An Overview of Proof Systems

Steven Xu and Jonathan Li

May 8, 2022

Abstract

In this survey, we compile a introduction for various constructions of argument and proof systems. We start off with a construction of linear PCPs, and then go over a construction of zk-SNARKs using square span programs. Finally, we take a look at a construction of preprocessing SNARKs from LIPs.

1 Introduction

In cryptography, a proof system is a way for a prover to convince a verifier of true some statement. The correctness and security of proof systems is defined by completeness and soundness— the ability to prove all statements of a certain type and the inability for the prover to lie respectively. Proof systems have many uses, especially in the world of authentication, where a user could use a proof system to convince a system of its identity. Modern proof systems come in many types, with some aiming to be zero knowledge, succinct, non-interactive, and/or proofs of knowledge.

2 Preliminaries

2.1 Interactive Proof and Argument Systems

An **interactive proof system** is a system between a prover P and a verifier V, where the prover is attempting to convince the verifier that a certain statement x is in some language $\mathcal{L}$. P and V exchange messages until V, and at the end of the interaction, V either accepts or rejects the proof, with $\langle P, V\rangle(x)$ denoting the V's choice: 0 for reject and 1 for accept. We also use $\text{View}_V(\langle P, V\rangle(x))$ to denote the transcript of the interaction between P and V.

The verifier is computationally bounded and must run in randomized polynomial time, whereas the prover is not computationally bounded. An interactive argument system is then an interactive proof system, except the prover can be assumed to be computationally bounded as well.

A proof system must satisfy two properties—**completeness** and **soundness**. Completeness states that for all statements $x \in \mathcal{L}$, the verifier must *always* accept at the end of the prover-verifier exchange. Soundness states that for all $x \notin \mathcal{L}$ and potentially malicious prover P^*, $\Pr[\langle P^*, V\rangle(x) = 1] < \frac{1}{3}$.

2.2 Proofs of Knowledge and Zero Knowledge

A proof system is considered a **proof of knowledge** system for some relation R if there exists some efficient extractor $\mathcal{E}$ such that for all statements x and (potentially malicious) provers P^*, we have that $\Pr[w \leftarrow \mathcal{E}^{P^*} : R(x, w) = 1] \geq \Pr[\langle P^*, V\rangle(x) = 1] - \varepsilon$. $\mathcal{E}^{P^*}$ means that the extractor is given blackbox access

to P^*, including the power to rewind. In essence, a proof of knowledge system should convince the verifier that the prover knows a witness w to a statement x. Note that knowledge implies soundness, since the prover cannot know a witness w if it doesn't exist (x is not a true statement).

An interactive proof system is considered **zero knowledge** if for all (potentially malicious) verifiers V^*, there exists an efficient simulator S such that for all $x \in \mathcal{L}$, the distributions $\text{View}_V(\langle P, V\rangle(x))$ and $S(x)$ are indistinguishable. There are three variants of zero knowledge: perfect, statistical, and computational, requiring that the two distributions be the equal, statistically indistinguishable, and computationally indistinguishable respectively. S is given access to the V^*'s random coins and the statement x.

For non-interactive proof systems, there are two models of security for zero knowledge. The first is the CRS model, where a common reference string (CRS) σ is setup by a trusted third party ahead of time. S and $\mathcal{E}$ are given access to a simulation trapdoor τ which is also output by the setup, but which neither P or V ever see. The second model is the random oracle model, where both P and V are given access to some shared random oracle H, and $S, \mathcal{E}$ are allowed to program the random oracle.

Combining all our definitions above, the term **SNARK** means succinct non-interactive argument of knowledge, and a **zk-SNARK** is a SNARK which is also zero knowledge.

3 Linear PCPs and LIPs

Definition 1. A **linear probabilistically-checkable proof (LPCP)** system for a binary relation $\mathcal{R}$ over a finite field $\mathbb{F}$ is a proof system in which the PCP oracle is restricted to compute a linear function $\pi : \mathbb{F}^m \to \mathbb{F}$ of the verifier's queries. More formally, let P_{LPCP} and V_{LPCP} be a deterministic prover algorithm and a probabilistic oracle verifier algorithm, respectively. We say that $(P_{\mathsf{LPCP}}, V_{\mathsf{LPCP}})$ is an input-oblivious k-query LPCP for $\mathcal{R}$ over $\mathbb{F}$ with knowledge error ε and query length m if it satisfies the following:

- **Syntax.** The verifier makes k input-oblivious queries to π on any input x and oracle π then decides whether to accept or reject.
- **Completeness.** For every $(x, w) \in \mathcal{R}$, $P_{\mathsf{LPCP}}(x, w)$ outputs a description of a linear function $\pi : \mathbb{F}^m \to \mathbb{F}$ such that $V^{\pi}_{\mathsf{LPCP}}(x)$ accepts with probability 1.
- **Knowledge.** There exists a knowledge extractor E_{LPCP} such that for every linear function $\pi^* : \mathbb{F}^m \to \mathbb{F}$, if the probability that $V^{\pi^*}_{\mathsf{LPCP}}(x)$ accepts is greater than ε, then $E^{\pi^*}_{\mathsf{LPCP}}(x)$ outputs w such that $(x, w) \in \mathcal{R}$.

LPCPs are of interest because they are easier to construct than traditional PCPs and can be used to construct LIPs, which can in turn be used to construct SNARKs. Such a construction is presented in section 5.

Definition 2. A **linear interactive proof (LIP)** over a finite field $\mathbb{F}$ is defined as a standard interactive proof (as given by Goldwasser et al.) but with the following differences:

- Each message exchanged between the prover P_{LIP} and the verifier V_{LIP} is a vector $\mathbf{q}_i \in \mathbb{F}^m$.
- Each of the prover's messages is computed by applying a linear function $\Pi_i : \mathbb{F}^m \to \mathbb{F}^k$, determined only by the input x, the witness w, and the round number i, to the verifier's previous messages $(\mathbf{q}_1, \ldots, \mathbf{q}_i)$.
- Knowledge should hold only with respect to affine prover strategies $\Pi^* = (\Pi, \mathbf{b})$, where Π is a linear function, and $\mathbf{b}$ is some affine shift.

The following LPCP construction from the Walsh-Hadamard Code is given by [BCI+13], which is an extension of the Hadamard-based PCP of Arora et al. (ALMSS).

Let $\mathbb{F}$ be a finite field, and let $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^\ell$ be an arithmetic circuit over $\mathbb{F}$ of size s.

Definition 3. The **LPCP prover algorithm** P_{LPCP} takes as input $(\mathbf{x}, \mathbf{w}) \in \mathbb{F}^n \times \mathbb{F}^h$ such that $C(\mathbf{x}, \mathbf{w}) = 1$, computes the values $z_1, \ldots, z_s$ of all wires in $C(\mathbf{x}, \mathbf{w})$, and outputs the linear function $\pi : \mathbb{F}^{s+s^2} \to \mathbb{F}$ given by z_i for all $i \in [s]$ and $z_i \cdot z_j$ for all $i, j \in [s]$.

Definition 4. The **LPCP query algorithm** Q_{LPCP} consists of a matrix $A_C \in \mathbb{F}^{s \times (s+s^2)}$ and a vector $\mathbf{b}_C \in \mathbb{F}^{s-n}$. Vectors $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3 \in \mathbb{F}^{s+s^2}$ are computed and output as follows:

1. $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \leftarrow \mathbb{F}^{2s}$. Define $\mathbf{r}_\mathbf{x}$ as the first n coordinates of $\mathbf{r}_1$ and $\mathbf{r}_C$ as the last $s - n$ coordinates of $\mathbf{r}_1$;
2. $\mathbf{q}_1 := \mathbf{r}_1 \cdot A_C$;
3. the first s elements of $\mathbf{q}_2$ is $\mathbf{r}_2$ and the last s^2 elements are 0;
4. the first s elements of $\mathbf{q}_3$ are 0 and the last s^2 elements are $\mathbf{r}_2[i] \cdot \mathbf{r}_2[j]$ for all $i, j \in [s]$.

Definition 5. The **LPCP decision algorithm** D_{LPCP} takes as input $\mathbf{x}$, $\mathbf{u} = (u_C, \mathbf{r}_\mathbf{x})$, and answers $a_1, a_2, a_3 \in \mathbb{F}$ and verifies that $(a_1 - (u_C + \langle \mathbf{r}_\mathbf{x}, \mathbf{x} \rangle), a_2^2 - a_3) = (0, 0)$.

Theorem 1. *The given construction is a 3-query LPCP with knowledge error* $2/|\mathbb{F}|$*, query length* $s + s^2$*, and degree* $(2, 2)$*. Furthermore,*

- P_{LPCP} *is an arithmetic circuit of size* $O(s^2)$*;*
- Q_{LPCP} *is an arithmetic circuit of size* $O(s^2)$*;*
- D_{LPCP} *is an arithmetic circuit of size* $O(n)$*.*

4 zk-SNARKs from Square Span Programs

[DFGK14] introduces a new characterization of NP relations called square span programs, which they then use along with bilinear groups maps to construct a zk-SNARK. We'll refer to this zk-SNARK as the square span argument system.

Definition 6. A **square span program** Q over a finite field $\mathbb{F}$ is a set of $m+1$ polynomials $v_0(x), \ldots, v_m(x) \in \mathbb{F}[x]$ along with a target polynomial $t(x) \in \mathbb{F}[x]$ such that $\deg v_i(x) \leq \deg t(x)$ for all i. Q is defined to accept an input $a_1, \ldots, a_l \in \mathbb{F}$ if and only if there exist $a_{l+1}, \ldots, a_m \in \mathbb{F}$ such that

$$t(x) \text{ divides } \left(v_0(x) + \sum_{i=1}^{m} a_i v_i(x) \right)^2 - 1.$$

m is called the size of Q, and $d = \deg t(x)$ is called the degree of Q. We say that Q satisfies a circuit $C : \{0,1\}^l \to \{0,1\}$ if Q accepts if and only if $a_1, \ldots, a_l \in \{0,1\}$ and $C(a_1, \ldots, a_l)$.

Theorem 2. *Square span programs are NP-complete, and a boolean circuit* C *consisting of* m *wires and* n *gates has a square span program over* $\mathbb{Z}_p$ *of size* m *and degree* $m + n$ *which verifies* C*, where* $p \geq \max(n, 8)$*.*

In the construction of the square span argument system, we will split the input into $a_1, \ldots, a_{l_u}$ and $a_{l_u+1}, \ldots, a_l$. The first part will contain the object x of the argument (e.g a graph in a Hamiltonian cycle argument), and the second part (of length $l_w = l - l_u$) will contain the witness w. C will check that w is a valid witness for x for the relevant relation $R \subset U \times W$.

The next tool we'll need are bilinear group maps. The bilinear group generator defined below gives us a bilinear group map along with its three associated groups (2 input, 1 output) for every security parameter λ.

Definition 7. $\mathcal{G}$ is called a **bilinear group generator** for any security parameter λ if $\mathbb{G}(1^\lambda)$ returns a five-tuple $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$, where $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ are groups of order p, and $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$. e must be a bilinear map, i.e. $e(U^a, V^b) = e(U, V)^{ab}$ for all U, V. If U, V are generators, then $e(U, V)$ must be a generator.

For the remainder of this section, G, H, V are taken to be in $\mathbb{G}$ and $\hat{G}, \hat{H}, \hat{V} \in \hat{G}$ by default. We will need three assumptions to hold for our groups and bilinear group map in order for the square span argument system to be knowledge-sound:

1. The q-power knowledge of exponent (q-**PKE**) assumption states that given $G, \hat{G}, G^s, \hat{G}^s, \ldots, G^{s^q}, \hat{G}^{s^q}$ as well as an auxiliary input generator Z, it's difficult to create $V, \hat{V}$ such that $e(V, \hat{G}) = e(G, \hat{V})$ without knowing $a_0, \ldots, a_q$ such that $V = \prod_{i=0}^{q} (G^{s_i})^{a_i}$. For the square span argument system, Z will simply output the parts of the CRS which are not already given.

2. The q-power Diffie-Hellman (q-**PDH**) assumption states that given $G, \hat{G}, G^s, \hat{G}^s, \ldots, \cancel{G^{s^{q+1}}}, \cancel{\hat{G}^{s^{q+1}}}, \ldots, G^{s^{2q}}, \hat{G}^{s^{2q}}$, it's difficult to compute $G^{s^{q+1}}$.

3. The q-target strong Diffie-Hellman (q-**TSDH**) states that given $G, \hat{G}, G^s, \hat{G}^s, \ldots, G^{s^q}, \hat{G}^{s^q}$, it's hard find an $r \in \mathbb{Z}_p$ and compute $e(G, \hat{G})^{\frac{1}{s-r}}$.

Finally, we are ready to construct the square span argument system. Let R be the relevant relation and λ the security parameter.

- $\mathsf{Setup}(1^\lambda)$:

  1. Sample the bilinear group generator to get $gk = (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(\lambda)$.
  2. Compute the boolean circuit $C_r : \{0,1\}_u^l \times \{0,1\}_w^l \to \{0,1\}$ which checks R, and generate a square span program $Q = (v_0, \ldots, v_m, t)$ which satisfies C_r over $\mathbb{Z}_p$ with degree d.
  3. Sample $G \xrightarrow{R} \mathbb{G}$, $\hat{G}, \tilde{G} \xrightarrow{R} \hat{\mathbb{G}}$, and $\beta, s \xrightarrow{R} \mathbb{Z}_p$ such that $t(s) \neq 0$.
  4. Return $\sigma = \left(Q, gk, \left\{G^{s^i}, \hat{G}^{s^i}\right\}_{i=1}^{d}, \left\{G^{\beta v_i(s)}\right\}_{i>l_u}, G^{\beta t(s)}, \tilde{G}, \tilde{G}^{\beta}\right)$ as the CRS.

- $\mathsf{Prove}(\sigma, u, w)$: Parse u as $(a_1, \ldots, a_{l_u}) \in \{0,1\}^{l_u}$ and use w to compute $a_{l_u+1}, \ldots, a_l$ such that $t(x)$ divides $(r(x))^2 - 1$, where $r(x) = v_0(x) + \sum_{i=1}^{m} a_i v_i(x)$. Sample $\delta \xrightarrow{R} \mathbb{Z}_p$, and let

$$r_u(x) = \sum_{i=1}^{l_u} a_i v_i(x),\ r_w(x) = \sum_{i>l_u}^{m} a_i v_i(x),\ q(x) = r_w(x) + \delta t(x),\ h(x) = \frac{(q(x))^2 - 1}{t(x)},$$

  Finally, return

$$\pi = (H, \hat{V}, V_w, B_w) = \left(G^{h(s)}, \hat{G}^{r(s)+\delta t(s)}, G^{q(s)}, G^{\beta q(s)}\right).$$

- $\mathsf{Verify}(\sigma, u, \pi)$: Compute $V = G^{r_u(s)} V_w = G^{r(s)+\delta t(s)}$ and check that

$$e(V, \hat{G}) = e(G, \hat{V}),\ e(H, \hat{G}^{t(s)}) = e(V, \hat{V}) e(G, \hat{G})^{-1},\ e(V_w, \tilde{G}^{\beta}) = e(B_w, \tilde{G}).$$

  Note that since the verifier does not have w, it does not know $a_{l_u+1}, \ldots, a_l$. However, it does have u, allowing it to compute $a_1, \ldots, a_{l_u}$.

Completeness follows by translating the three verification criterion into the exponent—i.e

$$r(s) + \delta t(s) = r(s) + \delta t(s),\ h(x)t(x) = (q(s))^2 - 1,\ q(s)\beta = \beta q(s).$$

for the first, second, and third check respectively. Informally, zero knowledge holds because δ is random and $t(s) \neq 0$, so $q(s)$ is random in $\mathbb{Z}_p$. But all of π can be efficiently computed from $q(s)$, since $r(s) = r_u(s) + q(s)$ and $r_u(s)$ is public. Therefore, the proof doesn't leak any information about the witness.

Finally, we can give a high level overview of the knowledge soundness. By d-PKE assumption, since the adversary was able to compute $V, \hat{V}$, it must know the coefficients of what's supposed to be $\log_G(V) = r(x) + \delta t(x)$ and therefore $r_w(x)$. $r_w(x)$ then allows us to calculate the witness w. The correctness of the exponent of V is enforced by the d-PDH and d-TSDH assumptions.

5 Preprocessing SNARKs from LIPs

[BCI+13] gives the following construction to combine an arbitrary LIP with linear-only encryption to obtain a designated-verifier preprocessing SNARK.

Definition 8. A **preprocessing SNARK** is a SNARK where the generator G may run in $\mathsf{poly}(\lambda, T)$ time, where λ is the security parameter, and T is the time bound.

Let $\{F_\lambda\}_{\lambda \in \mathbb{N}}$ be a field ensemble, $\mathcal{C} = \{C_\ell\}_{\ell \in \mathbb{N}}$ a family of circuits, and $(P_{\mathsf{LIP}}, V_{\mathsf{LIP}})$ an input-oblivious two-message LIP for a relation $\mathcal{R}_{\mathcal{C}}$, where for the field $\mathbb{F}_\lambda$, the verifier message is in $\mathbb{F}_\lambda^k$, the prover message is in $\mathbb{F}_\lambda^k$, and the knowledge error is $\varepsilon(\lambda)$. Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add}, \mathsf{ImVer})$ be a linear-only encryption scheme with plaintext field $\mathbb{F}_\lambda$, for security parameter λ. We now construct a preprocessing SNARK (G, P, V) for $\mathcal{R}_{\mathcal{C}}$ as follows.

- $G(1^\lambda, 1^\ell)$ uses the LIP query algorithm $Q_{\mathsf{LIP}}(\mathbb{F}_\lambda, 1^\ell)$ to generate a LIP message $\mathbf{q} \in \mathbb{F}_\lambda^m$ and a secret state $\mathbf{u} \in \mathbb{F}^m$, generates $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$, computes $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, q_i)$ for $i \in [m]$, and outputs $((\mathsf{pk}, c_1, \ldots, c_m), (\mathsf{sk}, \mathbf{u}))$.
- $P(\sigma, x, w)$ uses the LIP prover algorithm $P_{\mathsf{LIP}}(\mathbb{F}_\lambda, 1^\ell, x, w)$ to get a matrix $\Pi \in \mathbb{F}_\lambda^{k \times m}$ representing its message function, generates ciphertexts $c'_1, \ldots, c'_k$ encrypting $\Pi\mathbf{q}$ using Add, and outputs $(c'_1, \ldots, c'_k)$.
- $V(\tau, x, \pi)$ verifies that $\mathsf{ImVer}(\mathsf{sk}, c'_i) = 1$, for $i \in [k]$, defines $a_i := \mathsf{Dec}(\mathsf{sk}, c'_i)$, and outputs the decision of $D_{\mathsf{LIP}}(\mathbb{F}_\lambda, 1^\ell, x, \mathbf{u}, (a_1, \ldots, a_k))$.

This construction gives a preprocessing SNARK with knowledge error $\varepsilon(\lambda) + \mathsf{negl}(\lambda)$. Furthermore,

- $\mathsf{time}(G) = \mathsf{time}(Q_{\mathsf{LIP}}) + m \cdot \mathsf{poly}(\lambda)$,
- $\mathsf{time}(P) = \mathsf{time}(P_{\mathsf{LIP}}) + k^2 m \cdot \mathsf{poly}(\lambda)$,
- $\mathsf{time}(V) = \mathsf{time}(D_{\mathsf{LIP}}) + k \cdot \mathsf{poly}(\lambda)$,

- $|\sigma| = m \cdot \mathsf{poly}(\lambda)$, $|\tau| = \mathsf{poly}(\lambda) + m'$, and $|\pi| = k \cdot \mathsf{poly}(\lambda)$.

Definition 9. An encryption scheme has the **linear targeted malleability** property if for any polynomial-size adversary A and plaintext generator $\mathcal{M}$ there exists a polynomial-size simulator S such that

$$\left\{ \begin{array}{l|r} \mathsf{pk}, & (\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ a_1,\ldots,a_m & (s,a_1,\ldots,a_m) \leftarrow \mathcal{M}(\mathsf{pk}) \\ s & (c_1,\ldots,c_m) \leftarrow (\mathsf{Enc}(\mathsf{pk},a_1),\ldots,\mathsf{Enc}(\mathsf{pk},a_m)) \\ \mathsf{Dec}(\mathsf{sk},c'_1),\ldots,\mathsf{Dec}(\mathsf{sk},c'_k) & (c'_1,\ldots,c'_k) \leftarrow A(\mathsf{pk},c_1,\ldots,c_m;z) \\ & \text{where} \\ & \mathsf{ImVer}(\mathsf{sk},c'_1)=1,\ldots,\mathsf{ImVer}(\mathsf{sk},c'_k)=1 \end{array} \right\}$$

and

$$\left\{ \begin{array}{l|r} \mathsf{pk}, & (\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ a_1,\ldots,a_m & (s,a_1,\ldots,a_m) \leftarrow \mathcal{M}(\mathsf{pk}) \\ s & (\Pi,\mathbf{b}) \leftarrow S(\mathsf{pk};z) \\ a'_1,\ldots,a'_k & (a'_1,\ldots,a'_k)^\mathsf{T} \leftarrow \Pi \cdot (a_1,\ldots,a_m)^\mathsf{T} + \mathbf{b} \end{array} \right\},$$

where $\Pi \in \mathbb{F}^{k\times m}$, $\mathbf{b} \in \mathbb{F}^k$, and s is an arbitrary string, are computationally indistinguishable for any sufficiently large $\lambda \in \mathbb{N}$ and any auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$.

If the LIP $(P_{\mathsf{LIP}}, Q_{\mathsf{LIP}})$ has knowledge error $\varepsilon(\lambda)$ and $\mathcal{E}$ is an encryption scheme with linear targeted malleability, then the above construction is a designated-verifier **non-adaptive** preprocessing SNARK.

6 Related Work

One area we didn't get to cover is (not linear) PCPs. The construction of PCPs is long and difficult, with one of the "easier" proofs of the PCP theorem being [Din07]. Some proof systems are built on top of PCPs, one of which is Kilian's protocol [Kil92], which is succinct.

The papers we covered are also mostly theoretical in nature. There are some works concerned with the implementation and practical efficiency, one of which is Pinocchio [PHGR13]. The authors of Pinocchio implemented and benchmarked a verifiable computation toolchain, and they claim that verifiable computation is "almost practical".

There are also proof systems built on lattice based cryptography, which is conjectured to be post-quantum. One such work is [ISW21], a construction which greatly increased the efficiency of post-quantum zkSNARKs, especially in size.

7 Future Work

It is an open question whether preprocessing SNARK from LIP construction in section 5 maintains security against adaptively-chosen statements. There are no known attacks on the construction in the adaptive case, but no proofs of security are known either. Future work could include an investigation into this problem.

Another direction of study could concern implementations of various existing zk-SNARK constructions. Most constructions do not come with an implementation, and hence efficiency is difficult to compare. For example, Pinocchio is based on quadratic span programs rather than the simpler square span programs, but it's not obvious how to compare their efficiency.

Finally, we can always dream that one day, our cryptographic assumptions will be either proven or disproven, solving P = NP in the process.

References

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. 2013.

[DFGK14] George Danezis, Cedric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct nizk arguments. 2014.

[Din07] Irit Dinur. The pcp theorem by gap amplification. *J. ACM*, 54(3):12–es, jun 2007.

[ISW21] Yuval Ishai, Hang Su, and David J. Wu. Shorter and faster post-quantum designated-verifier zksnarks from lattices. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 212–234, New York, NY, USA, 2021. Association for Computing Machinery.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 723–732, New York, NY, USA, 1992. Association for Computing Machinery.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252, 2013.