17

Linear PCPs and Succinct Arguments

17.1 Overview: Interactive Arguments From "Long", Structured PCPs

"Short" PCPs for circuit satisfiability were covered in Section 9—by short, we mean the PCP proof has length quasilinear in the circuit size. Recall (Section 9.2) that Kilian [168] showed that short PCPs can be transformed into succinct arguments by first having the prover cryptographically commit to the PCP string π via Merkle-hashing. Then the argument-system verifier can simulate the PCP verifier, who queries a small number of symbols of π. The argument system prover can reveal these symbols of the committed PCP proof π, along with succinct authentication information to establish that the revealed symbols are indeed consistent with the committed string.

Unfortunately, short PCPs are quite complicated and remain impractical.[1] This section covers succinct arguments obtained via a similar methodology, but without resorting to complicated and impractical short PCPs.

[1] Though interactive analogs of PCPs called IOPs can achieve reasonable concrete performance (Section 10).

Arguments Without Short PCPs. Why is the prover inefficient if one instantiates Kilian's argument system [168] with a PCP of superpolynomial length $L = n^{\omega(1)}$? The problem is that $\mathcal{P}$ has to materialize the full proof π in order to compute its Merkle-hash and thereby commit to π. Materializing a proof of superpolynomial length clearly takes superpolynomial time.

But Ishai *et al.* [156] (IKO) showed that if π is highly structured in a manner made precise below, then there is a way for $\mathcal{P}$ to cryptographically commit to π *without* materializing all of it. This enables IKO to use structured PCPs of exponential length to obtain succinct interactive arguments. Such "long" PCPs turn out to be much simpler than short PCPs. The commitment protocol of [156] is based on any semantically secure, additively-homomorphic cryptosystem, such as ElGamal encryption [110]. Here, an additively-homomorphic encryption scheme is analogous to the notion of an additively-homomorphic commitment scheme (Section 12.3) which we exploited at length in Sections 13 and 14. It is an encryption scheme for which one can compute addition over encrypted data without decrypting the data.

Linear PCPs. The type of structure in the PCP proof that IKO [156] exploit is linearity. Specifically, in a *linear PCP*, the proof is interpreted as a function mapping $\mathbb{F}^v \to \mathbb{F}$ for some integer $v > 0$. A linear PCP is one in which the "honest" proof is (the evaluation table of) a linear function π. That is, π should satisfy that for any two queries $q_1, q_2 \in \mathbb{F}^v$ and constants $d_1, d_2 \in \mathbb{F}$, $\pi(d_1 q_1 + d_2 q_2) = d_1 \pi(q_1) + d_2 \pi(q_2)$. This is the same as requiring that π be a v-variate polynomial of total degree 1, with constant term equal to 0. Note that in a linear PCP, soundness should hold even against "cheating" proofs that are non-linear. So the only difference between a linear PCP and a PCP is that in a linear PCP, the honest proof is guaranteed to have special structure.

Putting it All Together. Summarizing the above, the arguments of IKO [156] conceptually proceed in the same two steps as Kilian's argument based on short PCPs. In the first step, the prover commits to the proof π, but unlike in Kilian's approach, here the prover can leverage the linearity of π to commit to it without ever materializing π in full. In the second step, the argument system verifier simulates

the linear PCP verifier, asking the prover to reveal certain evaluations of π, which the prover does using the reveal phase of the commitment protocol.

Hence, in order to give an efficient argument system based on linear PCPs, IKO [156] had to do two things. First, give a linear PCP for arithmetic circuit satisfiability. Second, give a commit/reveal protocol for linear functions.

In the results of IKO's seminal paper, there are downsides to both of the steps above. First, when applied to circuits of size S, IKO's linear PCP has length $|\mathbb{F}|^{O(S^2)}$, i.e., exponential in the *square* of the circuit size. To achieve practicality, this needs to be reduced to exponential in S itself rather than its square. Second, IKO's commit/reveal protocol for linear functions is *interactive*. This means that the arguments obtained thereof are also interactive. Moreover, the arguments cannot be rendered non-interactive via the Fiat-Shamir transformation because they are not public-coin. Interactivity, and associated lack of public verifiability, renders the arguments unusable in many practical scenarios.

Subsequent work addressed both of the downsides above. First, Gennaro *et al.* (GGPR) [125] give a linear PCP[2] of length $|\mathbb{F}|^{O(S)}$. Second, alternate transformations were given to turn linear PCPs into succinct *non-interactive* arguments [56], [125]. These transformations use pairing-based cryptography in a manner reminiscent of KZG commitments (Section 15.2). Applying these transformations to GGPR's linear PCP yields SNARKs, variants of which are widely used in practice today.[3]

Characteristics of the Resulting Arguments. A downside of long PCPs is that if the proof π has length L, then even writing down one of the verifier's queries to π requires $\log L$ field elements. If L is exponential in the circuit size $S = |\mathcal{C}|$, then even $\log(L)$ is *linear* in the circuit size. This means that even *specifying* the linear PCP verifier's queries takes time $\Omega(S)$. Compare this to the MIPs, PCPs, and IOPs from Sections 8–10, where the verifier's total runtime was $O(n + \text{polylog}(S))$, where

[2] The observation that GGPR's protocol is actually a linear PCP as defined by IKO was made in later work [56], [227].

[3] A variant of the SNARK due to Groth [142] is especially popular. See Section 17.5.6.

n is the size of the public input to $\mathcal{C}$. In the zk-SNARKs we obtain in this section, these long messages from the PCP verifier to specify its queries to π will translate into a long structured reference string, of length $\Omega(S)$, the generation of which produces "toxic waste" that could be used to forge "proofs" of false statements if it is not discarded.

This is analogous to how many SNARKs for circuit-satisfiability that use KZG-polynomial commitments (Section 15.2) require an SRS of size $\Omega(S)$. An important difference is that the SRS in KZG-based SNARKs depends only on the size S of the circuit under consideration, while the SRS arising in the SNARKs of this section is *circuit-specific*. If one tweaks the circuit under consideration even slightly, a new trusted setup must be run.

However, the communication in the reverse direction of the linear PCP, from prover to verifier, is very small (a constant number of field elements) and the verifier's online verification phase is especially fast as a consequence. This will ultimately lead to SNARKs with state-of-the-art proof length and verification costs.

Outline for the Section. Section 17.2 covers IKO's commit/reveal protocol for linear functions. Section 17.3 describes IKO's linear PCP of length $|\mathbb{F}|^{O(S^2)}$. Section 17.4 covers GGPR's linear PCP of length $|\mathbb{F}|^{O(S)}$. Section 17.5 covers transformations from linear PCPs to SNARKs via pairing-based cryptography.

17.2 Committing to a Linear PCP without Materializing It

Let π be a linear function $\mathbb{F}^v \to \mathbb{F}$. This section sketches the technique of IKO [156] for allowing the prover to first commit to π in a "commit phase" and then answer a series of k queries $q^{(1)}, \ldots, q^{(k)} \in \mathbb{F}^v$ in a "reveal phase". Roughly speaking, the security guarantee is that at the end of the commit phase, there is *some* function π' (which may not be linear) such that, if the verifier's checks in the protocol all pass and $\mathcal{P}$ cannot break the cryptosystem used in the protocol, then the prover's answers in the reveal phase are all consistent with π'.[4]

[4] This section actually sketches a refinement of IKO's commitment/reveal protocol, due to Setty *et al.* [228]. The original protocol of IKO guaranteed that for each

In more detail, the protocol uses a semantically secure homomorphic cryptosystem. Roughly speaking, semantic security guarantees that, given a ciphertext $c = \text{Enc}(m)$ of a plaintext m, any probabilistic polynomial time algorithm cannot "learn anything" about m. Semantic security is an analog of the hiding property of commitment schemes such as Pedersen commitments (Section 12.3). A cryptosystem is (additively) homomorphic if, for any pair of plain texts (m_1, m_2) and fixed constants $d_1, d_2 \in \mathbb{F}$, it is possible to efficiently compute the encryption of $d_1 m_1 + d_2 m_2$ from the encryptions of m_1 and m_2 individually. Here, in the context of linear PCPs, m_1 and m_2 will be elements of the field $\mathbb{F}$, so the expression $d_1 m_1 + d_2 m_2$ refers to addition and scalar multiplication over $\mathbb{F}$.

Many additively homomorphic encryption schemes are known, such as the popular ElGamal encyption scheme, whose security is based on the Decisional Diffie-Hellman assumption introduced in Section 15.1.

Commit Phase. In the commit phase, the verifier chooses a vector $r = (r_1, \ldots, r_v) \in \mathbb{F}^v$ at random, encrypts each entry of r and sends all v encryptions to the prover. Since π is linear, there is some vector $d = (d_1, \ldots, d_v) \in \mathbb{F}^v$ such that $\pi(q) = \sum_{i=1}^{v} d_i \cdot q_i = \langle d, q \rangle$ for all queries $q = (q_1, \ldots, q_v)$. Hence, using the homomorphism property of the encryption scheme, the prover can efficiently compute the encryption of $\pi(r)$ from the encryptions of the individual entries of r. Specifically, $\text{Enc}(\pi(r)) = \text{Enc}(\sum_{i=1}^{v} d_i r_i)$, and by the homomorphism property of Enc, this last expression can be efficiently computed from $\text{Enc}(r_1), \ldots, \text{Enc}(r_v)$. The prover sends this encryption to the verifier, who decrypts it to obtain (what is claimed to be) $s = \pi(r)$.

Remark 17.1. At the end of the commit phase, using the homomorphism property of Enc and the linearity of π, the honest prover has managed to send to the verifier an encryption of $\pi(r)$, even though the prover has no idea what r is (this is what semantic security of Enc

query i, there is a separate function π_i to which $\mathcal{P}$ was committed. Setty *et al.* [228] tweaked the protocol of IKO in a way that both reduced costs and guaranteed that $\mathcal{P}$ was committed to answering all k queries using a single function π' (which is possibly non-linear).

guarantees). Moreover, the prover has accomplished this in $O(v)$ time. This is far less than the $\Omega(|\mathbb{F}|^v)$ time required to evaluate π at all points, which would be required if the prover were to build a Merkle tree with the evaluations of π as the leaves.

One may wonder whether the use of an additively homomorphic *encryption* scheme Enc can be replaced with an additively homomorphic *commitment* scheme such as Pedersen commitments. Indeed, given a Pedersen commitment to each entry of r, the prover could compute a Pedersen commitment c^* to $\pi(r)$ using additive homomorphism, despite not knowing r, just as the prover in this section is able to compute $\text{Enc}(\pi(r))$ given $\text{Enc}(r_1), \ldots, \text{Enc}(r_v)$. The problem with this approach is that the verifier, who does not know π, would not be able to open c^* to $\pi(r)$. In contrast, by using an encryption scheme, the verifier can decrypt $\text{Enc}(\pi(r))$ to $\pi(r)$ despite not knowing π.

Reveal Phase. In the reveal phase, the verifier picks k field elements $\alpha_1, \ldots, \alpha_k \in \mathbb{F}$ at random, and keeps them secret. The verifier then sends the prover the queries $q^{(1)}, \ldots, q^{(k)}$ in the clear, as well as $q^* = r + \sum_{i=1}^{k} \alpha_i \cdot q^{(i)}$. The prover returns claimed answers $a^{(1)}, \ldots, a^{(k)}, a^* \in \mathbb{F}$, which are supposed to equal $\pi(q^{(1)}), \ldots, \pi(q^{(k)}), \pi(q^*)$. The verifier checks that $a^* = s + \sum_{i=1}^{k} \alpha_i \cdot a^{(i)}$, accepting the answers as valid if so, and rejecting otherwise.

Clearly the verifier's check will pass if the prover is honest. The proof of binding roughly argues that the only way for $\mathcal{P}$ to pass the verifier's checks, if $\mathcal{P}$ does not answer all queries using a single function, is to know the α_i's, in the sense that one can efficiently compute the α_i's given access to such a prover. But if the prover knows the α_i's, then the prover must be able to solve for r, since $\mathcal{V}$ reveals $q^* = r + \sum_{i=1}^{k} \alpha_i \cdot q^{(i)}$ to the prover. But this would contradict the semantic security of the underlying cryptosystem, which guarantees that the prover learned nothing about r from the encryptions of r's entries.

17.2.1 Detailed Presentation of Binding Property When $k = 1$

We present the main idea of the proof of binding, in the case that only one query is made in the reveal phase, i.e., $k = 1$. What does it mean

for the prover *not* to be bound to a fixed function after the commitment phase of the protocol? It means that there are at least two runs of the reveal protocol, where in the first run, the verifier sends queries q_1 and $q^* = r + \alpha \cdot q_1$, and the prover responds with answers a_1 and a^*, while in the second run the verifier sends queries q_1 and $\hat{q} = r + \alpha' \cdot q_1$, and the prover responds with answers $a'_1 \neq a_1$ and $\hat{a}$. That is, in two different runs of the reveal protocol, the prover responded to the same query q_1 with two different answers, and managed to pass the verifier's checks.

As indicated above, we will argue that in this case, the prover must know α and α'. But, as we now explain, this breaks the semantic security of the encryption scheme.

Why the prover knowing α and α' means semantic security is broken. Roughly speaking, this is because if the prover really learned nothing about r from $\text{Enc}(r_1), \ldots, \text{Enc}(r_v)$, as promised by semantic security of Enc, then it should be impossible for the prover to determine α with probability noticeably better than random guessing, even given q_1, $q^* = r + \alpha \cdot q_1$, and $\hat{q} = r + \alpha' \cdot q_1$. This is because, without knowing r, all that the equations

$$q^* = r + \alpha q_1 \tag{17.1}$$

and

$$\hat{q} = r + \alpha' \cdot q_1 \tag{17.2}$$

tell the prover about α and α' is that they are two field elements satisfying $q^* - \hat{q} = (\alpha - \alpha')q_1$. This is because, for every pair $\alpha, \alpha' \in \mathbb{F}$ satisfying Equations (17.1) and (17.2) for r, and any $c \in \mathbb{F}$, the pair $\alpha + c, \alpha' + c$ also satisfy both equations when r is replaced by $r - cq_1$. So without knowing anything about r, Equations (17.1) and (17.2) reveal no information whatsoever about α itself. Equivalently stated, if the prover knows α, then the prover must have learned something about r, in violation of semantic security of Enc.

Showing that the prover must know α and α'. Recall that s is the decryption of the value sent by the prover in the commit phase, which is claimed to be $\text{Enc}(\pi(r))$. Since the prover cannot decrypt, the

prover does not know s. Even so, if the verifier's checks in the two runs of the reveal phase pass, then the prover does know that:

$$a^* = s + \alpha a_1, \tag{17.3}$$

and

$$\hat{a} = s + \alpha' a_1. \tag{17.4}$$

Subtracting these two equations means that the prover knows that

$$(a^* - \hat{a}) = \alpha a_1 - \alpha' a_1', \tag{17.5}$$

Similarly, even though the prover doesn't know r, the prover does know that Equations (17.1) and (17.2) hold. Subtracting those two equations implies that $q^* - \hat{q} = (\alpha - \alpha')q_1$.

We may assume that none of the queries are the all-zeros vector, since any linear function π evaluates to 0 on the all-zeros vector. Hence, if we let j denote any nonzero coordinate of q_1, then:

$$q_j^* - \hat{q}_j = (\alpha - \alpha')q_{1,j}. \tag{17.6}$$

Since $a_1 \neq a_1'$, Equations (17.5) and (17.6) express α and α' via two linearly independent equations in two unknowns, and these have a unique solution. Hence, the prover can solve these two equations for α and α' as claimed.

17.3 A First Linear PCP for Arithmetic Circuit Satisfiability

Let $\{\mathcal{C}, x, y\}$ be an instance of arithmetic circuit satisfiability (see Section 6.5.1). For this section, we refer to a setting $W \in \mathbb{F}^S$ of values to each gate in $\mathcal{C}$ as a transcript for $\mathcal{C}$.

The linear PCP of this section is from IKO [156], and is based on the observation that W is a correct transcript if and only if W satisfies the following $\ell = S + |y| - |w|$ constraints (there is one constraint for every other non-output gate of $\mathcal{C}$, there are two constraints for each output gate of $\mathcal{C}$, and there are no constraints for any witness elements).

- For each input gate a, there is a constraint enforcing that $W_a - x_a = 0$. This effectively insists that the transcript W actually corresponds to the execution of $\mathcal{C}$ on input x, and not some other input.

- For each output gate a there is a constraint enforcing that $W_a - y_a = 0$. This effectively insists that the transcript W actually correspond to an execution of $\mathcal{C}$ that produces outputs y, and not some other set of outputs.
- If gate a is an addition gate with in-neighbors $\text{in}_1(a)$ and $\text{in}_2(a)$, there is a constraint enforcing that $W_a - \left(W_{\text{in}_1(a)} + W_{\text{in}_2(a)}\right) = 0$.
- If gate a is a multiplication, there is a constraint enforcing that $W_a - W_{\text{in}_1(a)} \cdot W_{\text{in}_2(a)} = 0$.

Together, the last two types of constraints insist that the transcript actually respects the operations performed by the gates of $\mathcal{C}$. That is, any addition (respectively, multiplication) gate actually computes the addition (respectively, product) of its two inputs. Note that the constraint for gate a of $\mathcal{C}$ is always of the form $Q_a(W) = 0$ for some polynomial Q_a of degree at most 2 in the entries of W.

For a transcript W for $\{\mathcal{C}, x, y\}$, let $W \otimes W$ denote the length-(S^2) vector whose (i,j)th entry is $W_i \cdot W_j$. Let $(W, W \otimes W)$ denote the vector of length $S + S^2$ obtained by concatenating W with $W \otimes W$. Let

$$f_{(W,W\otimes W)}(\cdot) \coloneqq \langle \cdot, (W, W \otimes W) \rangle.$$

That is, $f_{(W,W\otimes W)}$ is the linear function that takes as input a vector in $\mathbb{F}^{S+S^2}$ and outputs its inner product with $(W, W \otimes W)$. Consider a linear PCP proof π containing all evaluations of $f_{(W,W\otimes W)}$. π is typically called the *Hadamard encoding* of $(W, W \otimes W)$. Notice that π has length $|\mathbb{F}|^{S+S^2}$, which is enormous. However, $\mathcal{P}$ will never need to explicitly materialize all of π.

$\mathcal{V}$ needs to check three things. First, that π is a linear function. Second, assuming that π is a linear function, $\mathcal{V}$ needs to check that π is of the form $f_{(W,W\otimes W)}$ for some transcript W. Third, $\mathcal{V}$ must check that W satisfies all ℓ constraints described above.

First Check: Linearity Testing. Linearity testing is a considerably simpler task than the more general low-degree testing problems considered in the MIP of Section 8.2 (linearity testing is equivalent to testing

that an m-variate function equals polynomial of total degree 1 (with no constant term), while the low-degree testing problem considered in Section 8.2 tested whether an m-variate function is multilinear, which means its total degree can be as large as m).

Specifically, to perform linearity testing, the verifier picks two random points $q^{(1)}, q^{(2)} \in \mathbb{F}^{S+S^2}$ and checks that $\pi(q^{(1)} + q^{(2)}) = \pi(q^{(1)}) + \pi(q^{(2)})$, which requires three queries to π. If π is linear then the test always passes. Moreover, it is known that if the test passes with probability $1 - \delta$, then there is some linear function f_d such that π is δ-close to f_d [60], at least over fields of characteristic 2.[5]

Second Check. Assuming π is linear, π can be written as f_d for some vector $d \in \mathbb{F}^{S+S^2}$. To check that d is of the form $(W, W \otimes W)$ for some transcript W, $\mathcal{V}$ does the following.

- $\mathcal{V}$ picks $q^{(3)}, q^{(4)} \in \mathbb{F}^S$ at random.
- Let $(q^{(3)}, \mathbf{0})$ denote the vector in $\mathbb{F}^{S+S^2}$ whose first S entries equal $q^{(3)}$ and whose last S^2 entries are 0. Similarly, let $(\mathbf{0}, q^{(3)} \otimes q^{(4)})$ denote the vector whose first S entries equal 0, and whose last S^2 entries equal $q^{(3)} \otimes q^{(4)}$. $\mathcal{V}$ checks that $\pi(q^{(3)}, \mathbf{0}) \cdot \pi(q^{(4)}, \mathbf{0}) = \pi(\mathbf{0}, q^{(3)} \otimes q^{(4)})$. This requires three queries to π.

The check will pass if π is of the claimed form, since in this case

$$\pi(q^{(3)}, \mathbf{0}) \cdot \pi(q^{(4)}, \mathbf{0}) =$$

$$\left(\sum_{i=1}^{S} W_i q_i^{(3)}\right) \cdot \left(\sum_{j=1}^{S} W_j q_j^{(4)}\right) = \sum_{1 \le i,j \le S} W_i W_j q_i^{(3)} q_j^{(4)} = \langle q^{(3)} \otimes q^{(4)}, W \otimes W \rangle$$

$$= \pi(\mathbf{0}, q^{(3)} \otimes q^{(4)}).$$

If π is not of the claimed form, the test will fail with high probability over the choice of $q^{(3)}$ and $q^{(4)}$. This holds because $\pi(q^{(3)}, \mathbf{0}) \cdot \pi(q^{(4)}, \mathbf{0}) = f_d(q^{(3)}, \mathbf{0}) \cdot f_d(q^{(4)}, \mathbf{0})$ is a quadratic polynomial in the entries of $q^{(3)}$ and

[5] See [9, Theorem 19.9] for a short proof of this statement based on Discrete Fourier analysis. Over fields of characteristic other than 2, the known soundness guarantees of the linearity test are weaker. See [227, Proof of Lemma A.2] and [27, Theorem 1.1].

$q^{(4)}$, as is $f_d\left(\mathbf{0}, q^{(3)} \otimes q^{(4)}\right)$, and the Schwartz-Zippel lemma (Lemma 3.3) guarantees that any two distinct low-degree polynomials can agree on only a small fraction of points.

Third Check. Once $\mathcal{V}$ is convinced that $\pi = f_d$ for some d of the form $(W, W \otimes W)$, $\mathcal{V}$ is ready to check that W satisfies all ℓ constraints described above. This is the core of the linear PCP.

In order to check that $Q_i(W) = 0$ for all constraints i, it suffices for $\mathcal{V}$ to pick random values $\alpha_1, \ldots, \alpha_\ell \in \mathbb{F}$, and check that $\sum_{i=1}^{\ell} \alpha_i Q_i(W) = 0$. Indeed, this equality is always satisfied if $Q_i(W) = 0$ for all i; otherwise, $\sum_{i=1}^{\ell} \alpha_i Q_i(W)$ is a nonzero multilinear polynomial in the variables $(\alpha_1, \ldots, \alpha_\ell)$, and the Schwartz-Zippel lemma guarantees that this polynomial is nonzero at almost all points $(\alpha_1, \ldots, \alpha_\ell) \in \mathbb{F}^\ell$.

Notice that $\sum_{i=1}^{\ell} \alpha_i Q_i(W)$ is itself a degree-2 polynomial in the entries of W, which is to say that it is a linear combination of the entries of $(W, W \otimes W)$. Hence it can be evaluated with one additional query to π.

Soundness Analysis. A formal proof of the soundness of the linear PCP just described is a bit more involved than indicated above, but not terribly so. Roughly it proceeds as follows. If the prover passes the linearity test with probability $1-\delta$, then π is δ-close to a linear function f_d. Hence, as long as the 4 queries in the second and third checks are distributed uniformly in $\mathbb{F}^{S+S^2}$, then with probability at least $1 - 4 \cdot \delta$, the verifier will never encounter a point where π and f_d differ, and we can treat π as f_d for the remainder of the analysis. However, the queries in the second and third checks are not uniformly distributed in $\mathbb{F}^{S+S^2}$ as described. Nonetheless, they can be made uniformly distributed by replacing each query q with two random queries q' and q'' subject to the constraint that $q' + q'' = q$. This way, the marginal distributions of q' and q' are uniform over $\mathbb{F}^{S+S^2}$. And by linearity of f_d, it holds that $f_d(q)$ can be deduced to equal $f_d(q') + f_d(q'')$. With this change, the soundness analysis of the second and third steps are as indicated above.

Costs of the Argument System. One obtains an argument system by combining the above linear PCP with IKO's commitment/reveal protocol for linear functions (Section 17.3). The costs of this argument system are summarized in Table 17.1. Total communication from $\mathcal{V}$ to $\mathcal{P}$ is $\Theta(S^2)$, and hence $\mathcal{V}$ and $\mathcal{P}$'s runtime is also $\Theta(S^2)$. On the positive side, the communication in the reverse direction is just a constant number of field elements per input. Also, if $\mathcal{V}$ is simultaneously verifying $\mathcal{C}$'s execution over a large *batch* of inputs, then the $\Theta(S^2)$ communication and time costs can be amortized over the entire batch.

Such $\Theta(S^2)$ costs are very high, precluding practicality. Conceptually, the reason for the $\Theta(S^2)$ costs above is that the prover is forced to materialize the vector $W \otimes W$, whose (i,j)'th entry is $W_i \cdot W_j$. This is effectively forcing the prover to compute the product of every two gates i and j in the circuit, irrespective of the circuit's wiring pattern. That is, the prover must compute the product of every pair of gates in the circuit, irrespective of whether those two gates are actually multiplied together by another gate in the circuit. Then the third check in the linear PCP above effectively ignores almost all of those S^2 products, checking the validity only of the at most S products that actually correspond to multiplication gates in the circuit.

Section 17.4 below explains how to "cut out" the unnecessary products above, reducing the $\Theta(S^2)$ costs to $\Theta(S)$.

Table 17.1: Costs of the argument system from Section 17.3 for arithmetic circuit satisfiability when run on a circuit $\mathcal{C}$ of size S. Note that the verifier's cost and the communication cost can be amortized when simultaneously outsourcing $\mathcal{C}$'s execution on a large *batch* of inputs. The stated bound on $\mathcal{P}$'s time assumes $\mathcal{P}$ knows a witness w for $\mathcal{C}$.

$\mathcal{V} \to \mathcal{P}$ **Communication**	$\mathcal{P} \to \mathcal{V}$ **Communication**	$\mathcal{V}$ **Time**	$\mathcal{P}$ **Time**
$O(S^2)$ field elements	$O(1)$ field elements	$O(S^2)$	$O(S^2)$

17.4 GGPR: A Linear PCP of Size $O(|\mathbb{F}|^S)$ for Circuit-SAT and R1CS

In a breakthrough result, Gennaro *et al.* [125] gave a linear PCP of length $O(|\mathbb{F}|^S)$ for arithmetic circuit satisfiability, where S denotes the size of the circuit.[6] In fact, their linear PCP also solves the more general problem of R1CS satisfiability (see Section 8.4 for a detailed introduction to R1CS and how to transform arithmetic circuit satisfiability instances into R1CS-satisfiability instances).[7, 8] In this section, we choose to present the linear PCPs of [125] in the context of R1CS rather than arithmetic circuits because we feel this leads to a clearer description of the protocol, and is more general. The linear PCPs of [125] have been highly influential, and form the foundation of many of the implementations of argument systems.

Recap of R1CS. For the reader's convenience, we briefly recall the definition of R1CS. An R1CS instance over field $\mathbb{F}$ is of the form

$$Az \circ Bz = Cz, \tag{17.7}$$

where A, B, and C are public matrices in $\mathbb{F}^{\ell \times S}$. Here, $\circ$ denotes entrywise-wise product. An R1CS instance is satisfiable if there is some vector $z \in \mathbb{F}^S$ satisfying Equation (17.7). Note that z can be thought of as the R1CS-analog of the circuit transcript W appearing in Section 17.3.

Equivalently, if $a_i, b_i, c_i \in \mathbb{F}^S$ respectively denote the ith rows of A, B, and C, then an R1CS instance consists of ℓ *constraints*, with the ith constraint of the form

$$\langle a_i, z \rangle \cdot \langle b_i, z \rangle - \langle c_i, z \rangle = 0. \tag{17.8}$$

The linear PCPs of [125] crucially exploit that the left hand side of Equation (17.8) can be evaluated in constant time given three linear

[6] The argument system of Gennaro *et al.* can be understood in multiple ways, and [125] did not present it within the framework of linear PCPs. Subsequent work [56], [227] identified the protocol of Gennaro *et al.* as an example of a linear PCP.

[7] Gennaro *et al.* referred to R1CS satisfiability problems as *Quadratic Arithmetic Programs* (QAPs).

[8] IKO's linear PCP covered in Section 17.3 also applied to R1CS satisfiability, though we only presented it in the context of circuit satisfiability.

functions evaluated at z, namely $\langle a_i, z\rangle$, $\langle b_i, z\rangle$, and $\langle c_i, z\rangle$. This is a stronger notion of structure than was exploited Section 17.3. Section 17.3 only exploited that each circuit gate corresponds to a constraint involving a polynomial (in the entries of the circuit transcript W) of total degree at most 2. This meant that the constraint is a linear function of the entries of the vector $W \otimes W$, but this vector has length S^2, leading to quadratic costs in the linear PCP of that section.

The linear PCP. Our ultimate goal is to associate any vector $z \in \mathbb{F}^S$ with a univariate polynomial $g_z(t)$ that vanishes on H if and only if z satisfies the R1CS instance (Equation (17.7)). To define g_z, we must first define several constituent polynomials that together capture the R1CS matrices.

Polynomials capturing matrix columns. Let $H \coloneqq \{\sigma_1, \ldots, \sigma_\ell\}$ be a set of ℓ arbitrary distinct elements in $\mathbb{F}$, one for each constraint in the R1CS instance. For each $j \in \{1, \ldots, S\}$, we define three univariate polynomials $\mathcal{A}_j$, $\mathcal{B}_j$, and $\mathcal{C}_j$, meant to "capture" the j'th column of A, B, and C respectively. Specifically, for each $j \in \{1, \ldots, S\}$, define three degree-$(\ell - 1)$ polynomials via interpolation as follows.

$$\begin{aligned}
\mathcal{A}_j(\sigma_i) &= A_{i,j} \text{ for all } i \in \{1, \ldots, \ell\}.\\
\mathcal{B}_j(\sigma_i) &= B_{i,j} \text{ for all } i \in \{1, \ldots, \ell\}.\\
\mathcal{C}_j(\sigma_i) &= C_{i,j} \text{ for all } i \in \{1, \ldots, \ell\}.
\end{aligned}$$

Turning z into a polynomial that vanishes on H if and only if z is a satisfying assignment. Let $g_z(t)$ denote the following univariate polynomial:

$$\left(\sum_{\text{columns } j \in \{1,\ldots,S\}} z_j \cdot \mathcal{A}_j(t)\right) \cdot \left(\sum_{\text{columns } j \in \{1,\ldots,S\}} z_j \cdot \mathcal{B}_j(t)\right) - \left(\sum_{\text{columns } j \in \{1,\ldots,S\}} z_j \cdot \mathcal{C}_j(t)\right). \tag{17.9}$$

By design, g_z vanishes on H if and only all R1CS constraints are satisfied, i.e., if and only if z is a satisfying assignment for the R1CS instance. Indeed,

$$g_z(\sigma_i) = \left(\sum_{\text{columns } j} z_j \cdot A_{i,j}\right) \cdot \left(\sum_{\text{columns } j} z_j \cdot B_{i,j}\right) - \left(\sum_{\text{columns } j} z_j \cdot C_{i,j}\right) = \langle a_i, z\rangle \cdot \langle b_i, z\rangle - \langle c_i, z\rangle,$$

where a_i, b_i, and c_i the ith rows of A, B, and C respectively (see Equation (17.8)). Hence $g_z(\sigma_i) = 0$ if and only if the ith constraint in the R1CS instance is satisfied by z.

Checking Whether g_z Vanishes on H. To check whether g_z vanishes on H, we rely on Lemma 9.3, which also played a key role in our constructions of efficient PCPs and IOPs (Sections 9 and 10) and is restated here for the reader's convenience.

Lemma 17.1. (Ben-Sasson and Sudan [50]) Let $\mathbb{F}$ be a field and $H \subseteq \mathbb{F}$. For $d \geq |H|$, a degree-d univariate polynomial g over $\mathbb{F}$ vanishes on H if and only if the polynomial $\mathbb{Z}_H(t) \coloneqq \prod_{\alpha \in H}(t - \alpha)$ divides g, i.e., if and only if there exists a polynomial h^* with $\deg(h^*) \leq d - |H|$ such that $g = \mathbb{Z}_H \cdot h^*$.

By inspection, the degree of the polynomial g_z is at most $d = 2(\ell - 1)$, where ℓ is the number of constraints. By Lemma 9.3, to convince $\mathcal{V}$ that g_z vanishes on H, the proof merely needs to convince $\mathcal{V}$ that $g_z = \mathbb{Z}_H \cdot h^*$ for some polynomial h^* of degree $d - |H| = \ell - 1$. To be convinced of this, $\mathcal{V}$ can pick a random point $r \in \mathbb{F}$ and check that

$$g_z(r) = \mathbb{Z}_H(r) \cdot h^*(r). \tag{17.10}$$

Indeed, because any two distinct degree $(\ell - 1)$ polynomials can agree on at most $\ell - 1$ points, if $g_z \neq \mathbb{Z}_H \cdot h^*$, then this equality will fail with probability at least $1 - (\ell - 1)/|\mathbb{F}|$.

To this end, a correct proof represents two linear functions. The first is $f_{\text{coeff}(h^*)}$, where $\text{coeff}(h^*)$ denotes the vector of coefficients of h^*

(recall that f_v for any vector $v \in \mathbb{F}^\ell$ denotes the ℓ-variate linear function $f_v(x) := \langle v, x \rangle$). The second is f_z. Note that

$$f_{\text{coeff}(h^*)}(1, r, r^2, \ldots, r^{\ell-1}) = h^*(r), \tag{17.11}$$

so $\mathcal{V}$ can evaluate $h^*(r)$ with a single query to the proof. Similarly, $\mathcal{V}$ can evaluate g_z at r by evaluating f_z at the three vectors $(\mathcal{A}_1(r), \ldots, \mathcal{A}_S(r))$, $(\mathcal{B}_1(r), \ldots, \mathcal{B}_S(r))$, and $(\mathcal{C}_1(r), \ldots, \mathcal{C}_S(r))$.

Remark 17.2. In applications, there will in fact be a public input $x \in \mathbb{F}^n$ to the R1CS instance, and it will be required that the satisfying vector z have $z_i = x_i$ for $i = 1, \ldots, n$ (and these requirements are not otherwise included in the R1CS constraints). One can easily modify the linear PCP to enforce this by letting $z' = (z_{n+1}, \ldots, z_S)$ and replacing the linear function f_z in the proof with $f_{z'}$. Whenever the linear PCP verifier wants to query f_z at some vector $q = (q'', q') \in \mathbb{F}^n \times \mathbb{F}^{S-n}$, note that $f_z(q) = \left(\sum_{j=1}^n q''_j \cdot x_j\right) + f_{z'}(q')$. Hence, the the verifier can query the proof $f_{z'}$ at q' to obtain $v = f_{z'}(q')$ and set $f_z(q) = \left(\sum_{j=1}^n q''_j \cdot x_j\right) + v$.

Just as in the linear PCP of Section 17.3, the verifier also has to perform linearity testing on $f_{\text{coeff}(h^*)}$ and f_z. The verifier must also replace the four queries described above with two queries each to ensure that all queries are uniformly distributed. These complications arise because we required that a linear PCP be sound against proofs that are *non-linear* functions of queries. We remark that for purposes of the non-interactive argument system of the next section (Section 17.5), the linear PCP verifier need not perform linearity testing nor ensure that any of its queries are uniformly distributed. This is because the cryptographic techniques in that section bind the argument system prover to *linear functions*, so the underlying information-theoretic protocol does not need to bother testing whether the function is in fact linear. In contrast, the cryptographic techniques of Section 17.2 only bind the prover to some function which was not necessarily linear, hence the need for the underlying linear PCP to be sound against proofs that are non-linear functions.

Argument System Costs. One can obtain an argument system by combining the linear PCP above with the commitment protocol for

linear functions of Section 17.2. This argument system is not currently used in practice; one reason for this is that it is interactive. Still, it is instructive to examine the costs of the resulting argument system. They next section (Section 17.5) will provide a different transformation of the linear PCP above into a succinct argument that addresses the downsides of Section 17.2 while essentially preserving the costs.

The costs of the argument system are summarized in Table 17.2. The honest prover $\mathcal{P}$ needs to perform the following steps, assuming $\mathcal{P}$ knows a satisfying assignment z for the R1CS instance. First, compute the polynomial $g_z(t)$. Second, divide g_z by $\mathbb{Z}_H$ to find the quotient polynomial h^*. Third, run the linear commitment/reveal protocol described in Section 17.2, to commit to $f_{\text{coeff}(h^*)}$ and f_z and answer the verifier's queries.

For simplicity, let us assume that the number of R1CS constraints (matrix rows) ℓ is less than or equal to the number of columns S, and that $O(S)$ many entries of the matrices A, B, and C are non-zero—this is the case for R1CS instances resulting from circuit-satisfiability instances of size S, see Section 8.4. The third step can clearly be done in time $O(S)$.[9] The first step, of computing g_z, can be done in time $O(S \log^2 S)$ using standard multipoint interpolation algorithms based on the Fast Fourier Transform (FFT).[10] The second step can be done in time $O(S \log S)$ using FFT-based polynomial division algorithms.

Total communication from $\mathcal{V}$ to $\mathcal{P}$ is $\Theta(S)$ as well, but the communication in the reverse direction is just a constant number of field elements per input. Because the communication from $\mathcal{V}$ to $\mathcal{P}$ is so

[9] More precisely, this step requires taking a linear combination of $O(S)$ ciphertexts of a homomorphic encryption scheme.

[10] In somewhat more detail, the polynomial g_z can be expressed as $\mathcal{A} \cdot \mathcal{B} - \mathcal{C}$ where $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ are degree-$(\ell - 1)$ polynomials whose coefficients over the Lagrange basis corresponding to the set H can be computed in time proportional to the number of nonzero entries of each matrix. Given these coefficients, fast multi-point interpolation algorithms can evaluate g_z at any desired set $H' \subseteq \mathbb{F}$ of size ℓ in $\mathbb{F}$ in time $O(S \log^2 S)$. Since g_z has degree $2(\ell - 1)$, if H' is disjoint from H, then the 2ℓ evaluations of g_z at points in $H \cup H'$ uniquely specify g_z. In fact, for all $\sigma \in H'$, $h^*(\sigma)$ can be computed directly from $g_z(\sigma)$, via: $h^*(\sigma) = g_z(\sigma) \cdot \mathbb{Z}_H(\sigma)^{-1}$. Since h^* has degree at most $\ell - 1$, these evaluations uniquely specify h^*. The coefficients of h^* in the standard monomial basis can be computed in $O(S \log S)$ time using FFT-based algorithms.

Table 17.2: Costs of the argument system from Section 17.4 when run on a circuit satisfiability instance $\{\mathcal{C}, x, w\}$ of size S, or R1CS instance with $O(S)$ nonzero entries in each of the matrices A, B, C. The $\tilde{O}$ notation hides polylogarithmic factors in S. Note that the verifier's cost and the communication cost can be amortized when outsourcing the circuit $\mathcal{C}$'s execution on a *batch* of inputs. The stated bound on $\mathcal{P}$'s time assumes $\mathcal{P}$ knows a witness w for $\mathcal{C}$ or a solution vector z for the R1CS instance.

$\mathcal{V} \to \mathcal{P}$ **Communication**	$\mathcal{P} \to \mathcal{V}$ **Communication**	$\mathcal{V}$ **Time**	$\mathcal{P}$ **Time**
$O(S)$ field elements	$O(1)$ field elements	$\tilde{O}(S)$	$\tilde{O}(S)$

large, i.e., $\Theta(S)$, $\mathcal{V}$'s runtime is also $\Theta(S)$. So this argument system in uninteresting if applied to a *single input* to $\mathcal{C}$: the verifier would be just as fast in the trivial protocol in which the prover sends a witness w to the verifier, and the verifier checks on her own that $\mathcal{C}(x, w) = y$. However, if $\mathcal{V}$ is simultaneously verifying $\mathcal{C}$'s execution over a large batch of inputs, then the $\Theta(S)$ cost for $\mathcal{V}$ can be amortized over the entire batch.

Note that the verifier's check does require $\mathcal{V}$ to evaluate $\mathbb{Z}_H(r)$, where $\mathbb{Z}_H(X) = \prod_{\sigma \in H}(X - \sigma)$ is the vanishing polynomial of H. Since the verifier requires $O(S)$ time just to specify the linear PCP queries, $\mathcal{V}$'s asymptotic time bound in the linear PCP is not affected by computing $\mathbb{Z}_H(r)$ directly via S subtractions and multiplications. Nonetheless, H could be chosen carefully to ensure that $\mathbb{Z}_H(x)$ is sparse, thereby enabling $\mathbb{Z}_H(r)$ to be evaluated in $o(S)$ time. For example, if H is chosen to be a multiplicative subgroup of $\mathbb{F}$ of order ℓ (the setting considered in Section 10.3, see Equation (10.1)), then $\mathbb{Z}_H(X) = X^\ell - 1$. Clearly, then $\mathbb{Z}_H(r)$ can be evaluated with $O(\log n)$ field multiplications.

In the non-interactive argument presented in the next section, $\mathbb{Z}_H(r)$ can simply be provided to the verifier as part of the trusted setup procedure, which takes time $O(S)$ regardless of whether or not H is chosen to ensure $\mathbb{Z}_H$ is a sparse polynomial.

17.5 Non-Interactivity and Public Verifiability

17.5.1 Informal Overview

We have already seen (Section 17.2) how to convert the linear PCP of the previous section to a succinct interactive argument using any additively-homomorphic encryption scheme. We cannot apply the Fiat-Shamir transformation to render this argument system non-interactive because it is not public coin—the argument system makes use of an additively homomorphic encryption scheme for which the verifier chooses the private key, and if the prover learns the private key it can break soundness of the argument system.

Instead, linear PCPs can be converted to non-interactive arguments using pairing-based techniques extremely similar to the KZG polynomial commitment scheme of Section 15.2. The idea is as follows (we simplify slightly in this informal overview, deferring a complete description of the protocol until Section 17.5.3).[11] Rather than having the verifier send the linear PCP queries to the prover "in the clear" as in the interactive argument of Section 17.2, the entries of the linear PCP queries $q^{(1)}, \ldots, q^{(k)}$, will be *encoded* in the exponent of a group generator g for pairing-friendly group $\mathbb{G}$, and the encodings provided to the prover via inclusion in a structured reference string. The argument system then exploits the additive homomorphism of the encoding, i.e., that the encoding g^{x+y} of $x + y \in \mathbb{F}_p$ equals the product of the encodings $\mathsf{g}^x, \mathsf{g}^y$ of x and y individually, so long as $|\mathbb{G}| = p$.[12] If $\pi(x) = \sum_j c_j x_j$ denotes a linear PCP proof, the additive homomorphism allows the prover to evaluate the encodings of $\pi(q^{(1)}), \ldots, \pi(q^{(k)})$ and send them to the verifier. Finally, the argument system verifier accepts if and only if the linear PCP verifier would accept the responses $\pi(q^{(1)}), \ldots, \pi(q^{(k)})$. Since the argument system prover did not send $\pi(q^{(1)}), \ldots, \pi(q^{(k)})$ in

[11] In this section, we use the serif font g rather than g to denote a generator of a pairing-friendly group $\mathbb{G}$, to distinguish the group generator from the polynomial $g_{x,y,W}$ defined in the previous section.

[12] The encoding g^x of x is an (unblinded) Pedersen commitment to x (Section 12.3). But in the SNARK of this section, neither the prover nor the verifier can open these "commitments", i.e., the exponents of the group elements in the structured reference string are "computationally hidden" from both prover and verifier. This is why we refer to an SRS entry g^x as an encoding of x rather than a commitment to x.

the clear, but rather *encodings* $\mathsf{g}^{\pi(q^{(1)})}, \ldots, \mathsf{g}^{\pi(q^{(k)})}$, it is not immediately obvious how the argument system verifier can make this determination. This is where pairings come in.

Observe that the verifier's check in the linear PCP is a total-degree-2 function of the responses to the PCP queries. Indeed, recall that the linear PCP verifier checks that $g_z(r) = \mathbb{Z}_H(r) \cdot h^*(r)$. Letting

$$q^{(1)} = (A_1(r), \ldots, A_S(r)),$$

$$q^{(2)} = (B_1(r), \ldots, B_S(r)),$$

and

$$q^{(3)} = (C_1(r), \ldots, C_S(r)),$$

then

$$g_z(r) = f_z(q^{(1)}) \cdot f_z(q^{(2)}) - f_z(q^{(3)}),$$

which is clearly a function of total degree two in the linear PCP prover responses $f_z(q^{(1)})$, $f_z(q^{(2)})$, and $f_z(q^{(3)})$, with a single multiplication operation. Similarly, letting $q^{(4)} = (1, r, \ldots, r^S)$, the right hand side of the verifier's check is a linear (i.e., total degree 1) function of the linear PCP prover response $f_{\text{coeff}(h^*)}(q^{(4)})$.

Recall from Section 15.1 that the entire point of pairings is that they allow for a single "multiplication check" to be performed on encoded values, without the need to decode the values. This enables the argument system verifier to perform the linear PCP verifier's check "in the exponent". That is, if the argument system prover responds to the ith query with g^{v_i}, the verifier can use the bilinear map associated with $\mathbb{G}$ to check whether the PCP verifier would have accepted if the PCP prover had answered query $q^{(i)}$ with value v_i.

17.5.2 A Complication: Linear Interactive Proofs vs. Linear PCPs

The argument system sketched above runs into the following complication. While (under appropriate Knowledge of Exponent assumptions) the pairing-based cryptography forces the argument system prover to answer each encoded linear PCP query in a manner consistent with a linear function, it does not ensure that all queries are answered with *the*

same linear function.[13] That is, for the argument system to be sound, we really need the underlying linear PCP to be sound against provers that use a different linear function to answer each query.[14] Such a linear PCP is called a (2-message) *linear interactive proof* (LIP) [56].

Bitansky *et al.* [56] give a simple and efficient method for translating any linear PCP into a LIP. Specifically, if soundness of the linear PCP requires that queries $q^{(1)}, \ldots, q^{(k')}$ be answered with the same linear function, the LIP verifier simply adds an extra query $q^{(k+1)} = \sum_{i=1}^{k'} \beta_i q^{(i)}$ to the linear PCP, where $\beta_1, \ldots, \beta_{k'}$ are randomly chosen field elements known only to the verifier. That is, $q^{(k+1)}$ is a random linear combination of the relevant linear PCP queries. The LIP verifier checks that the answer a_{k+1} to the $(k+1)$'st query equals $\sum_{i=1}^{k'} \beta_i a_i$, and if so, feeds answers $a_1, \ldots, a_k$ to the linear PCP verifier. It can be shown that if the linear PCP is complete and knowledge-sound, then the resulting LIP is as well. We omit the proof of this fact, but the idea is to argue that if the LIP prover does not answer all $k'+1$ queries $q^{(1)}, \ldots, q^{(k')}, q^{(k+1)}$ using the same linear function for each query, then there is some nonzero linear function π such that the prover will pass the LIP verifier's final check only if $\pi(\beta_1, \ldots, \beta_{k'}) = 0$. Since $\beta_1, \ldots, \beta_{k'}$ are chosen uniformly at random from $\mathbb{F}$, the Schwartz-Zippel lemma (Lemma 3.3) implies that this occurs with probability at most $1/|\mathbb{F}|$.

17.5.3 Complete Description of the SNARK

Here is the entire non-interactive argument system. Recall that $q^{(1)}, \ldots, q^{(4)}$ were defined in Section 17.5.1. Accounting for the transformation from a linear PCP to an LIP of Section 17.5.2, we define a 5th query vector $q^{(5)} := \sum_{i=1}^{3} \beta_i q^{(i)}$, where $\beta_1, \ldots, \beta_3$ are randomly chosen elements of $\mathbb{F}_p$. We do not include the 4th query in this random linear combination because soundness of the linear PCP from Section 17.4 only requires that the first 3 queries be answered with the same linear

[13] In fact, the cryptography does not prevent the prover from answering the ith (encoded) query $q^{(i)}$ with (an encoding of) a linear combination of entries of *all of the queries* $q^{(1)}, \ldots, q^{(3)}$.

[14] More precisely, owing to Footnote 13, the linear PCP needs to be secure against provers that answer each of the four queries with different linear function of all four queries.

function f_z, and completeness in fact requires that the 4th query be answered with a different linear function, namely $f_{\text{coeff}(h^*)}$.

For every entry $q_j^{(i)}$ of each of the five LIP queries $q^{(1)}, \ldots, q^{(5)}$, the SRS contains the pair $(\mathsf{g}^{q_j^{(i)}}, \mathsf{g}^{\alpha q_j^{(i)}})$ where α is chosen at random from $\{1, \ldots, p-1\}$. The verification key (i.e., the information provided to the verifier by the trusted setup procedure) contains the quantities g, g^{α}, $\mathsf{g}^{\mathbb{Z}_H(r)}$, g^{β_1}, g^{β_2}, g^{β_3}. Note that all quantities in the SRS can be computed during the setup phase because they depend only on the R1CS matrices, not on the witness vector z.

Using the SRS and additive homomorphism, the prover computes and sends to the verifier five pairs of group elements $(\mathsf{g}_1, \mathsf{g}_1'), \ldots,$ $(\mathsf{g}_4, \mathsf{g}_4'), (\mathsf{g}_5, \mathsf{g}_5')$ claimed to equal

$$(\mathsf{g}^{f_z(q^{(1)})}, \mathsf{g}^{\alpha \cdot f_z(q^{(1)})}),$$

$$(\mathsf{g}^{f_z(q^{(2)})}, \mathsf{g}^{\alpha \cdot f_z(q^{(2)})}),$$

$$(\mathsf{g}^{f_z(q^{(3)})}, \mathsf{g}^{\alpha \cdot f_z(q^{(3)})}),$$

$$(\mathsf{g}^{f_{\text{coeff}(h^*)}(q^{(4)})}, \mathsf{g}^{\alpha \cdot f_{\text{coeff}(h^*)}(q^{(4)})}),$$

and

$$(\mathsf{g}^{f_z(q^{(5)})}, \mathsf{g}^{\alpha \cdot f_z(q^{(5)})}).$$

The verifier performs the following checks. First, it checks that

$$e\left(\mathsf{g}_1, \mathsf{g}_2\right) = e\left(\mathsf{g}_3, \mathsf{g}\right) \cdot e\left(\mathsf{g}^{\mathbb{Z}_H(r)}, \mathsf{g}_4\right). \tag{17.12}$$

Second, it checks that

$$\prod_{i=1}^{3} e(\mathsf{g}^{\beta_i}, \mathsf{g}_i) = e(\mathsf{g}_5, \mathsf{g}). \tag{17.13}$$

Third, for each of the five pairs $(\mathsf{g}_i, \mathsf{g}_i')$ for $i = 1, \ldots, 5$, the verifier checks that

$$e(\mathsf{g}_i, \mathsf{g}^{\alpha}) = e(\mathsf{g}, \mathsf{g}_i'). \tag{17.14}$$

17.5.4 Establishing Completeness and Knowledge-Soundness

Completeness of the SNARK holds by design. Indeed, by bilinearity of e, the first check of the verifier (Equation (17.12)) is specifically designed to pass if $g_{x,y,W}(r) = \mathbb{Z}_H(r) \cdot h^*(r)$ and the prover returns the prescribed proof elements. The second check (Equation (17.13)) will pass if and only if $\mathsf{g}_5 = \prod_{i=1}^{3} \mathsf{g}_i^{\beta_i}$, which will be the case if the prover behaves as prescribed. Similarly, the final set of checks (Equation (17.14)) will pass if indeed $\mathsf{g}_i' = \mathsf{g}_i^{\alpha}$ for all i.

The proof of knowledge-soundness relies on the following two cryptographic assumptions. These are mild variants of the two assumptions (PKoE and SDH) that we relied upon for the pairing-based polynomial commitment of Section 15.2.

Knowledge of Exponent Assumption (KEA). This is a Variant of the PKoE assumption. Recall that the SRS for the SNARK of this section consists of $t = O(S)$ many pairs of the form $(\mathsf{g}_i, \mathsf{g}_i^{\alpha})$ for $i = 1, \ldots, t$. The Knowledge of Exponent assumption essentially guarantees that for any polynomial-time algorithm that is given such an SRS as input and is capable of outputting pairs (f, f') such that $f' = f^{\alpha}$, there is an efficient extractor algorithm that outputs coefficients $c_1, \ldots, c_t$ explaining (f, f'), in the sense that $f = \prod_{i=1}^{t} \mathsf{g}_i^{c_i}$. See Section 15.2 for discussion of the intuition behind such an assumption and why it is reasonable.

Poly-Power Discrete Logarithm is Hard. This assumption posits that, if r is chosen at random from $\mathbb{F}_p$, then any polynomial time algorithm, when given as input the encodings of $t \leq \text{poly}(S)$ many powers of r (i.e., g, g^r, g^{r^2}, $\ldots$, g^{r^t}), is incapable of outputting r except with negligible probability.

Informally, the final set of five checks the SNARK verifier performs (Equation (17.14)) guarantees by KEA that the SNARK prover answers all of the LIP queries using linear functions, and in fact the prover "knows" these linear functions. To clarify, since in the SNARK the LIP queries are encoded in the exponent of g, the SNARK prover is applying the linear function to the exponents, by taking products of constant powers of the encoded query entries in the SRS.

The remaining two checks that the SNARK verifier performs ensures that these linear functions would convince the verifier to accept in the LIP obtained by applying the transformation of Section 17.5.2 to the linear PCP of Section 17.4. Knowledge-soundness of the SNARK then follows from that of the LIP.

In more detail, the analysis establishing that the SNARK is knowledge sound shows how to transform any argument system prover that convinces the argument system verifier to accept with non-negligible probability into either a witness vector z for the R1CS instance, or a polynomial time algorithm $\mathcal{A}$ that breaks the poly-power discrete logarithm assumption. Because the SNARK prover passes the final set of 5 checks performed by the verifier (Equation (17.14)) with non-negligible probability, the KEA implies that there is an efficient extractor $\mathcal{E}$ outputting linear functions $\pi_1, \ldots, \pi_5 \colon \mathbb{F}^t \to \mathbb{F}$ that "explain" the query responses as linear combinations (in the exponent) of SRS elements. That is, for $i = 1, \ldots, 5$, if the SRS σ consists of pairs of group elements (f_j, f_j^{α}) for $j = 1, \ldots, |\sigma|$, let $c_{i,1}, \ldots, c_{i,|\sigma|}$ denote the coefficients of π_i. Then for $i \in \{1, 2, 3, 4, 5\}$,

$$\mathsf{g}_i = \prod_{j=1}^{|\sigma|} f_j^{c_{i,j}}.$$

For notational convenience, let us write π_1 as f_z, and π_4 as $f_{\text{coeff}(h^*)}$. Let g_z and h^* be the polynomials implied by z and h^* via Equations (17.9) and (17.11). The argument system's verifier's first and second checks ensure that these linear functions convince the LIP verifier to accept with non-negligible probability. In particular, the LIP soundness analysis then implies that $\pi_1 = \pi_2 = \pi_3 = f_z$ and hence that $g_z(r) = \mathbb{Z}_H(r) \cdot h^*(r)$.

If $g_z = \mathbb{Z}_H \cdot h^*$, then z satisfies the R1CS instance, so to prove knowledge-soundness it suffices to suppose that $g_z \neq \mathbb{Z}_H \cdot h^*$ and show that this would contradict the poly-power discrete logarithm assumption.

If $g_z \neq \mathbb{Z}_H \cdot h^*$, then since both the left hand side and right hand side are polynomials of degree most 2ℓ, there are at most 2ℓ points r' for which $g_z(r') = \mathbb{Z}_H(r') \cdot h^*(r')$, and all such points r' can be enumerated in poly(S) time using a polynomial factorization algorithm. Consider

the algorithm $\mathcal{A}$ that selects one of these points r' at random. Clearly $\mathcal{A}$ runs in polynomial time, and with non-negligible probability (at least $1/(2\ell)$), it outputs r. We claim that this violates the poly-power discrete logarithm assumption. Indeed, since $\mathcal{A}_1, \ldots, \mathcal{A}_S, \mathcal{B}_1, \ldots, \mathcal{B}_S, \mathcal{C}_1, \ldots, \mathcal{C}_S$, are all polynomials of degree at most ℓ that are all computable in poly(S) time, the SRS for the SNARK of this section consists entirely of encodings of known linear combinations of powers-of-r (i.e., of products of known powers of $\mathsf{g}, \mathsf{g}^r, \mathsf{g}^{r^2}, \ldots, \mathsf{g}^{r^{\ell-1}}$), plus additional group elements equal to these values raised to either α, β_1, β_2, β_3, $\alpha \cdot \beta_1$, $\alpha \cdot \beta_2$, or $\alpha \cdot \beta_3$, where $\alpha, \beta_1, \beta_2, \beta_3$ are uniform random elements of $\{1, \ldots, p-1\}$. Hence, a string distributed identically to the entire SRS of the SNARK (which $\mathcal{A}$ is given access to) can be computed in polynomial time given the input encodings referred to in the poly-power discrete logarithm assumption. Since $\mathcal{A}$ outputs r with non-negligible probability, $\mathcal{A}$ violates the assumption.

17.5.5 Handling Public Input

For clarity, the presentation of the SNARK above elided the following detail. As per Remark 17.2, in many applications, there will in fact be a public input $x \in \mathbb{F}^n$ to the R1CS instance, and it will be required that the satisfying vector z have $z_i = x_i$ for $i = 1, \ldots, n$, with these requirements not otherwise included in the R1CS constraints. Remark 17.2 explained how to modify the linear PCP to enforce this. Essentially, the prover is forced to "ignore" the first n entries of z. Since the verifier knows x, the verifier on its own can "determine the contributions" of those entries of z to the verification checks.

This translates into the following modifications to the SNARK. First, let $z' = (z_{n+1}, \ldots, z_S)$ and replace the linear function f_z in the prescribed SNARK proof with $f_{z'}$. In more detail, letting $q^{(i)'} \in \mathbb{F}^{S-n}$ denote the last $S-n$ entries of $q^{(i)}$, the SNARK proof elements $(\mathsf{g}_1, \mathsf{g}_1')$, $(\mathsf{g}_2, \mathsf{g}_2')$, $(\mathsf{g}_3, \mathsf{g}_3')$, and $(\mathsf{g}_5, \mathsf{g}_5')$ are now respectively claimed to equal:

$$(\mathsf{g}^{f_{z'}(q^{(1)'})}, \mathsf{g}^{\alpha \cdot f_{z'}(q^{(1)'})}),$$
$$(\mathsf{g}^{f_{z'}(q^{(2)'})}, \mathsf{g}^{\alpha \cdot f_{z'}(q^{(2)'})}),$$

$$(\mathsf{g}^{f_{z'}(q^{(3)'})}, \mathsf{g}^{\alpha \cdot f_{z'}(q^{(3)'})}),$$

and

$$(\mathsf{g}^{f_{z'}(q^{(5)'})}, \mathsf{g}^{\alpha \cdot f_{z'}(q^{(5)'})}).$$

Second, the SRS entries $\mathsf{g}^{q_1^{(i)}}, \ldots, \mathsf{g}^{q_n^{(i)}}$ for $i \in \{1, 2, 3, 5\}$ get added to the verification key. Note that the prover does not need to know these entries so they can be omitted from the proving key. *Neither* the prover nor verifier need to know the α'th powers of these entries, and in fact, those α'th powers *must* be omitted from the proving and verification keys for the SNARK to be sound.[15]

For $i \in \{1, 2, 3\}$, let $\mathsf{g}^{(x,i)} = \mathsf{g}^{\sum_{j=1}^{n} x_j \cdot q_j^{(i)}}$, which can be computed by the verifier using the now-expanded verification key. Finally, the verifier's first check (Equation (17.12)) changes to

$$e\left(\mathsf{g}_1 \cdot \mathsf{g}^{(x,1)}, \mathsf{g}_2 \cdot \mathsf{g}^{(x,2)}\right) = e\left(\mathsf{g}_3 \cdot \mathsf{g}^{(x,3)}, \mathsf{g}\right) \cdot e\left(\mathsf{g}^{\mathbb{Z}_H(r)}, \mathsf{g}_4\right). \quad (17.15)$$

Remark 17.3. It is often asserted that the verifier's work in the SNARK above is dominated by the handful of evaluations of the bilinear map e performed across all of its checks. There are 17 such bilinear map evaluations in the SNARK presented above; as discussed in Section 17.5.6, Groth [142] gave a variant SNARK that reduces this number to 3. In fact, these bilinear map evaluations will only dominate the verifier's costs if the size n of the public input is reasonably small. Otherwise, the verifier's processing of the public input, specifically the computation of $\mathsf{g}^{(x,i)}$ above, will dominate, as this requires a multi-exponentiation of size n. Fortunately, in many applications, the public input is merely a *commitment* to a much larger witness, and hence is small.

17.5.6 Achieving Zero-Knowledge

The SNARK above is not zero-knowledge. One reason for this is that the proof contains encodings of f_z evaluated at various points, where z

[15] A variant SNARK given in [46] accidentally included these group elements in the SRS. Gabizon [121] gave an attack on the resulting SNARK, showing that these group elements' inclusion enables any prover to take a valid proof of knowledge of a witness w such that $\mathcal{C}(x, w) = y$, and for any $x' \neq x$ turn it into a "proof" for the potentially invalid statement that $\mathcal{C}(x', w) = y$. This flawed SNARK was deployed for several years in the cryptocurrency ZCash before the vulnerability was discovered. if exploited, it could have permitted unlimited counterfeiting of currency.

is a satisfying assignment for the R1CS instance. This leaks information to the verifier that the verifier cannot compute on its own, since the verifier does not know z.

To render the SNARK zero-knowledge, we modify the underlying LIP to be honest-verifier zero-knowledge. This ensures that the resulting SNARK is zero-knowledge even against dishonest verifiers, by the following reasoning. Because the SNARK verifier does not send any message to the prover, honest-verifier and malicious-verifier zero-knowledge are equivalent for the SNARK. The SNARK verifier only sees the verification key, which is generated in polynomial time and is independent of the witness vector z, and encodings of the LIP prover's responses to the LIP verifier's queries. Once the proving and verification keys are generated, these encodings are deterministic, efficiently computable functions of the responses. Hence, since the LIP is honest-verifier perfect zero-knowledge, so is the resulting SNARK. That is, the simulator for the SNARK verifier's view simply runs the simulator for the LIP verifier's view, and outputs encodings of the LIP prover's messages instead of the messages themselves.

Rendering the LIP Honest-Verifier Zero-Knowledge. Recall that in the nonzero-knowledge LIP, the prover established that $g_z = \mathbb{Z}_H \cdot h^*$ for an R1CS solution vector z, where

$$g_z(t) = \left(\sum_{\text{columns } j \in \{1,\ldots,S\}} z_j \cdot \mathcal{A}_j(t) \right) \cdot \left(\sum_{\text{columns } j \in \{1,\ldots,S\}} z_j \cdot \mathcal{B}_j(t) \right) - \left(\sum_{\text{columns } j \in \{1,\ldots,S\}} z_j \cdot \mathcal{C}_j(t) \right).$$

This required the LIP verifier to pick a random $r \in \mathbb{F}$ and obtain from the prover the following four evaluations:

$$h^*(r),$$
$$\sum_{\text{columns } j} z_j \cdot \mathcal{A}_j(r),$$
$$\sum_{\text{columns } j} z_j \cdot \mathcal{B}_j(r),$$

and

$$\sum_{\text{columns } j} z_j \cdot \mathcal{C}_j(r).$$

These four values leak information to the LIP verifier, who cannot efficiently compute W or h^*.

To render the LIP zero-knowledge, the prover picks three random values $r_A, r_B, r_C \in \mathbb{F}$, and considers a "perturbed" version g'_z of g_z, in which each constituent function comprising g_z has added to it a random multiple of the vanishing polynomial $\mathbb{Z}_H$ of H. Specifically, letting

$$\mathcal{A}(t) \coloneqq \sum_{\text{columns } j} z_j \cdot \mathcal{A}_j(t),$$

$$\mathcal{B}(t) \coloneqq \sum_{\text{columns } j} z_j \cdot \mathcal{B}_j(t),$$

$$\mathcal{C}(t) \coloneqq \sum_{\text{columns } j} z_j \cdot \mathcal{C}_j(t),$$

define:

$$g'_z(t) \coloneqq (\mathcal{A}(t) + r_A \mathbb{Z}_H(t)) \cdot (\mathcal{B}(t) + r_B \mathbb{Z}_H(t)) - (\mathcal{C}(t) + r_C \mathbb{Z}_H(t)). \tag{17.16}$$

Note that $g'_z(t) = g_z(t) + r_B \mathbb{Z}_H(t) \mathcal{A}(t) + r_A \mathbb{Z}_H(t) \mathcal{B}(t) + r_A r_B \left(\mathbb{Z}_H(t)\right)^2 - r_C \mathbb{Z}_H(t)$. Just as g_z vanished on H if and only if z is a satisfies the R1CS instance, the same can be said for g'_z, because the "added factors" in g'_z are multiples of the polynomial $\mathbb{Z}_H$, which vanishes on H.

To prove that g'_z vanishes on H, it is sufficient for the prover to establish that there exists a polynomial h' such that $g'_z = h' \cdot \mathbb{Z}_H$. Note that this is satisfied by

$$h' = h^* + r_B \cdot A(t) + r_A \cdot B(t) + r_A r_B \mathbb{Z}_H - r_C.$$

The LIP verifier can (with soundness error at most $2\ell/(|\mathbb{F}| - \ell)$) check that this equality of formal polynomials holds by confirming that the right hand and left hand sides agree at a random point $r \in \mathbb{F} \setminus H$.

The zero-knowledge LIP proof consists of two linear functions. The first is claimed to equal $f_{\text{coeff}(h')}$, defined as usual so that $f_{\text{coeff}(h')}(1, r, r^2, \ldots, r^{\deg(h')}) = h'(r)$. The second is prescribed to equal $f_{z'}$ where W' is

the vector $z \circ r_A \circ r_B \circ r_C \in \mathbb{F}^{S+3}$, where $\circ$ denotes concatenation. That is, z' is the satisfying R1CS vector z, with the random values r_A, r_B, r_C that are chosen by the prover appended.

The honest LIP verifier will query $f_{z'}$ at three locations:

$$q^{(1)} = (\mathcal{A}_1(r), \ldots, \mathcal{A}_S(r), \mathbb{Z}_H(r), 0, 0),$$
$$q^{(2)} = (\mathcal{B}_1(r), \ldots, \mathcal{B}_S(r), 0, \mathbb{Z}_H(r), 0),$$

and

$$q^{(3)} = (\mathcal{C}_1(r), \ldots, \mathcal{C}_S(r), 0, 0, \mathbb{Z}_H(r))$$

to obtain the three values:

$$v_1 \coloneqq r_A \cdot \mathbb{Z}_H(r) + \sum_{\text{columns } j} z_j \cdot \mathcal{A}_j(r),$$

$$v_2 \coloneqq r_B \cdot \mathbb{Z}_H(r) + \sum_{\text{columns } j} z_j \cdot \mathcal{B}_j(r),$$

and

$$v_3 \coloneqq r_C \cdot \mathbb{Z}_H(r) + \sum_{\text{columns } j} z_j \cdot \mathcal{C}_j(r).$$

The honest LIP verifier will then pick a point $r \in \mathbb{F} \setminus H$ at random and query $f_{\text{coeff}(h')}$ at a single point $q^{(4)} \coloneqq (1, r, r^2, \ldots, r^{\ell-1})$ to obtain a value v_4 claimed to equal $h'(r)$. Then the verifier will check that

$$v_1 \cdot v_2 - v_3 = v_4 \cdot \mathbb{Z}_H(r).$$

Finally, following Section 17.5.2, in order to confirm that the LIP prover answered the queries $q^{(1)}$, $q^{(2)}$, and $q^{(3)}$ with the same linear function, the verifier will also choose $\beta_1, \beta_2, \beta_3$ at random from $\mathbb{F}$ and query $f_{z'}$ at location $q^{(5)} = \sum_{i=1}^{3} \beta_i q^{(i)}$ to obtain response v_5, and check that $\sum_{i=1}^{3} \beta_i v_i = v_5$. By the discussion in Section 17.5.2, if the LIP prover passes this check, then with high probability a single linear function $f_{z'}$ was used to answer $q^{(1)}$, $q^{(2)}$, and $q^{(3)}$.

Analysis of the LIP. Completeness of this LIP holds by design. Soundness holds because for any linear functions $f_{\text{coeff}(h')}$ and $f_{z'}$ that cause the LIP verifier to accept, $\text{coeff}(h')$ must specify the coefficients of a

polynomial h' and z' must specify a witness $z \in \mathbb{F}^S$ followed by three values r_A, r_B, r_C such that

$$h'(r) \cdot \mathbb{Z}_H(r) = g'_z(r),$$

where $g'_z(t)$ is as defined in Equation (17.16). This implies that z is a valid circuit transcript.

The LIP is honest-verifier zero-knowledge by the following reasoning. Since $r \notin H$, we may conclude that $\mathbb{Z}_H(r) = \prod_{a \in H}(r-a) \neq 0$. Combined with the fact that r_A, r_B, and r_C are independent, uniform random field elements, it follows that $\mathbb{Z}_H(r) \cdot r_A$, $\mathbb{Z}_H(r) \cdot r_B$, and $\mathbb{Z}_H(r) \cdot r_C$ are uniform random field elements as well. Hence, $f_{z'}(q^{(1)})$, $f_{z'}(q^{(2)})$, and $f_{z'}(q^{(3)})$ are themselves uniform random field elements, as each is some fixed quantity plus a uniform random field element (e.g., $f_{z'}(q^{(1)}) = f_z(\mathcal{A}_1(r), \ldots, \mathcal{A}_S(r)) + \mathbb{Z}_H(r) \cdot r_A$).

Meanwhile, for any choice of $r \in \mathbb{F}$, $v_4 = h'(r)$ is always equal to

$$(v_1 \cdot v_2 - v_3)\, \mathbb{Z}_H(r)^{-1}. \tag{17.17}$$

Hence, the simulator can choose r at random from $\mathbb{F}$, and simply set v_1, v_2, v_3 (i.e., the simulated responses to queries $q^{(1)}$, $q^{(2)}$, $q^{(3)}$) to be uniform random field elements, and then set v_4 as per Equation (17.17). Finally, the simulator chooses $\beta_1, \beta_2, \beta_3$ at random from $\mathbb{F}$, and computes the simulated response v_5 to $q^{(5)}$ as $\sum_{i=1}^{3} \beta_i v_i$. This is a perfect simulation of the LIP verifier's view.

Historical Notes. The zk-SNARK described above is nearly identical to the one for QAPs given in [125]. Minor differences arise in our treatment, stemming from our use of the linear-PCP-to-LIP from subsequent work [56] in the construction and analysis of the SNARK presented here. [203] provided concrete improvements to the zk-SNARK from [125], and implemented the resulting variant. Other optimized variants were presented in [42], [46], [121].

Groth's SNARK. Groth [142] gave an influential variant of the zk-SNARK of this section, in which the proof consists of only 3 group elements, and proved his SNARK to be knowledge sound in the generic

group model. Roughly speaking, this reduction in proof size can be traced to two differences from the SNARK covered in this section. First, he gave an LIP in which the verifier only makes 3 queries, rather than 5 as in the LIP we cover. This alone reduces the number of group elements from 10 to 6. Second, establishing security in the generic group model rather than relying on Knowledge of Exponent assumptions allows for a halving of the number of group elements, as it is possible to ensure that each group element $\mathbf{g}_i$ in the proof need not be paired with $\mathbf{g}_i^{\alpha}$. Fuchsbauer *et al.* [117] extended the security proof of Groth's SNARK to the Algebraic Group Model.