recipe [BCI+13]:

1. Construct an information-theoretically secure proof system secure against prover satisfying some sort of constraints.
2. Use cryptographic tools to bind computationally-bounded provers to respect the above constraints.

The security of the second step is usually proven in either the random oracle model or in the CRS (common reference string) model, which we'll introduce later today.

The information-theoretic primitive behind many of the SNARG constructions is *probabilistically checkable proof*, or PCP. Informally, a PCP is a (non-interactive) proof system, in which the prover outputs a proof $\pi \in \{0,1\}^m$, and the verifier, which is an efficient randomized algorithm, can check the proof by reading the input and only a small number of bits of the proof. The PCP theorem [ALM+98, AS98]—one of the most amazing results in Complexity Theory—states that every NP-language has a PCP in which the verifier reads only a constant (i.e., independent of the statement length) number of bits of the proof.

In the last homework assignment you will see how to transform such a PCP into a SNARG in the random oracle model. However, the resulting SNARG is very inefficient in practice, with the main bottleneck being the extremely high time required by the prover to construct the proof.

### 2.1 Linear PCPs

Here we take a different approach and show how to construct a SNARG from a different type of PCP, called a *linear PCP* [IKO07, BCI+13]. Informally, the difference between a classic PCP and a linear PCP is the following:

- In a "classic PCP", the proof is a string $\pi \in \{0,1\}^m$, each query is a bit position $i \in [m]$, and the response is the $i$-th bit $\pi_i$ of the proof.
- In a linear PCP, the proof is as a vector $\pi \in \mathbb{F}^m$, each query is also a vector $q \in \mathbb{F}^m$, and the response is given by the inner product $\langle \pi, q \rangle \in \mathbb{F}$. The name comes from the fact that each query is essentially a linear combination of the elements of the proof.

The completeness and soundness conditions are then as follows:

Completeness: If $(x, w) \in R_{\mathcal{L}}$, then if we set $\pi \leftarrow P(x, w)$, then $\Pr\left[V^{\langle \pi, \cdot \rangle}(x)\right] = 1$.[2]

Soundness: For all $x \notin \mathcal{L}$ and all $\pi^* \in \mathbb{F}^m$, we have $\Pr[V^{\langle \pi^*, \cdot \rangle}(x) = 1] \leq \epsilon$.

There are several constructions of linear PCPS, among them:

- Linear PCPs based on the Walsh-Hadamard code [ALM+98, IKO07] with query length $m = O(|\mathcal{C}|^2)$.
- Linear PCPs based on quadratic span programs [GGPR13] with query length $m = (|\mathcal{C}|)$.

An additional useful property that the above linear PCPs have is that they are *oblivious*, meaning that the queries $q_1, \ldots, q_k$ do not depend on the input statement $x$.

# 3 From Linear PCPs to SNARGs

We sketch the transformation from a linear PCP to a SNARG *with preprocessing* for a *designated verifier*.

In this model, the proof phase is preceded by an additional setup phase where:

$\mathsf{Setup}(1^\lambda) \to (\sigma, \tau)$ where $\sigma$ is a public common reference string, and $\tau$ is a *private* verification state.

$\mathsf{Prove}(\sigma, x, w) \to \pi'$

$\mathsf{Verify}(\tau, x, \pi') \to \mathsf{accept/reject}$

The SNARG is said to be designated verifier since the the verification state is given only to the verifier, and not to the prover, and soundness holds as long as the prover does not learn the verification state.[3]

#### Strawman SNARG from linear PCP

$\mathsf{Setup}(1^\lambda) \to (\sigma, \tau)$: generate queries $q_1, \ldots, q_k \in \mathbb{F}^m$ and linear PCP verification state $\tau_{\text{LPCP}}$. Output $\sigma = (q_1, \ldots, q_k)$ and $\tau = \tau_{\text{LPCP}}$.

$\mathsf{Prove}(\sigma, x, w) \to \pi'$ :

1. Construct a linear PCP $\pi$ for $x, w$.
2. Compute responses $r_1 = \langle \pi, q_1 \rangle, \ldots, r_k = \langle \pi, q_k \rangle$,
3. Output the proof that consists of the linear PCP responses $r_1, \ldots, r_k \in \mathbb{F}$.

$\mathsf{Verify}(\tau, x, \pi') \to \mathsf{accept/reject}$: run the linear-PCP verification procedure using $r_1, \ldots, r_k$ and $\tau_{\text{LPCP}}$.

This strawman on itself is not sound, since a malicious prover does not need to observe the linear-PCP requirements:

[2] The notation $V^{\langle \pi, \cdot \rangle}$ means that the verifier can issue linear queries to the proof as defined above.

[3] In contrast, a publicly-verifiable SNARG in the CRS model requires no secret state for verification, but rather only the public CRS $\sigma$. Such a SNARG may still have a *trusted setup*. Secrets generated during setup must be destroyed. If the setup is compromised or the secrets leak, the security might no longer hold.

**Problem 1: A malicious prover can choose the proof $\pi \in \mathbb{F}^m$ *after* seeing the verifier's queries.** Note that in the (linear-)PCP model, the prover outputs the proof string *before* the verifier samples the random queries, and if the order is reversed, the proof system is no longer sound, since the prover can "cook" the proof to match the queries.

**Solution: Encrypt the queries with additively-homomorphic encryption.** We modify the proof system as follows:

$\mathsf{Setup}(1^\lambda) \to (\sigma, \tau)$ :

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
2. Generate queries $q_1, \ldots, q_k \in \mathbb{F}^m$ and linear PCP verification state $\tau_{\text{LPCP}}$.
3. Encrypt each element of each of the queries: for $i \in [k]$ and $j \in [m]$, set $\mathsf{ct}_{ij} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, q_{ij})$ and $\mathsf{ct}_i = (\mathsf{ct}_{i1}, \ldots, \mathsf{ct}_{im})$.
4. Output $\sigma = (\mathsf{pk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ and $\tau = (\mathsf{sk}, \tau_{\text{LPCP}})$.

The prover can then homomorphically compute

$$\mathsf{ct}'_i \leftarrow \mathsf{Enc}(\mathsf{pk}, r_i) = \mathsf{Enc}(\mathsf{pk}, \langle \pi, q_i \rangle) = \sum_{j \in [m]} \pi_j \cdot \mathsf{Enc}(\mathsf{pk}, q_{ij}) = \sum_{j \in [m]} \pi_j \cdot \mathsf{ct}_{ij}\,.$$

The verifier decrypts $\mathsf{ct}'_1, \ldots, \mathsf{ct}'_k$ using the secret key to obtain the responses $r_1, \ldots, r_k$ and applies the linear-PCP verification using the state $\tau_{\text{LPCP}}$.

**Problem 2: A malicious prover might not follow linear strategy.** In other words we need to somehow ensure that each $\mathsf{ct}'_i$ is equal to $\mathsf{Enc}(\mathsf{pk}, \langle \pi, q_i \rangle)$ for some $\pi \in \mathbb{F}^m$.

**Solution: Assume that the encryption scheme is "linear-only".** We won't go into much detail here, but say informally that we assume that "any ciphertext that the adversary can compute can be explained by a linear function of the provided ciphertexts". (The formal definition captures this by requiring the existence of an extractor that can extract the linear function from the adversary.) This is not a typical cryptographic assumption (non-falsifiable).

**Problem 3: A malicious prover can apply a *different* linear function on each query.** In other words, even if we have solved the previous problem and the response to each query is given by a linear function: $r_1 = \langle \pi_1, q_1 \rangle, \ldots, r_k = \langle \pi_k, q_k \rangle$, we need to somehow assure that all the $\pi_i$ are consistent (since otherwise we cannot rely on the soundness of the LPCP).

**Solution: add a random linear check.** We add an additional random query of the form:

$$\alpha_1, \ldots, \alpha_k \xleftarrow{\text{R}} \mathbb{F} \quad \text{and} \quad q_{k+1} \leftarrow \sum_{i \in [k]} \alpha_i q_i\,.$$

The verifier then checks that $r_{k+1} \stackrel{?}{=} \sum_{i \in [k]} \alpha_i r_i$.

Observe that

$$\left( \sum_{i \in [k]} \alpha_i r_i \right) - r_{k+1} = \sum_{i \in [k]} \alpha_i \langle \pi_i, q_i \rangle - \langle \pi_{k+1}, q_{k+1} \rangle = \sum_{i \in [k]} \alpha_i \langle \pi_i, q_i \rangle - \langle \pi_{k+1}, \sum_{i \in [k]} \alpha_i q_i \rangle = \sum_{i \in [k]} \alpha_i \langle \pi_i - \pi_{k+1}, q_i \rangle\,.$$

If the prover is honest and uses the same linear function $\pi_i = \pi$ for all responses $i \in [k+1]$, then the check clearly passes. However, if there exists an $i^*$ such that $\pi_i \neq \pi_{k+1}$, then the probability over the choice of $\alpha_{i^*}$ (for every choice of all other $\alpha_i$) that this sum is zero is at most $1/|\mathbb{F}|$.

**Putting everything together:**

$\mathsf{Setup}(1^\lambda) \to (\sigma, \tau)$:

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ for a linear-only encryption scheme
2. Generate queries $q_1, \ldots, q_k \in \mathbb{F}^m$, consistency query $q_{k+1}$, and linear PCP verification state $\tau_{\text{LPCP}}$.
3. Encrypt each element of each of the queries: for $i \in [k+1]$ and $j \in [m]$, set $\mathsf{ct}_{ij} \leftarrow \mathsf{Enc}(\mathsf{pk}, q_{ij})$ and $\mathsf{ct}_i = (\mathsf{ct}_{i1}, \ldots, \mathsf{ct}_{im})$.
4. Output $\sigma = (\mathsf{pk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_{k+1})$ and $\tau = (\mathsf{sk}, \tau_{\text{LPCP}})$.

$\mathsf{Prove}(\sigma, x, w) \to \pi'$:

1. Construct a linear PCP $\pi$ for $x, w$.
2. Compute encrypted responses $\mathsf{ct}'_i = \mathsf{Enc}(\mathsf{pk}, \langle \pi, q_i \rangle)$ from encrypted queries $\mathsf{ct}_1, \ldots, \mathsf{ct}_{k+1}$.
3. Output SNARG $\pi' = (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_{k+1})$.

$\mathsf{Verify}(\tau, x, \pi') \to \mathsf{accept}/\mathsf{reject}$: decrypt ciphertexts in $\pi'$ using $\mathsf{sk}$ from $\tau$ and verify response using the linear-PCP verification procedure and its state $\tau_{\text{LPCP}}$ from $\tau$.

*Completeness* follows from the completeness of the linear PCP and the correctness of the encryption scheme.
*Soundness* follows from the following:

- Linear-only encryption guarantees that each response can be explained by a linear function.
- Consistency query guarantees that the same linear function $\pi$ is used for all responses.
- Semantic security of the encryption scheme guarantees that the prover's linear function is independent of the queries.
- Soundness of the linear PCP guarantees overall soundness.

*Succinctness* follows from the fact the SNARG consists of $k+1$ ciphertexts where $k$ is the number of queries in the linear PCP, which is constant.

## References


[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, 2013. https://eprint.iacr.org/2012/718.pdf.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security Symposium*, 2014. https://eprint.iacr.org/2013/879.pdf.

[Gab17] Ariel Gabizon. Explaining snarks, 2017. https://z.cash/blog/snark-explain/.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT*, 2013. https://eprint.iacr.org/2012/215.pdf.

[IKO07] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *IEEE Conference on Computational Complexity*, 2007.

[WB15] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, 2015. https://doi.org/10.1145/2641562.

[Wu18] David J. Wu. CS355 lecture notes, 2018. https://crypto.stanford.edu/cs355/18sp/lec16.pdf.