14

Polynomial Commitments from Hardness of Discrete Logarithm

Polynomial Commitments Schemes and a Trivial Solution. Recall that polynomial commitment scheme is meant to simulate the following idealized process. An untrusted prover $\mathcal{P}$ has in its head a polynomial q (for applications to succinct arguments, we are primarily interested in the cases the q is a univariate polynomial, or a multilinear polynomial). $\mathcal{P}$ sends a complete description of q to the verifier $\mathcal{V}$ (say, a list of all of q's coefficients over an appropriate basis). $\mathcal{V}$, having learned q, can evaluate q at any point z of its choosing. In particular, once $\mathcal{P}$ sends the polynomial q to $\mathcal{V}$, $\mathcal{P}$ cannot go and "change" q based on the point z at which $\mathcal{V}$ wishes to evaluate it. Let us call this procedure, in which $\mathcal{P}$ explicitly sends q to $\mathcal{V}$, the *trivial polynomial commitment scheme*.

There are three potential issues with the trivial polynomial commitment scheme, two of which involve efficiency considerations.

- In our applications to SNARKs (Sections 7–10), q may be very large—often as large as the entire statement being proved. So having $\mathcal{P}$ send all coefficients of q to $\mathcal{V}$ will require a huge amount of communication. Hence, using the trivial polynomial commitment does not yield *succinct* arguments.

- $\mathcal{V}$ has to spend time linear in the number of coefficients to compute $q(z)$ (i.e., the trivial polynomial commitment would not yield a *work-saving* argument, meaning one whose verifier is faster than the trivial one that is sent a witness and checks its correctness).
- $\mathcal{V}$ learns the entire polynomial q. This may be incompatible with zero-knowledge (in applications to SNARKs, q typically "encodes" a witness, and hence sending q to $\mathcal{V}$ leaks the entire witness).

Using cryptography, one can hope to address all three issues while achieving the same functionality as the trivial polynomial commitment scheme. Specifically, $\mathcal{P}$ can compute a "compressing" commitment c to q and send only c to the verifier. Compressing means that c is much smaller than q, addressing the first issue above regarding succinctness. Because c is smaller than p itself, c does not bind $\mathcal{P}$ to q in a statistical sense. That is, there will *exist* many different polynomials for which c is a valid commitment, and when the verifier asks $\mathcal{P}$ the evaluation $q(z)$, $\mathcal{P}$ will be able to respond with $p(z)$ for any valid "opening" polynomial p of c. However, it is possible to design polynomial commitment schemes that are *computationally binding*, meaning that any efficient prover (e.g., one unable to solve the discrete logarithm problem, or find a collision in a cryptograph hash function) is unable to respond to any evaluation query z with a quantity other than $q(z)$. More precisely, along with a claimed value v for $q(z)$, the prover will send an *evaluation proof* π. Computational binding guarantees that any efficient prover will be unable to generate a convincing π unless indeed $v = q(z)$.

We have already covered some polynomial commitment schemes in earlier sections (e.g., Sections 10.4.2 and 10.5). As with those earlier schemes, in this section we will see polynomial commitment schemes in which π can be checked far faster than what would be required just to read an explicit description of q. This addresses the first two issues of the trivial scheme (succinctness and verifier time). We will also see schemes where π reveals nothing about q, and even schemes where, if desired, the verifier does not actually learn the requested evaluation $q(z)$ but rather a *hiding commitment* to $q(z)$. In this manner, the polynomial commitment schemes can support zero-knowledge, leaking no information about q (and the witness it encodes) to the verifier.

Revealing $q(z)$ Itself vs. a Commitment to $q(z)$. The polynomial commitment schemes we describe in Section 14 reveal to the verifier (Pedersen) commitments to the value $v = q(z)$, because this is what is required for their use in the zero-knowledge arguments of Section 13.2. Other zero-knowledge arguments (such as those in Section 13.3) call for v to be revealed explicitly to the verifier. Fortunately, it is easy to modify the commitment schemes of Section 14 to reveal v to the verifier, for example by having the prover use Protocol 3 to establish in zero-knowledge that it knows how to open the commitment to value v (see the final paragraph of Section 12.3.1 for details). The pairing-based polynomial commitment scheme in Section 15 is described in the setting where the evaluation $v = q(z)$ is revealed explicitly to the verifier.

Polynomial Commitments from Earlier Sections and How they Compare to this Section. We have previously seen that one way to obtain a polynomial commitment scheme is to combine an appropriate PCP or IOP with Merkle-hashing (Sections 10.4.2 and 10.5). Whereas the Merkle-hashing approach only exploited "symmetric key" cryptographic primitives (namely collision-resistant hash functions, combined with the random oracle model to remove interaction), the approaches in this section are based on "public key" cryptographic primitives. Such primitives require stronger cryptographic assumptions such as hardness of the discrete logarithm problem in elliptic curve groups. Discussion of the pros and cons of IOP-based polynomial commitments vs. the commitments of this section can be found in Section 16.3.

Overview of this Section's Schemes. Known polynomial commitment schemes tend to be somewhat more general: they enable a prover to commit to any *vector* $u \in \mathbb{F}^n$, and then later prove statements about the inner product of u with any vector $y \in \mathbb{F}^n$ requested by the verifier. In the polynomial commitment scheme, u will be the coefficients of the polynomial q to be committed over an appropriate basis (e.g., the standard monomial basis for univariate polynomial, or the Lagrange basis for multilinear polynomials). Evaluating $q(z)$ is then equivalent to computing the inner product of u with the vector y obtained by evaluating each basis polynomial at z.

For example, if q is univariate, say, $q(X) = \sum_{i=0}^{n-1} u_i X^i$, then for any input z to q $q(z) = \langle u, y \rangle$ where $y = (1, z, z^2, \ldots, z^{n-1})$ consists of powers of z, and $\langle u, y \rangle = \sum_{i=0}^{n-1} u_i y_i$ denotes the inner product of u and y. Similarly, if q is multilinear, say $q(X) = \sum_{i=0}^{2^\ell} u_i \chi_i(X)$, where $\chi_1, \ldots, \chi_{2^\ell}$ denotes the natural enumeration of the Lagrange basis polynomials,[1] then for $z \in \mathbb{F}_p^\ell$, $q(r) = \langle u, y \rangle$ where $y = \langle \chi_1(z), \ldots, \chi_{2^\ell}(z) \rangle$ is the vector of all Lagrange basis polynomials evaluated at z.

Hence, to commit to q, it suffices to commit to the *vector* u *of coefficients* of q. Then to later reveal (a commitment to) $q(z)$, it suffices to reveal (a commitment to) the inner product of u with the vector y.

Tensor Structure in the Evaluation Vector. Exactly as in Section 10.5.1, in both the univariate and multilinear cases above, the vector y has a tensor-product structure. Some, but not all, of the polynomial commitment schemes covered in this section will exploit this tensor structure (specifically, the schemes in Sections 14.3 and 15.4); the others support inner products of a committed vector with an *arbitrary* vector y.

What we mean by tensor structure is the following. In the univariate case, let $n - 1$ equal the degree of q and let us assume $n = m^2$ is a perfect square, and define $a, b \in \mathbb{F}^m$ as $a := (1, z, z^2, \ldots, z^{m-1})$ and $b := (1, z^m, z^{2m}, \ldots, z^{m(m-1)})$. If we view y as an $m \times m$ matrix with entries indexed as $(y_{1,1}, \ldots, y_{m,m})$, then y is simply the outer product $b \cdot a^T$ of a and b. That is, $y_{i,j} = z^{i \cdot m + j} = b_i \cdot a_j$. Similarly, if q is an ℓ-variate multilinear polynomial, suppose that $2^\ell = m^2$, and let $z_1, z_2 \in \mathbb{F}^{\ell/2}$ denote the first half and second half of $z \in \mathbb{F}^\ell$. Then let $\chi'_1, \ldots, \chi'_m$ denote the natural enumeration of the $(\ell/2)$-variate Lagrange basis polynomials, and define $a, b \in \mathbb{F}^m$ as $a := (\chi'_1(z_1), \ldots, \chi'_m(z_1))$ and $b := (\chi'_1(z_2), \ldots, \chi'_m(z_2))$. Then $y = b \cdot a^T$. That is, $y_{i,j} = \chi_{i \cdot m + j}(z) = \chi'_i(z_1) \cdot \chi'_j(z_2) = b_i \cdot a_j$.

In summary, for both univariate and multilinear polynomials q, once the coefficient vector u of q is committed, computing $q(z)$ is equivalent to evaluating the inner product of u with a vector y satisfying $y_{i,j} = b_i \cdot a_j$

[1] See Lemma 3.7 for a definition of the Lagrange basis polynomials. In the natural enumeration, if i has binary representation $i_1, \ldots, i_\ell \in \{0,1\}^\ell$, then $\chi_i(X_1, \ldots, X_\ell) = \prod_{j=1}^{\ell} (X_i i_j + (1 - X_i)(1 - i_j)) = \left(\prod_{j:\, i_j = 1} X_j\right) \left(\prod_{j:\, i_j = 0} (1 - X_j)\right)$.

for some m-dimensional vectors a, b, where m is the square root of the number of coefficients of q. Equivalently, we can express the inner product of u and y as a vector-matrix-vector product:

$$\langle u, y \rangle = \sum_{i,j=1,\ldots,m} u_{i,j} b_i a_j = b^T \cdot u \cdot a, \tag{14.1}$$

where on the right hand side we are viewing u as an $m \times m$ matrix. See Figures 14.1 and 14.2 for examples in both the univariate and multilinear cases.

14.1 A Zero-Knowledge Scheme with Linear Size Commitments

We begin by describing a scheme that does not improve over the costs of the trivial polynomial commitment scheme, but does render it zero-knowledge. That is, is the prover's commitment to q is as large of q itself, and given the commitment to q, the verifier on its own can derive a commitment to $q(z)$ for any input z of the verifier's choosing.

$$q(z) = 3 + 5z + 7z^2 + 9z^3 + z^4 + 2z^5 + 3z^6 + 4z^7 + 2z^8 + 4z^9 + 6z^{10} + 8z^{11} + 3z^{13} + 6z^{14} + 9z^{15}$$

$$= \begin{array}{|c|c|c|c|} \hline 1 & z^4 & z^8 & z^{12} \\ \hline \end{array} \cdot \begin{array}{|c|c|c|c|} \hline 3 & 5 & 7 & 9 \\ \hline 1 & 2 & 3 & 4 \\ \hline 2 & 4 & 6 & 8 \\ \hline 0 & 3 & 6 & 9 \\ \hline \end{array} \cdot \begin{array}{|c|} \hline 1 \\ \hline z \\ \hline z^2 \\ \hline z^3 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 3 & 5 & 7 & 9 & 1 & 2 & 3 & 4 & 2 & 4 & 6 & 8 & 0 & 3 & 6 & 9 \\ \hline \end{array} \cdot$$

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 1 & z & z^2 & z^3 & z^4 & z^5 & z^6 & z^7 & z^8 & z^9 & z^{10} & z^{11} & z^{12} & z^{13} & z^{14} & z^{15} \\ \hline \end{array}$$

Figure 14.1: Example of a degree-15 univariate polynomial q expressed via its coefficients over the standard monomial basis. The second line shows that the evaluation $q(z)$ for any input z can be expressed as a vector-matrix-vector product, where the matrix is specified by the coefficients of q, and the two vectors by the evaluation point r. The third line shows $q(z)$ can be equivalently be expressed as an inner product between the coefficient vector of q and an "evaluation vector" consisting of powers of z.

$$
\begin{aligned}
&q(r_1, r_2, r_3, r_4) = \\
&3(1-r_1)(1-r_2)(1-r_3)(1-r_4) + 5(1-r_1)(1-r_2)(1-r_3)r_4 + 7(1-r_1)(1-r_2)r_3(1-r_4) + 9(1-r_1)(1-r_2)r_3r_4 \\
&+(1-r_1)r_2(1-r_3)(1-r_4) + 2(1-r_1)r_2(1-r_3)r_4 + 3(1-r_1)r_2r_3(1-r_4) + 4(1-r_1)r_2r_3r_4 \\
&+2r_1(1-r_2)(1-r_3)(1-r_4) + 4r_1(1-r_2)(1-r_3)r_4 + 6r_1(1-r_2)r_3(1-r_4) + 8r_1(1-r_2)r_3r_4 \\
&+3r_1r_2(1-r_3)r_4 + 6r_1r_2r_3(1-r_4) + 9r_1r_2r_3r_4
\end{aligned}
$$

$$
= \begin{array}{|c|c|c|c|} \hline (1-r_1)(1-r_2) & (1-r_1)r_2 & r_1(1-r_2) & r_1r_2 \\ \hline \end{array} \cdot \begin{array}{|c|c|c|c|} \hline 3 & 5 & 7 & 9 \\ \hline 1 & 2 & 3 & 4 \\ \hline 2 & 4 & 6 & 8 \\ \hline 0 & 3 & 6 & 9 \\ \hline \end{array} \cdot \begin{array}{|c|} \hline (1-r_3)\,(1-r_4) \\ \hline (1-r_3)\,r_4 \\ \hline r_3(1-r_4) \\ \hline r_3\,r_4 \\ \hline \end{array}
$$

Figure 14.2: Example of a 4-variate multilinear polynomial q expressed via its coefficients over the Lagrange basis (see Lemma 3.7). The evaluation $q(r)$ for any input $r = (r_1, r_2, r_3, r_4) \in \mathbb{F}^4$ can be expressed as a vector-matrix-vector product, where the matrix is specified by the coefficients of q, and the two vectors by the evaluation point r.

Recall that in a Pedersen commitment (Section 12.3) over group $\mathbb{G}$ of prime order p with generators g, h, a commitment to a value $m \in \mathbb{F}_p$ is $c \leftarrow h^m \cdot g^r$ for a value $r \in \{0, \ldots, p-1\}$ randomly chosen by the committer. Pedersen commitments are perfectly hiding and computationally binding.

Commitment Phase. To commit to q, rather than the prover sending each entry of the coefficient vector u to the verifier "in the clear" as in the trivial scheme, $\mathcal{P}$ sends a Pedersen commitment c_i to each entry u_i of u. Pedersen commitments are hiding, so this reveals to the receiver nothing at all about u.

Evaluation Phase. Let y be the vector such that $q(z) = \langle u, y \rangle = \sum_i u_i y_i$. Since the verifier knows y and has a commitment to each entry u_i of u, using the homomorphism property of Pedersen commitments, the verifier can on its own derive a commitment c to $\sum_i u_i y_i$. $\mathcal{P}$ can prove in zero-knowledge that it knows how to open the commitment c via Protocol 7 (Section 12.3.1).

Extractability. In the above scheme, suppose that the committer, before sending the commitment, knows what the evaluation query z will

be. As we now explain, the committer can arrange to be able to open the commitment c derived by the verifier in the evaluation phase to a value $a \in \mathbb{F}_p$ of the committer's choosing, *without* being able to open any of the commitments c_i sent during the commit phase. Put another way, the commitment scheme is not extractable in this setting (see Section 7.4): the prover may not "know" a polynomial p of the claimed degree that explains its answers to all evaluation queries for which it is capable of passing the verifier's checks.

For example, suppose that $n = 2$, so the polynomial commitment consists of two Pedersen commitments, c_0 and c_1, each ostensibly a Pedersen commitment to one of the two coefficients of the polynomial. For simplicity, assume and the group generator h used as a blinding factor in the Pedersen commitment scheme is the identity element in $\mathbb{G}$ (equivalently, the blinding factor is simply omitted from the Pedersen commitment scheme).

Then a committer could choose c_0 to be a random group element, meaning the committer is unable to open c_0, and also set c_1 to $\left(g^a \cdot c_0^{-1}\right)^{z^{-1}}$ where z^{-1} denotes the multiplicative inverse of z modulo p. This ensures that the commitment c derived by the verifier during the evaluation phase above is $c_0 \cdot c_1^z = g^a$, which the committer can open to a, despite not knowing how to open the commitments c_0 and c_1.

One way to address the above issue and thereby achieve extractability is to modify the commitment phase to require the committer to prove in zero-knowledge, via Protocol 7, that it can open each Pedersen commitment c_i. Of course, this concretely increases the costs of the commitment phase.

In applications of polynomial commitment schemes to succinct arguments, the evaluation point z is chosen at random by the verifier and not known to the prover at the time the polynomial commitment is sent. In this setting, the above polynomial commitment scheme *is* extractable (without modifying the commitment phase). Specifically, by randomly choosing many different evaluation points, the extraction procedure can find n evaluation points $z^{(1)}, \ldots, z^{(i)}$ for which the committer is able to pass the verifier's checks. If $y^{(i)}$ denotes the vector such that

$q(z^{(i)}) = \langle u, y^{(i)} \rangle$, and the committer claims that $q(z^{(i)}) = v^{(i)}$,[2] then this yields n linearly independent equations in the n unknown entries of u, with the i'th equation being

$$\langle u, y^{(i)} \rangle = v^{(i)}.$$

The extractor then uses Gaussian elimination to efficiently solve these n equations for the entries of u, i.e., for the coefficients of the committed polynomial q.

Similar remarks apply to the polynomial commitment scheme with square-root verification costs given later, in Section 14.3.

14.2 Constant Size Commitments But Linear Size Evaluation Proofs

In the commitment scheme of Section 14.1, the commitment was as big as the polynomial being committed. In this section, we give a scheme that reduces the commitment size to constant (one group element). However, evaluation proofs (and hence also verification time) will become very large—as big as the polynomial being committed.[3]

Commitment Phase. Assume that n generators $g_1, \ldots, g_n$ for $\mathbb{G}$ are chosen at random from $\mathbb{G}$. To commit to $u \in \mathbb{F}_p^n$, committer will pick a random value $r_u \in \{0, 1, \ldots, |\mathbb{G}| - 1\}$ and send the value $\mathsf{Com}(u; r_u) := h^{r_u} \cdot \prod_{i=1}^{n} g_i^{u_i}$. This quantity is often referred to as a *generalized Pedersen commitment*, or a *Pedersen vector commitment* (a standard Pedersen commitment is equivalent to a generalized Pedersen commitment when $n = 1$). Note that Pedersen vector commitments are homomorphic: given two commitments c_u, c_w to two vectors u and w in $\mathbb{F}_p^n$, and any

[2] More precisely, the evaluation $v^{(i)}$ is not explicitly revealed by the committer in the scheme of this section. Rather, the committer proves knowledge of $v^{(i)}$ using Protocol 7. But $v^{(i)}$ can be efficiently extracted from the committer owing to the knowledge-soundness of Protocol 7.

[3] The number of public parameters for the scheme of this section is also very large, consisting of n randomly chosen group elements $g_1, \ldots, g_n$, where n is the length of the coefficient vector u. But in the random oracle model g_i can be chosen via a random oracle by evaluating the random oracle at input i, in which case the public parameter size is constant.

two scalars $a_1, a_2 \in \mathbb{F}_p$, one can compute a commitment to the linear combination $a_1 u + a_2 w$, as $c_u^{a_1} \cdot c_w^{a_2}$.

Pedersen *vector* commitments should contrasted with the scheme of Section 14.1, which committed to the vector $u \in \mathbb{F}_p^n$ by sending a different Pedersen commitment for each entry of u (using the same public group generator g for all n commitments). This was not a compressing commitment. Pedersen vector commitments compute a different Pedersen commitment $g_i^{u_i}$ for each entry u_i of u (but without a blinding factor, see Footnote 14), with each commitment using a different group generator $g_i \in \mathbb{G}$.[4] But rather than sending all n commitments to the verifier, they are all "compressed" into a single commitment using the group operation of $\mathbb{G}$ (and then the result is blinded by the factor h^{r_u}).

Evaluation Phase. Recall that evaluating a committed polynomial q at input z is equivalent to computing $\langle u, y \rangle$ for the coefficient vector u and a vector y derived from z. Suppose we are given a commitment $c_u = \mathsf{Com}(u, r_u)$, a public query vector y, and a commitment $c_v = \mathsf{Com}(v, r_v) = g_1^v \cdot h^{r_v}$ where $v = \langle u, y \rangle$, and the committer knows r_u and r_v but the verifier does not. The committer wishes to prove in zero-knowledge that it knows an openings u of c_u and v of c_v such that $\langle u, y \rangle = v$, as unless the prover can break the binding property of the commitments, this is equivalent to establishing that $q(z) = v$.

As with Protocol 7 (see the end of Section 12.3.1), directly opening the commitments to u and v would enable to verifier to easily check that $v = \langle u, y \rangle$, but would violate zero-knowledge. So instead the prover opens *derived* commitments, with both the prover and verifier contributing randomness to the derived commitments in a manner such that the derived commitments satisfy the same property that the prover claims is satisfied by the original commitments.

In more detail, first the committer samples a random n-dimensional vector d with entries in $\{0, \ldots, p-1\}$ and two random values $r_1, r_2 \in$

[4] If the same group generator g were used for all i, the commitment $\prod_{i=1}^n g^{u_i}$ would not bind the committer to the vector u, but rather only to some permutation of u. For example, if $n = 2$, then $u = (1, 2)$ would produce the same commitment as $u = (2, 1)$: in the first case the commitment would be $g^2 \cdot g = g^3$ and in the second case it would be $g \cdot g^2$, which also equals g^3.

$\{0, \ldots, p-1\}$. The committer sends two values $c_1, c_2 \in \mathbb{G}$ claimed to equal $\mathsf{Com}(d, r_1)$ and $\mathsf{Com}(\langle d, y\rangle, r_2)$. The verifier responds with a random challenge $e \in \{0, \ldots, p-1\}$. The prover responds with three quantities $u', r_{u'}, r_{v'} \in \{0, \ldots, p-1\}$ claimed to respectively equal the following (with all arithmetic done modulo p):

$$e \cdot u + d \in \{0, \ldots, p-1\}^n, \tag{14.2}$$

$$e \cdot r_u + r_1 \in \{0, \ldots, p-1\}, \tag{14.3}$$

$$e \cdot r_v + r_2 \in \{0, \ldots, p-1\}. \tag{14.4}$$

Finally, the verifier checks that $c_u^e \cdot c_1 = \mathsf{Com}(u', r_{u'})$ and $c_v^e \cdot c_2 = \mathsf{Com}(\langle u', y\rangle, r_{v'})$.

This protocol can be proved complete, special-sound, and perfect honest-verifier zero knowledge in a manner very similar to Protocol 7. Before writing out the formal analysis, we explain how each step of the protocol here is in direct analogy with each step of Protocol 7.

In Protocol 7, the prover's first message contained a commitment to a random value $d \in \{0, \ldots, p-1\}$. Here, since we are dealing with vector commitments, the prover's first message contains a commitment to a random *vector* d with entries in $\{0, \ldots, p-1\}$. Since this protocol is meant to establish not only that the prover knows how to open the commitment to vector u, but also that the prover knows how to open the second commitment to $\langle u, y\rangle$, the protocol here also has the prover send a second commitment, to $\langle d, y\rangle$.

In Protocol 7, after the verifier sent a random challenge e, the prover responded with an opening of the commitment to $e \cdot m + r_1$ that can be derived homomorphically from the commitments to m and d. Analogously, here the prover responds with opening information for the derived commitments to the vector $e \cdot u + d$ (that opening information is specified via Equations (14.2) and (14.3)) and to the value $\langle eu + d, y\rangle$ (the opening information is $(\langle u', y\rangle, r_{v'})$, where if the prover is honest, u' is specified as per Equation (14.2) and $r_{v'}$ is specified as per Equation (14.4)). In both protocols, the verifier simply checks that the opening information for both commitment(s) is valid. Effectively, the verifier confirms that the derived commitments, to vector $u' = eu + d$ and to value $\langle u', y\rangle$, satisfy the same relationship that the prover claims holds

Protocol 11 Evaluation phase of the polynomial commitment scheme of Section 14.2. If the committed polynomial is q and the evaluation point is z, $u \in \mathbb{F}_p^n$ denotes the coefficient vector of q (which is assumed to be defined over field $\mathbb{F}_p$ for prime p) and $y \in \mathbb{F}_p^n$ is a vector such that $q(z) = \langle u, y \rangle = \sum_{i=1}^n u_i \cdot y_i$.

1: Let $\mathbb{G}$ be a (multiplicative) cyclic group of prime order p over which the Discrete Logarithm relation is hard, with randomly chosen generators $h, g_1, \ldots, g_n$ and g.
2: Let $c_u = \mathsf{Com}(u; r_u) := h^{r_u} \cdot \prod_{i=1}^n g_i^{u_i}$ and $c_v = \mathsf{Com}(v, r_v) = g^v h^{r_v}$. Prover knows u, r_u, v, and r_v. Verifier only knows c_u, c_v, h, $g_1, \ldots, g_n$, and g.
3: Prover picks $d \in \{0, \ldots, p-1\}^n$ and $r_1, r_2 \in \{0, \ldots, p-1\}$ and sends to verifier $c_d := \mathsf{Com}(d, r_1)$ and $c_{\langle d,y \rangle} := \mathsf{Com}(\langle d, y \rangle, r_2)$.
4: Verifier sends challenge e chosen at random from $\{0, \ldots, |\mathbb{G}| - 1\}$.
5: Let $u' \leftarrow u \cdot e + d$ and $r_{u'} \leftarrow r_u \cdot e + r_1$, and let $c_{u'} \leftarrow \mathsf{Com}(u', r'_u)$. While Verifier does not know u' and $r_{u'}$, Verifier can derive $c_{u'}$ unaided from c_u and c_d using additive homomorphism, as $c_u^e \cdot c_d$.
6: Similarly, let $v' \leftarrow v \cdot e + \langle d, y \rangle = \langle u', y \rangle$ and $r_{v'} \leftarrow r_v \cdot e + r_2$, and let $c_{v'} \leftarrow \mathsf{Com}(v', r_{v'})$. While Verifier does not know v' and $r_{v'}$, Verifier can derive $c_{v'}$ unaided from c_v and $c_{\langle d,y \rangle}$ using additive homomorphism, as $c_v^e \cdot c_{\langle d,y \rangle}$.
7: Prover sends $(u', r_{u'})$ and $r_{v'}$ to Verifier.
8: Verifier checks that $(u', r_{u'})$ is valid opening information for $c_{u'}$, and that $(\langle u', y \rangle, r_{v'})$ is valid opening information for $c_{v'}$. Equivalently, Verifier checks that:

$$h^{r_{u'}} \cdot \prod_{i=1}^{n} g_i^{u'_i} = c_{u'}$$

and

$$h^{r_{v'}} \cdot g^{\langle u', y \rangle} = c_{v'}.$$

between the original committed vector u and value v, namely that the latter equals the inner product of the former with vector y.

Completeness, Special Soundness, and Zero-Knowledge. Completeness is clear by inspection of Protocol 11. For special soundness, let

$(c_d, c_{\langle d,y\rangle})$ be the first message sent by the prover, and let $((c_d, c_{\langle d,y\rangle}), e, (u^*, c_{u^*}, r^*))$ and $((c_d, c_{\langle d,y\rangle}), e', (\hat{u}, c_{\hat{u}}, \hat{r}))$ be two accepting transcripts. Owing to the transcripts passing the two tests performed by the verifier in Step 8 of Protocol 11, this means that:

$$h^{r_{u^*}} \cdot \prod_{i=1}^{n} g_i^{u_i^*} = c_u^e \cdot c_d, \tag{14.5}$$

$$h^{r_{\hat{u}}} \cdot \prod_{i=1}^{n} g_i^{\hat{u}_i} = c_u^{e'} \cdot c_d, \tag{14.6}$$

$$h^{r^*} \cdot g^{\langle u^*, y\rangle} = c_v^e \cdot c_{\langle d,y\rangle}. \tag{14.7}$$

$$h^{\hat{r}} \cdot g^{\langle \hat{u}, y\rangle} = c_v^{e'} \cdot c_{\langle d,y\rangle}. \tag{14.8}$$

Let

$$\bar{u} \coloneqq (u^* - \hat{u}) \cdot (e - e')^{-1} \mod |\mathbb{G}|,$$
$$r_{\bar{u}} \coloneqq (r_{u^*} - r_{\hat{u}}) \cdot (e - e')^{-1} \mod |\mathbb{G}|,$$

and

$$r_{\bar{v}} \coloneqq (r^* - \hat{r}) \cdot (e - e')^{-1} \mod |\mathbb{G}|$$

Dividing Equation (14.5) by Equation (14.6) implies that

$$h^{\bar{v}} \cdot \prod_{i=1}^{n} g_i^{\bar{u}_i} = c_u,$$

and dividing Equation (14.7) by Equation (14.8) implies that:

$$h^{\bar{r}} \cdot g^{\langle \bar{u}, y\rangle} = c_v.$$

That is, $(\bar{u}, r_{\bar{u}})$ is an opening of c_u and $(\langle \bar{u}, y\rangle, r_{\bar{v}})$ is an opening of c_v, and the two openings satisfy the claimed relationship that the value committed by c_v is the inner product of the vector committed by c_u with y.

To establish honest-verifier perfect zero-knowledge, consider the following simulator. To generate an accepting transcript $((c_d, c_{\langle d,y\rangle}), e, (u', r_{u'}, r_{v'}))$, the simulator proceeds as follows. First, it selects the verifier's challenge e at random from $\{0, \ldots, p-1\}$, and then picks a vector u' at random from $\{0, 1, \ldots, p-1\}^n$ and $r_{u'}, r_{v'}$ at random from

$\{0, 1, \ldots, p-1\}$. Finally, it chooses c_d and $c_{\langle d,y \rangle}$ to be the unique values that yield an accepting transcript, i.e., c_d is set to $c_u^{-e} \cdot h^{r_{u'}} \cdot \prod_{i=1}^{n} g_i^{u'_i}$ and $c_{\langle d,y \rangle}$ is set to $c_v^{-e} \cdot h^{r_{v'}} \cdot g^{\langle u', y \rangle}$. These choices of c_d and c_v are specifically chosen to ensure that the generated transcript is an accepting one. One can show that the distribution over accepting transcripts generated by this simulator is equal to the distribution generated by the honest verifier interacting with the honest prover by establishing a one-to-one correspondence between transcripts that the simulator outputs with transcripts generated by the honest verifier interacting with the honest prover (details omitted for brevity).

Costs. The commitment consists of a single group element. The computational cost of computing the commitment is performing n group exponentiations. Naively performing each exponentiation independently using repeated squaring requires $O(\log |\mathbb{G}|)$ group multiplications per exponentiation, which implies $\Theta(n \log |\mathbb{G}|)$ group multiplications in total. However, Pippenger's multi-exponentiation algorithm [209] can reduce this quantity by a factor of $(\log(n) + \log\log |\mathbb{G}|)$.[5]

In the evaluation phase, the proof consists of $n + 2$ numbers in $\{0, \ldots, p-1\}$ that can be computed with $O(n)$ field operations in $\mathbb{F}_p$ in total. The verification procedure requires the verifier to perform $O(n)$ group exponentiations, which can be performed using Pippenger's algorithm in the time bound described in the previous paragraph.

14.3 Trading Off Commitment Size and Verification Costs

Recall that the polynomial commitment scheme of the previous section had very small commitments (1 group element), but large proofs-of-evaluation ($\Theta(n)$ group elements).

In this section, we show how to exploit tensor-structure in the vector y (captured in Equation (14.1)) to reduce the size of the proof in the evaluation phase of the polynomial commitment scheme of the previous section, at the cost of increasing the commitment size. For example,

[5] A *multi-exponentiation* in a multiplicative group is a product of powers of elements of the group.

we can set both the commitment size and the evaluation proof size to $\Theta(\sqrt{n})$ group elements. This technique was presented in the context of multilinear polynomials in a system called Hyrax [244], building directly on a univariate polynomial commitment scheme given in [66].

Commitment Phase. Recall that u denotes the coefficient vector of the polynomial q to which the committer wishes to commit, and as per Equation (14.1), we view u as an $m \times m$ matrix. Letting $u_j \in \mathbb{F}^m$ denote the jth column of u, the committer picks random numbers $r_1, \ldots, r_m \in \{0, \ldots, p-1\}$ and sends a set of vector commitments $c_1 = \mathsf{Com}(u_1, r_1), \ldots, c_m = \mathsf{Com}(u_m, r_m)$, one for each column. Here,

$$\mathsf{Com}(u_j, r_j) = h^{r_j} \cdot \prod_{k=1}^{m} g_k^{u_{j,k}}$$

for public parameters $g_1, \ldots, g_m \in \mathbb{G}$. Hence, compared to the previous section, we have increased the size of the commitment for u from 1 group element to m group elements. Rather than applying the vector-commitment scheme of the previous section to one vector of length m^2, we applied it m times, to vectors of length m.

Evaluation Phase. When the verifier asks the committer to provide a commitment to $q(z)$ for a verifier-selected input z, the prover sends a commitment c^* to $q(z) = \langle u, y \rangle = b^T \cdot u \cdot a$, where the m-dimensional vectors b and a are as in Equation (14.1) and are known to both the prover and verifier. Using the additive homomorphism of the commitment scheme, the verifier can on its own compute a commitment to the vector $u \cdot a$, namely $\prod_{j=1}^{m} \mathsf{Com}(u_j)^{a_j}$. At this point, the prover needs to prove that c^* is a commitment to $b^T \cdot (u \cdot a) = \langle b, u \cdot a \rangle$. Since the verifier has derived a commitment to the vector $u \cdot a$, this is exactly an instance of the problem that the protocol of the previous section was designed to solve, using a proof of size m.[6]

[6] A subtlety of this polynomial commitment scheme's security guarantee is that the protocol establishes only that the prover knows a z-dependent linear combination of the column commitments, where z is the evaluation point. This means that if the committer can choose z, the committer may be able to pass the evaluation phase

In summary, we have given a (public-coin) commitment scheme for univariate and multilinear polynomials in which the commitment size proof length in the evaluation phase, and total verifier time are equal to the square root of the number of coefficients of the polynomial.

14.4 Bulletproofs

In this section, we give a scheme in which the commitment size is constant and the proof length in the evaluation phase is *logarithmic* in the number of coefficients of the polynomial. However, the verifier's runtime to process the proof is *linear* in the number of coefficients. Compared to the commitment scheme of Section 14.2, this is a strict improvement because the proof length in the evaluation phase is logarithmic as opposed to linear in the length of the coefficient vector. Compared to the scheme of Section 14.3, the verifier's runtime is worse (linear rather than proportional to the square root of the number of coefficients), but the communication cost is much better (logarithmic as opposed to square root).

The scheme of this section is a variant of a system called Bulletproofs [80], which itself directly builds on a univariate polynomial commitment scheme given in [66]. Our presentation draws substantially on a perspective developed in [64].

14.4.1 Warm-up: Proof of Knowledge for Opening of Vector Commitment

Before presenting the polynomial commitment in full, we start with a warmup that illustrates the key ideas of the full Bulletproofs polynomial commitment scheme. Specifically, the warmup is a protocol enabling the prover to establish that it knows how to open a generalized Pedersen commitment to a vector $u \in \mathbb{F}_p^n$.

---

without knowing how to open every column commitment [179]. This weakened guarantee is nonetheless sufficient for the scheme's use in succinct interactive arguments and SNARKs derived thereof, where z is chosen at random by the SNARK verifier or via the Fiat-Shamir transformation.

Notational Changes for this Section. Recall that a generalized Pedersen commitment to u is $\mathsf{Com}(u; r_u) := h^{r_u} \cdot \prod_{i=1}^{n} g_i^{u_i}$ where r_u is chosen at random by the committer, and the g_i's are public generators in $\mathbb{G}$. To further simplify the presentation, let us omit the blinding factor h^{r_u} from the vector commitment, so that we now define $\mathsf{Com}(u) := \prod_{i=1}^{n} g_i^{u_i}$ (the resulting commitment scheme without the blinding factor is still computationally binding, but it is not perfectly hiding, see Footnote 14 in Section 12.3).

For the remainder of this section, we write $\mathbb{G}$ as an additive rather than multiplicative group. This is because, by doing so, we can think of $\mathsf{Com}(u) = \sum_{i=1}^{n} u_i \cdot g_i$ as the inner product between u and the vector $\mathbf{g} = (g_1, \ldots, g_n)$ of public group generators, and hence we denote $\sum_{i=1}^{n} u_i \cdot g_i$ as $\langle u, \mathbf{g} \rangle$.[7] Under this notation, the prover is claiming to know a vector u such that

$$\langle u, \mathbf{g} \rangle = c_u. \tag{14.9}$$

Overview of the Protocol. The protocol is vaguely reminiscent of the IOP-based polynomial commitment scheme FRI (Section 10.4.4), in the following sense. At the start of the protocol, the prover has sent a commitment c_u to a vector u of length n. The protocol proceeds in $\log_2 n$ rounds, where in each round i the verifier sends the prover a random field $\alpha_i \in \mathbb{F}_p$, and α_i is used to "halve the length of the committed vector". After $\log_2 n$ rounds, the prover is left with a claim that it knows a vector u of length 1 satisfy a certain inner product relationship. In this case, u is so short that the prover can succinctly prove the claim by simply sending u to the verifier.

In more detail, at the start of each round $i = 1, 2, \ldots, \log_2 n$, the prover has sent a commitment $c_{u^{(i)}}$ to some vector $u^{(i)}$ of length $n \cdot 2^{-(i-1)}$, and the prover must establish that it knows a vector $u^{(i)}$ such that $\langle u^{(i)}, \mathbf{g}^{(i)} \rangle = c_{u^{(i)}}$ (when $i = 1$, $u^{(i)} = u$ and $\mathbf{g}^{(i)} = \mathbf{g}$). The goal of round i is to reduce the claim that $\langle u^{(i)}, \mathbf{g}^{(i)} \rangle = c_{u^{(i)}}$ to a claim of the

[7] Strictly speaking, referring to $\sum_{i=1}^{n} u_i g_i$ as an inner product is a misnomer because the u_i's are integers in $\{0, 1, \ldots, p-1\}$ while the g_i's are elements of the group $\mathbb{G}$ of size p, but we ignore this and write $\mathsf{Com}(u) = \langle u, \mathbf{g} \rangle$ for the remainder of this section.

same form, namely that the prover knows a vector $\langle u^{(i+1)}, \mathbf{g}^{(i+1)} \rangle = c_{u^{(i+1)}}$ for some vector of group generators $\mathbf{g}^{(i+1)}$ that is known to the verifier, but where $u^{(i+1)}$ and $\mathbf{g}^{(i+1)}$ each have half the length of $u^{(i)}$ and $\mathbf{g}^{(i)}$. For notational brevity let us fix a round i and accordingly drop the superscript (i), simply writing u, $\mathbf{g}$, and c_u.

A First Attempt that Does Not Work. The idea of the protocol is to break u and $\mathbf{g}$ into two halves, writing $u = u_L \circ u_R$ and $\mathbf{g} = \mathbf{g}_L \circ \mathbf{g}_R$, where $\circ$ denotes concatenation. Then

$$\langle u, \mathbf{g} \rangle = \langle u_L, \mathbf{g}_L \rangle + \langle u_R, \mathbf{g}_R \rangle. \tag{14.10}$$

Suppose the verifier chooses a random $\alpha \in \mathbb{F}_p$ and define

$$u' = \alpha u_L + \alpha^{-1} u_R \tag{14.11}$$

and

$$\mathbf{g}' = \alpha^{-1} \mathbf{g}_L + \alpha \mathbf{g}_R. \tag{14.12}$$

Note that the verifier $\mathcal{V}$ can compute $\mathbf{g}'$ on its own since it knows $\mathbf{g}_L$ and $\mathbf{g}_R$ (but just as $\mathcal{V}$ does not know u, $\mathcal{V}$ also does not know u'). One might hope that for any choice of $\alpha \in \mathbb{F}_p$,

$$\langle u', \mathbf{g}' \rangle = \langle u, \mathbf{g} \rangle, \tag{14.13}$$

and moreover that the only way an efficient party can compute a u' satisfying Equation (14.13) is to know a u satisfying Equation (14.9) and then set $u' = \alpha u_L + \alpha^{-1} u_R$ as per Equation (14.11). If this were the case, then the prover's original claim, to know a u such that $\langle u, \mathbf{g} \rangle = c_u$, would be *equivalent* to the claim of knowing a u' such that $\langle u', \mathbf{g}' \rangle = c_u$. This would mean that the verifier would have (with no help whatsoever from the prover) successfully reduced the prover's original claim about knowing u to an equivalent claim of the same form, but about vectors of half the length.

The Actual Equality. Unfortunately, Equation (14.13) does not hold. But the following modification does, for any $\alpha \in \mathbb{F}_p$:

$$\begin{aligned}\langle u', \mathbf{g}' \rangle &= \langle \alpha u_L + \alpha^{-1} u_R, \alpha^{-1} \mathbf{g}_L + \alpha \mathbf{g}_R \rangle \\ &= \langle \alpha u_L, \alpha^{-1} \mathbf{g}_L \rangle + \langle \alpha^{-1} u_R, \alpha \mathbf{g}_R \rangle + \langle \alpha u_L, \alpha \mathbf{g}_R \rangle \\ &\quad + \langle \alpha^{-1} u_R, \alpha^{-1} \mathbf{g}_L \rangle \\ &= (\langle u_L, \mathbf{g}_L \rangle + \langle u_R, \mathbf{g}_R \rangle) + \alpha^2 \langle u_L, \mathbf{g}_R \rangle + \alpha^{-2} \langle u_R, \mathbf{g}_L \rangle \\ &= \langle u, \mathbf{g} \rangle + \alpha^2 \langle u_L, \mathbf{g}_R \rangle + \alpha^{-2} \langle u_R, \mathbf{g}_L \rangle. \end{aligned} \tag{14.14}$$

Here, the first equality uses the definitions of u' and $\mathbf{g}'$ (Equations (14.11) and (14.12)) and the final equality uses Equation (14.10). Relative to the hoped-for Equation (14.13) (which does not actually hold), Expression (14.14) involves "cross terms" $\alpha^2 \langle u_L, \mathbf{g}_R \rangle + \alpha^{-2} \langle u_R, \mathbf{g}_L \rangle$. The verifier $\mathcal{V}$ does not know these cross-terms, since they depend on the vectors u_L and u_R that are unknown to $\mathcal{V}$. In the actual protocol, $\mathcal{P}$ will simply send values v_L and v_R to $\mathcal{V}$ claimed to equal $\langle u_L, \mathbf{g}_R \rangle$ and $\langle u_R, \mathbf{g}_L \rangle$ *before learning the random value* α chosen by the verifier. If v_L and v_R are as claimed, then this allows $\mathcal{V}$ to compute the right hand side of Equation (14.14) (let's call it $c_{u'}$), and the prover can, in the next round, turn to proving knowledge of a u' such that $\langle u', \mathbf{g}' \rangle$ equals $c_{u'}$.

Self-Contained Protocol Description. Recall that at the start of the round, the prover has already sent a value c_u claimed to equal $\langle u, \mathbf{g} \rangle$. If u and $\mathbf{g}$ both have length 1, then establishing that the prover knows a u such that $\langle u, \mathbf{g} \rangle = c_u$ is equivalent to establishing knowledge of a discrete logarithm of c_u to base $\mathbf{g}$, which the prover can achieve by simply sending u to $\mathcal{V}$.[8] Otherwise, the protocol proceeds of follows: The prover starts by sending values v_L, v_R claimed to equal the cross terms $\langle u_L, \mathbf{g}_R \rangle$ and $\langle u_R, \mathbf{g}_L \rangle$. At that point the verifier chooses $\alpha \in \mathbb{F}_p$ at random and sends it to the prover.

Let $c_{u'} = c_u + \alpha^2 v_L + \alpha^{-2} v_R$. This value is specifically defined so that if v_L and v_R are as claimed, then $\langle u', \mathbf{g}' \rangle = c_{u'}$. Furthermore, the verifier can compute $\mathbf{g}'$ and $c_{u'}$ given c_u, α, v_L, and v_R. Accordingly, the

[8] For simplicity, we do not concern ourselves during this warmup with designing a zero-knowledge protocol; if we did want to achieve zero-knowledge, we would use Schnorr's protocol (Section 12.2.2) for the prover to establish knowledge of the discrete logarithm of c_u. See Section 14.4.2 for details.

next round of the protocol is then meant to establish that the prover indeed knows a vector u' such that

$$\langle u', \mathbf{g}' \rangle = c_{u'}. \tag{14.15}$$

This is exactly the type of claim that the protocol was meant to establish, but on vectors of length $n/2$ rather than n, so the protocol verifies this claim recursively. See Protocol 12 for pseudocode.

Protocol 12 A public-coin zero-knowledge argument of knowledge of an opening for a generalized Pedersen commitment c_u to a vector u of length n. The protocol consists of $\log_2 n$ rounds and 2 group elements communicated from prover to verifier per round, and satisfies knowledge-soundness assuming hardness of the discrete logarithm problem. For simplicity, we omit the blinding factor from the Pedersen commitment to u and treat the group $\mathbb{G}$ over which the commitments are defined as an additive group.

1: Let $\mathbb{G}$ be an *additive* cyclic group of prime order p over which the Discrete Logarithm relation is hard, with vector of generators $\mathbf{g} = (g_1, \ldots, g_n)$.
2: Input is $c_u = \mathsf{Com}(u) := \sum_{i=1}^{n} u_i g_i$. Prover knows u, Verifier only knows c_u, $g_1, \ldots, g_n$.
3: If $n = 1$, Prover sends u to the verifier and the verifier checks that $ug_1 = c_u$.
4: Otherwise, write $u = u_L \circ u_R$ and $\mathbf{g} = \mathbf{g}_L \circ \mathbf{g}_R$. Prover sends v_L, v_R claimed to equal $\langle u_L, g_R \rangle$ and $\langle u_R, g_L \rangle$.
5: Verifier responds with a randomly chosen $\alpha \in \mathbb{F}_p$.
6: Recurse on commitment $c_{u'} := c_u + \alpha^2 v_L + \alpha^{-2} v_R$ to vector $u' = \alpha u_L + \alpha^{-1} u_R$ of length $n/2$, using the vector of group generators $\mathbf{g}' := \alpha^{-1}\mathbf{g}_L + \alpha \mathbf{g}_R$.

Costs. It is easy to see that u' and $\mathbf{g}'$ have half the length of u and $\mathbf{g}$, and hence the protocol terminates after $\log_2 n$ rounds, with only two group elements sent by the prover to the verifier in each round. Both the prover and verifier's runtimes are dominated by the time required to update the generator vector in each round. Specifically, to

compute $\mathbf{g}'$ from $\mathbf{g}$ in each round, the verifier performs a number of group exponentiations proportional to the length of $\mathbf{g}$, which means the total number of group exponentiations is $O(n + n/2 + n/4 + \cdots + 1) = O(n)$.[9] Hence's the prover and verifier's runtime over the entire protocol is proportional to the time required to perform $O(n)$ group exponentiations.[10]

Completeness and Intuition for Knowledge-Soundness. The protocol is clearly complete, i.e., if the prover is honest (meaning that c_u indeed equals $\langle u, \mathbf{g}\rangle$ and v_L and v_R are as claimed), then indeed Equation (14.15) holds.

To explain the intuition for why knowledge-soundness holds, let us assume for the moment that the prover *does* know a vector u such that $c_u = \langle u, \mathbf{g}\rangle$, but the prover sends values v_L and v_R that are *not* equal to $\langle u_L, \mathbf{g}_R\rangle$ and $\langle u_R, \mathbf{g}_L\rangle$. Then, as we explain in the following paragraph, with high probability over the choice of α, Equation (14.15) will fail to hold. In this event, it is not clear how the prover will find a vector whose inner product with $\mathbf{g}'$ equals $c_{u'}$.

That Equation (14.15) fails to hold with high probability over the choice of α follows by the following reasoning. Let Q be the degree-4 polynomial

$$Q(\alpha) = \alpha^2 c_{u'} = \alpha^2 c_u + \alpha^4 v_L + v_R$$

and

$$P(\alpha) = \alpha^2 \cdot \langle u', \mathbf{g}'\rangle = \alpha^2 c_u + \alpha^4 \langle u_L, \mathbf{g}_R\rangle + \langle u_R, \mathbf{g}_L\rangle.$$

If v_L and v_R are not equal to $\langle u_L, \mathbf{g}_R\rangle$ and $\langle u_R, \mathbf{g}_L\rangle$, then Q and P are not the same polynomial. Since they both have degree at most 4, with probability at least $1 - 4/p$ over the random choice of α, $Q(\alpha) \neq P(\alpha)$.

[9] The terminology "group exponentiation" here, while standard, may be confusing because in this section we are referring to $\mathbb{G}$ as an additive group, while the terminology refers to a multiplicative group. In the additive group notation of this section, we are referring to taking a group element and multiplying it by α or α^{-1}. The same operation in multiplicative group notation would be denoted by raising the group element to the power α or α^{-1}, hence our use of the term group exponentiation.

[10] Actually, it is possible to optimize the verifier's computation in Bulletproofs to perform one multi-exponentiation of length $O(n)$ rather than $O(n)$ independent group exponentiations, enabling a speedup due to Pippenger's algorithm (Section 14.2) of roughly a factor of $O(\log n)$ group operations.

In this event, $c_{u'} \neq \langle u', \mathbf{g}' \rangle$, and hence the prover is left to prove a false claim in the next round.

The above line of reasoning suggests that a prover who knows an opening u of c_u should not be able to convince the verifier to accept with non-negligible probability if the prover does not behave as prescribed in each round (i.e., if sending commitments to values v_L and v_R not equal to $\langle u_L, \mathbf{g}_R \rangle$ and $\langle u_R, \mathbf{g}_L \rangle$). And if the prover does *not* know a u such that $\langle u, \mathbf{g} \rangle$, then, intuitively, the prover should be even worse off than knowing such a u but attempting to deviate from the prescribed protocol.

However, to formally establish knowledge-soundness, we must show that given any prover $\mathcal{P}$ that convinces the verifier to accept with non-negligible probability, there is an efficient algorithm to extract an opening u of c_u from $\mathcal{P}$. This requires a more involved analysis.

Proof of Knowledge-Soundness. Recall that in Section 12.2.3, we established the knowledge-soundness of any Σ-protocol via a two-step analysis. First, we showed that from any convincing prover for the Σ-protocol, one can efficiently extract a pair of accepting transcripts (a, e, z) and (a, e', z') that share the same first message a, but for which the verifier's challenges e and e' differ. Second, by special soundness of any Σ-protocol, there is an efficient procedure to extract a witness from any such pair of transcripts.

The first step of this analysis is called a *forking lemma*. This name comes from the procedure to obtain the pair of transcripts: one runs the prover once to (hopefully) produce an accepting transcript (a, e, z), then rewinds the prover to immediately after it sent its first message a, and "restarts it" with a different verifier challenge e'. This (hopefully) yields a second accepting transcript (a, e', z). One thinks of this as "forking" the protocol into two different executions after the prover's first message a is sent.

The analysis establishing knowledge-soundness of Protocol 12 follows a similar two-step paradigm. In the first step, a generalized forking lemma is proved for multi-round protocols such as Protocol 12. The lemma shows that given any convincing prover for the protocol, one can extract a collection of accepting transcripts whose messages overlap in

a manner analogous to how (a, e, z) and (a, e', z) above share the same first message, a. In the second step, an efficient procedure is given to extract a witness from any such tree of transcripts.

Step 1: A Forking Lemma for Multi-Round Protocols. First, we argue that there is a polynomial-time extraction algorithm $\mathcal{E}$ that, given any prover $\mathcal{P}$ for Protocol 12 that convinces the verifier to accept with non-negligible probability, constructs a 3-transcript-tree $\mathcal{T}$ for the protocol with non-negligible probability. Here, a 3-transcript tree is a collection of $|\mathcal{T}| = 3^{\log_2 n} \leq n^{1.585}$ accepting transcripts for the protocol, with the following relationship between the prover messages and verifier challenges in each transcript.

The transcripts correspond to the leaves of a complete tree where each non-leaf node has 3 children. The depth of the tree equals the number of verifier challenges sent in Protocol 12, which is $\log_2 n$. Each edge of the tree is labeled by a verifier challenge, and each non-leaf node is associated with a prover message (v_L, v_R). That is, if an edge of the tree connects a node at distance i from the root to a node at distance $i + 1$, then the edge is labelled by a value for the ith message that the verifier sends to the prover in Protocol 12. It is required that (a) no two edges of the tree are assigned the same label and (b) for each transcript at a leaf of the tree, the verifier's challenges in that transcript are given by the labels assigned to the edges of the root-to-leaf path for that leaf, and the prover's messages in the transcript are given by the prover responses associated with the nodes along the path.[11]

The idea is to generate the first leaf of the tree by running the prover and verifier once to (hopefully) generate an accepting transcript. Then generate that leaf's sibling by rewinding the prover until just before the verifier sends its last challenge, and restart the protocol with a fresh random value for the verifier's final challenge. Then to generate the next leaf, rewind the prover again until just before the verifier sends its second to last challenge, and restart the protocol from that point with a fresh random value for the verifier's second to last challenge.

[11] For comparison, recall that special soundness of Σ-protocols refers to a pair of accepting transcript (a, e, z) and (a, e', z) with $e \neq e'$. Such a pair of transcripts forms a "2-transcript tree" for a protocol consisting of a single verifier challenge.

And so on. Some complications arise to account for the possibility that the prover sometimes fails to convince the verifier to accept, and the (unlikely) possibility that this process leads to two edges labeled with the same value.

We provide a formal statement and proof of this result below; our presentation follows [66, Lemma 1].

Theorem 14.1. There is a probabilistic extractor algorithm $\mathcal{E}$ satisfying the following property. Given the ability to repeatedly run and rewind a prover $\mathcal{P}$ for Protocol 12 that causes the verifier to accept with probability at least ε for some non-negligible quantity ε, $\mathcal{E}$ runs in expected time at most $\text{poly}(n)$, and $\mathcal{E}$ outputs a 3-transcript tree $\mathcal{T}$ for Protocol 12 with probability at least $\varepsilon/2$.

Proof. $\mathcal{E}$ is a recursive procedure that constructs $\mathcal{T}$ in depth-first fashion. Specifically, $\mathcal{E}$ takes as input the identity of a node j in $\mathcal{T}$, as well as the verifier challenges associated with the edges along the path in $\mathcal{T}$ connecting j to the root, and the prover messages associated with the nodes along that path. $\mathcal{E}$ then (attempts to) produce the subtree of $\mathcal{T}$ rooted at j. (In the very first call to $\mathcal{E}$, j is the root node of $\mathcal{T}$, so in this case there are no edges or nodes along the path of the j to the root, i.e., itself).

If j is a leaf node, the input to $\mathcal{E}$ specifies a complete transcript for Protocol 12, so $\mathcal{E}$ simply outputs the transcript if it is an accepting transcript, and otherwise it outputs "fail".

If j is not a leaf node of $\mathcal{T}$, then the input to $\mathcal{E}$ specifies a partial transcript for Protocol 12 (if j has distance ℓ from the root, then the partial transcript specifies the prover messages and verifier challenges from the first ℓ rounds of Protocol 12). The first thing $\mathcal{E}$ does is associate a prover message with j by "running" $\mathcal{P}$ on the partial transcript to see how $\mathcal{P}$ would respond to the most recent verifier challenge in this partial transcript.

Second, $\mathcal{E}$ attempts to construct the subtree rooted at the left-most subchild of j, which we denote by j'. Specifically, $\mathcal{E}$ chooses a random verifier challenge to assign the edge (j, j') of $\mathcal{T}$, and then calls itself recursively on j'. If $\mathcal{E}$'s recursive call on j' returns "fail" (i.e., it fails to generate the subtree of $\mathcal{T}$ rooted at j'), then $\mathcal{E}$ halts and outputs "fail".

Otherwise, $\mathcal{E}$ proceeds to generate the subtrees of the remaining two children j'' and j''' of j by assigning fresh random verifier challenges to the edges connecting j to those nodes and calling itself recursively on j'' and j''' until it successfully generates these two subtrees (this may require many repetitions of the recursive calls, as $\mathcal{E}$ will simply keep calling itself on j'' and j''' until it finally succeeds in generating these two subtrees).

Expected Running Time of $\mathcal{E}$. Recall that when $\mathcal{E}$ is called on a non-leaf node j, it recursively calls itself once on the first child j' of j in an attempt to construct the subtree rooted at j', and then continues to construct the subtrees rooted at its other two children only if the recursive call on j' succeeds. Let ε' denote this probability. Then the expected number of recursive calls is $1+\varepsilon' \cdot 2/\varepsilon' = 3$. Here, the first term, 1, comes from the first recursive call, on j'. The first factor of ε' in the second term denotes the probability that $\mathcal{E}$ does not halt after the first recursive call. Finally, the factor $2/\varepsilon'$ captures the expected number of times $\mathcal{E}$ must be called on j'' and j''' before it succeeds in constructing the subtree rooted at these nodes (as $1/\varepsilon'$ is the expected value of a geometric random variable with success probability ε'). Meanwhile, when $\mathcal{E}$ is called on a leaf node, it simply checks whether or not the associated transcript is an accepting transcript, which requires $\mathrm{poly}(n)$ time. We conclude that the total runtime of $\mathcal{E}$ is proportional to the number of leaves (which is $3^{\log_2 n} \leq O(n^{1.585})$), times the runtime of the verifier in Protocol 12, which is clearly $\mathrm{poly}(n)$.

Success Probability of $\mathcal{E}$. The initial call to $\mathcal{E}$ on the root of $\mathcal{T}$ returns "fail" if and only if the very first recursive call made by *every* invocation of $\mathcal{E}$ in the call stack returns "fail". That is, $\mathcal{E}$ succeeds in outputting a tree of accepting transcripts when called on the root whenever its recursive call on the first child j of the root succeeds, which itself succeeds whenever its recursive call on the first child of j succeeds, and so forth. This probability is exactly the probability $\mathcal{P}$ succeeds in convincing the verifier to accept, namely ϵ.

We still need to argue that the probability that conditioned on $\mathcal{E}$ successfully outputting a tree, the probability that $\mathcal{E}$ assigns any two

edges in the graph the same challenge by $\mathcal{E}$ is negligible. To argue this, let us assume that $\mathcal{E}$ never runs for more than T time steps, for $T = p^{1/3}$. Here, p denotes the order of $\mathbb{G}$, and hence the size of the verifier's challenge space. We can ensure this by having $\mathcal{E}$ halt and output "fail" if it surpasses T time steps—by Markov's inequality, since $\mathcal{E}$ runs in expected time $\text{poly}(n)$, the probability $\mathcal{E}$ exceeds T timesteps is at most $\text{poly}(n)/T$, which is negligible assuming p is superpolynomially large in n. Hence, after ensuring this assumption holds, the probability $\mathcal{E}$ succeeds in outputting a tree of accepting transcripts is still at most ε minus a negligible quantity. If $\mathcal{E}$ never runs for more than T timesteps, then it only can only generate at most T random challenges of the verifier over the course of its execution. The probability of a collision amongst these at most T challenges is bounded above by $T^2/p \leq 1/p^{1/3}$, which is negligible. We conclude as desired that the probability $\mathcal{E}$ output a 3-transcript tree is at least ε minus a negligible quantity, which is at least $\varepsilon/2$ if ε is non-negligible. □

Step 2: Extracting a Witness from Any 3-Transcript Tree. Second, we must give a polynomial time algorithm that takes as input a 3-transcript tree for Protocol 12 and outputs a vector u such that $c_u = \sum_{i=1}^{n} u_i g_i$. The idea for how this is done is to iteratively compute a label u for each node in the tree, starting with the leaves and working layer-by-layer towards the root. For each node in the tree, the procedure will essentially reconstruct the vector u that the prover must have "had in its head" at that stage of the protocol's execution. That is, each node in the tree is associated with a vector of generators $\mathbf{g}'$ and a commitment c, and the extractor will identify a vector u' such that $\langle u', \mathbf{g}' \rangle = c$.

Associating a Generator Vector and Commitment with Each Node in the Tree. For any node in the tree, we may associate with that node a generator vector and commitment in the natural way. That is, Protocol 12 is recursive, and each node in the tree at distance i from the root corresponds to a call to Protocol 12 at depth i of the call stack. As per Line 2, the verifier in each recursive call to Protocol 12 is aware of a generator vector and a commitment c (supposedly a commitment to some vector known to the prover, using the generator vector).

For example, the root of the tree is associated with $\mathbf{g} = \mathbf{g}_L \circ \mathbf{g}_R$ and commitment $c = c_u$ that is input to the original call to Protocol 12. If the root is associated with prover message (v_L, v_R), then a child connected to the root by an edge of label α is associated with vector $\mathbf{g}' = \alpha^{-1}\mathbf{g}_L + \alpha\mathbf{g}_R$ and commitment $c' = c_u + \alpha^2 v_L + \alpha^{-2} v_R$, where v_L and v_R denote the prover messages associated with the edge. And so on down the tree.

Assigning a Label to Each Node of the Tree, Starting with the Leaves and Working Toward the Root. Given a 3-transcript tree, begin by labelling each leaf with the prover's final message in the protocol. Because every leaf transcript is accepting, if a leaf is assigned label u, generator g, and commitment c, then we know that $g^u = c$.

Now assume by way of induction that, for each node at distance at most $\ell \geq 0$ from the leaves, if the node is associated with generator vector $\mathbf{g}$ and commitment c, the label-assigning procedure has successfully assigned a label vector u to the node such that $\langle u, \mathbf{g} \rangle = c$. We explain how to extend the procedure to assign such labels to nodes at distance $\ell + 1$ from the leaves.

To this end, consider such a node j and let the associated generator vector be $\mathbf{g} = \mathbf{g}_L \circ \mathbf{g}_R$ and associated commitment be c. For $i = 1, 2, 3$, let $\mathbf{g}_i$ and c_i denote the generator vector and commitment associated with j's ith child, u_i denote the label that has already been assigned to the ith child, and α_i denote the verifier challenge associated with the edge connecting j to its ith child. By construction of the generators and commitment associated with each node in the tree, for each i, the following two equations hold, relating the generators and commitment for node j to those of its children:

$$\mathbf{g}_i = \alpha_i^{-1}\mathbf{g}_L + \alpha_i \mathbf{g}_R \tag{14.16}$$

and

$$c_i = c + \alpha_i^2 v_L + \alpha_i^{-2} v_R. \tag{14.17}$$

Moreover, by the inductive hypothesis, the label-assigning algorithm has ensured that

$$\langle u_i, \mathbf{g}_i \rangle = c_i. \tag{14.18}$$

At an intuitive level, Equation (14.18) identifies a vector u_i "explaining" the commitment c_i of child i in terms of the generator vector $\mathbf{g}_i$, while Equations (14.16) and (14.17) relate c_i and $\mathbf{g}_i$ to c and $\mathbf{g}$. We would like to put all of this information together to identify a vector u "explaining" c in terms of $\mathbf{g}$.

To this end, combining Equations (14.16)–(14.18), we conclude that

$$\langle u_i, \alpha_i^{-1}\mathbf{g}_L + \alpha_i \mathbf{g}_R \rangle = c + \alpha_i^2 v_L + \alpha_i^{-2} v_R,$$

and by applying the distributive law to the left hand side, we finally conclude that:

$$\langle \alpha_i^{-1} u_i, \mathbf{g}_L \rangle + \langle \alpha_i u_i, \mathbf{g}_R \rangle = c + \alpha_i^2 v_L + \alpha_i^{-2} v_R. \tag{14.19}$$

Equation (14.19) "almost" achieves our goal of identify a vector u such that $\langle u, \mathbf{g} \rangle = c$, in the sense that if the "cross terms" $\alpha_i^2 v_L + \alpha_i^{-2} v_R$ did not appear in Equation (14.19) for, say, $i = 1$, then the vector $u = \alpha_1^{-1} u_1 \circ \alpha_1 u_1$ would satisfy $\langle u, \mathbf{g} \rangle = c$. The point of deriving Equation (14.19) not only for $i = 1$, but also for $i = 2$ and $i = 3$ is that we can use the latter two equations to "cancel out the cross terms" from the right hand side of the equation for $i = 1$. Specifically, there exists some coefficients $\beta_1, \beta_2, \beta_3 \in \mathbb{F}_p$ such that

$$\sum_{i=1}^{3} \beta_i \cdot \left(c + \alpha_i^2 v_L + \alpha_i^{-2} v_R \right) = c. \tag{14.20}$$

This follows from the fact that the following matrix is full rank, and hence has the vector $(1, 0, 0)$ in its row-span:

$$A = \begin{bmatrix} 1 & \alpha_1^2 & \alpha_1^{-2} \\ 1 & \alpha_2^2 & \alpha_2^{-2} \\ 1 & \alpha_3^2 & \alpha_3^{-2} \end{bmatrix}. \tag{14.21}$$

One way to see that A is invertible is to directly compute the determinant as $-\frac{(\alpha_1^2-\alpha_2^2)(\alpha_1^2-\alpha_3^2)(\alpha_2^2-\alpha_3^2)}{\alpha_1^2\alpha_2^2\alpha_3^2}$, which is clearly nonzero so long as α_1, α_2, and α_3 are all distinct. Moreover, $(\beta_1, \beta_2, \beta_3)$ can be computed efficiently–in fact, it equals the first row of A^{-1}.

Equation (14.20) combined with Equation (14.19) implies that

$$u = \sum_{i=1}^{3} (\beta_i \cdot \alpha_i^{-1} \cdot u_i) \circ (\beta_i \cdot \alpha_i \cdot u_i) \tag{14.22}$$

satisfies $\langle u, \mathbf{g} \rangle = c$, where $\circ$ denotes concatenation.

In this manner, labels can be assigned to each node in the tree, starting with the leaves and proceeding layer-by-layer towards the root. The label u assigned to the root satisfies $\langle u, \mathbf{g} \rangle = c_u$ as desired.

Example of the knowledge extractor. Although in Protocol 12, $\mathbf{g} \in \mathbb{G}^n$ will be a vector of elements of the elliptic curve group $\mathbb{G}$, and $\langle u, \mathbf{g} \rangle$ will also be a group element, for illustration we give an example of the extraction procedure with group elements replaced by integers.

Suppose that $n = 2$, and the committed vector $u = (u_1, u_2)$ is $(1, 6)$ while the commitment key $\mathbf{g} = (g_1, g_2)$ is $(12, 1)$. Then the commitment c to u is $\langle u, \mathbf{g} \rangle = 1 \cdot 12 + 6 \cdot 1 = 18$. The prescribed prover begins Protocol 12 by sending the cross terms $v_L = 1 \cdot 1 = 1$ and $v_R = 6 \cdot 12 = 72$.

The knowledge extraction procedure is not given the vector u that the prover "had in its head" when producing the commitment c or the cross-terms v_L and v_R. Nonetheless, it needs to identify a vector u such that $\langle u, \mathbf{g} \rangle = 18$. To do so, it first generates a (depth-1) 3-transcript tree for Protocol 12. Suppose the three produced accepting transcripts respectively have verifier challenge $\alpha_1 = 1$, $\alpha_2 = 2$, and $\alpha_3 = 3$. Then the three transcripts (one per leaf of the tree) are respectively associated with the following values:

- For leaf $i = 1$, the verifier computes:
  - $\mathbf{g}' = \alpha_1^{-1} \cdot g_1 + \alpha_1 \cdot g_2 = 1^{-1} \cdot 12 + 1 \cdot 1 = 13$.
  - $c' = c + \alpha_1^2 v_L + \alpha_1^{-2} v_R = 18 + 1 \cdot 1 + 1^{-1} \cdot 72 = 91$.

  Since the leaf captures an accepting transcript, the prover in the final round of the protocol must provide a value u' such that $\langle u', \mathbf{g}' \rangle = 91$, and hence $u' = 91/13 = 7$.

- For leaf $i = 2$, the verifier computes:
  - $\mathbf{g}' = \alpha_2^{-1} \cdot g_1 + \alpha_2 \cdot g_2 = 2^{-1} \cdot 12 + 2 \cdot 1 = 8.$
  - $c' = c + \alpha_2^2 v_L + \alpha_2^{-2} v_R = 18 + 4 \cdot 1 + 4^{-1} \cdot 72 = 40.$

  The prover in the final round of the protocol must provide a value u' such that $\langle u', \mathbf{g}' \rangle = 40$, and hence $u' = 40/8 = 5$.
- For leaf $i = 3$, the verifier computes:
  - $\mathbf{g}' = \alpha_3^{-1} \cdot g_1 + \alpha_3 \cdot g_2 = 3^{-1} \cdot 12 + 3 \cdot 1 = 7.$
  - $c' = c + \alpha_3^2 v_L + \alpha_3^{-2} v_R = 18 + 9 \cdot 1 + 9^{-1} \cdot 72 = 35.$

  The prover in the final round of the protocol must provide a value u' such that $\langle u', \mathbf{g}' \rangle = 35$, and hence $u' = 35/7 = 5$.

For the matrix A defined as per Equation (14.21), the first row of A^{-1} is $(\beta_1, \beta_2, \beta_3)$ where $\beta_1 = -\frac{13}{24}$, $\beta_2 = \frac{8}{3}$, and $\beta_3 = -\frac{9}{8}$. Then given the three values of u' constructed above, the reconstructed vector u from Equation (14.22) has first entry equal to

$$-\frac{13}{24} \cdot 1^{-1} \cdot 7 + \frac{8}{3} \cdot 2^{-1} \cdot 5 - \frac{9}{8} \cdot 3^{-1} \cdot 5 = 1$$

and second entry equal to

$$-\frac{13}{24} \cdot 1 \cdot 7 + \frac{8}{3} \cdot 2 \cdot 5 - \frac{9}{8} \cdot 3 \cdot 5 = 6.$$

Hence, in this example, the extractor successfully reconstructed the vector $u = (1, 6)$ such that $\langle u, \mathbf{g} \rangle = c$.

14.4.2 The Polynomial Commitment Scheme

The preceding section describes a (non-zero-knowledge) argument of knowledge of an opening $u \in \mathbb{F}_p^n$ of a generalized Pedersen commitment c_u, i.e., a u such that $\sum_{i=1}^n u_i \cdot g_i = c_u$. To obtain a (non-zero-knowledge) polynomial commitment scheme, we need to modify this argument of

knowledge to establish not only that

$$\sum_{i=1}^{n} u_i \cdot g_i = c_u, \tag{14.23}$$

but also that

$$\sum_{i=1}^{n} u_i \cdot y_i = v \tag{14.24}$$

for some public vector $y \in \mathbb{F}_p^n$ and $v \in \mathbb{F}_p$ (recall from Section 14.2 that u will be the coefficient vector of the committed polynomial, y will be a vector derived from the point at which the verifier requests to evaluate the committed polynomial, and v will be the claimed evaluation of the polynomial).

The idea is that Equations (14.23) and (14.24) are of exactly the same form, namely they both involve computing the inner product of u with another vector (though each g_i is a group element in $\mathbb{G}$, while each y_i is a field element in $\mathbb{F}_p$). So one can simply run two parallel invocations of Protocol 12, using the same verifier challenges in both, but with the second instance replacing the vector of group generators $\mathbf{g}$ with the vector y, and the group element c_u with the field element v. See Figure 13 for a complete description of the protocol.

Sketch of How to Achieve Zero-Knowledge. To render Protocol 13 zero-knowledge, one can apply commit-and-prove style techniques. This means that in every round, the prover does not send v'_L and v'_R to the verifier in the clear, but rather sends Pedersen commitments to these quantities (if one wants perfect rather than computational zero-knowledge, then a blinding factor h^z for randomly chosen z should be included in the Pedersen commitments as per Protocol 5; likewise, the group elements v_L and v_R sent in each round should be blinded as well). At the very end of the protocol, the prover proves in zero-knowledge that the committed values sent over the course of the $\log_2 n$ rounds of the protocol would have passed the check performed by the verifier in the final round of Protocol 13 (Line 3).[12]

[12] Bulletproofs [80] contains an optimization that reduces the number of commitments sent by the prover in each round from 4 to 2, by effectively compressing the

Protocol 13 Extending Protocol 12 to an evaluation-proof for a polynomial commitment scheme, where u is the coefficient vector of the committed polynomial q, and c_u is the commitment to the polynomial. If the verifier requests the evaluation $q(z)$, then v denotes the claimed evaluation and y denotes the vector such that $q(z) = \langle u, y \rangle$. Note that in applications of polynomial commitment schemes to succinct arguments, the evaluation point z, and hence y and v, are typically not chosen by the verifier until after the prover sends the commitment c_u.

1: Let $\mathbb{G}$ be an *additive* cyclic group of prime order p over which the Discrete Logarithm relation is hard, with vector of generators $\mathbf{g} = (g_1, \ldots, g_n)$. Let $y \in \mathbb{F}_p^n$ be a public vector and public value $v \in \mathbb{F}_p$.
2: Input is $c_u = \mathsf{Com}(u) := \sum_{i=1}^{n} u_i g_i$. Prover knows u, Verifier only knows c_u, $\mathbf{g}$, y, and v.
3: If $n = 1$, the prover sends u to the verifier and the verifier checks that $ug_1 = c_u$ and that $uy_1 = v$.
4: Otherwise, write $u = u_L \circ u_R$, $\mathbf{g} = \mathbf{g}_L \circ \mathbf{g}_R$, and $y = y_L \circ y_R$. Prover sends v_L, v_R claimed to equal $\langle u_L, g_R \rangle$ and $\langle u_R, g_L \rangle$, as well as v'_L, v'_R claimed to equal $\langle u_L, y_R \rangle$ and $\langle u_R, y_L \rangle$.
5: Verifier responds with a randomly chosen $\alpha \in \mathbb{F}_p$.
6: Recurse on commitment $c_{u'} := c_u + \alpha^2 v_L + \alpha^{-2} v_R$ to vector $u' = \alpha u_L + \alpha^{-1} u_R$ of length $n/2$, using the vector of group generators $\mathbf{g}' := \alpha^{-1}\mathbf{g}_L + \alpha \mathbf{g}_R$, and using public vector $y' := \alpha^{-1} y_L + \alpha y_R$ and public value $v' = v + \alpha^2 v'_L + \alpha^{-2} v'_R$.

Non-interactive protocol via Fiat-Shamir. Protocols 12 and 13 are public coin, and hence can be rendered non-interactive using the Fiat-Shamir transformation. Despite Bulletproofs being a super-constant round protocol, recent work has placed tight bounds on the concrete knowledge-soundness of the resulting non-interactive protocol in the random oracle model [14], [249] (these analyses apply more generally to any protocol satisfying a generalization of special-soundness (Section 12.2.1) to multi-round settings, namely that a valid witness can be

two commitments v_L and v'_L into a single commitment, and similarly for v_R and v'_R. This is the primary optimization in Bulletproofs [80] over earlier work [66].

extracted from an appropriate transcript tree). This yields an extractable polynomial commitment scheme in which evaluation proofs are non-interactive.

Dory: Reducing Verifier Time To Logarithmic. Recall that the Bulletproofs polynomial commitment scheme achieves constant commitment size, and evaluation proofs consisting of $O(\log n)$ group elements, but both the prover and verifier had to perform $\Theta(n)$ exponentiations in the group $\mathbb{G}$. Lee [179] showed how to reduce the verifier's runtime to $O(\log n)$ group exponentiations, following a one-time setup phase costing $O(n)$ group exponentiations. Lee naming the resulting commit scheme Dory.

More precisely, the setup phase in Dory produces a logarithmic-sized "verification key" derived from the length-n public vector of group generators $\mathbf{g}$ such that any party with the verification key can implement the verifier's checks in only $O(\log n)$ rather than $O(n)$ group exponentiations. One can think of the verification key as a small "summary" of the public vector $\mathbf{g}$ that suffices to implement the verifier's check in $O(\log n)$ time, in the sense that once the verifier knows the verification key, it can forget the actual generators that the key summarizes.

Note that, unlike the KZG polynomial commitments that we cover in Section 15.2, this pre-processing phase is *not* what is called a *trusted setup*, which refers to a pre-processing phase that produces "toxic waste" (also called a trapdoor) such that any party with the trapdoor can break binding of the polynomial commitment scheme. That is, while the setup phase in Dory produces a structured verification key (meaning the key does not consist of randomly chosen group elements), there is no trapdoor, and anyone willing to invest the computational effort can derive the key. Protocols such as Dory that do not require a trusted setup are often called *transparent*. We defer detailed coverage of Dory to Section 15.4 because it uses pairing-based cryptography, a topic we introduce in Section 15.

Combining Techniques. The protocol of Section 14.3 that leveraged the polynomial commitment scheme of Section 14.2 as a subroutine can

replace the subroutine with any extractable additively-homomorphic vector-commitment scheme supporting inner product queries, including Bulletproofs or Dory. If combined with Bulletproofs, the resulting scheme reduces the public parameter size of Bulletproofs from n to $\Theta(\sqrt{n})$, maintains an evaluation-proof size of $O(\log n)$ group elements, and reduces the number of group exponentiations the verifier has to perform at the end of the protocol from n to $\Theta(\sqrt{n})$. The downside relative to vanilla Bulletproofs is that the size of the commitment increases from one group element to $\Theta(\sqrt{n})$ group elements.

If combined with Dory, the resulting scheme does not asymptotically reduce any costs relative to Dory alone, but does reduce constant factors in the prover's runtime and the runtime of the pre-processing phase [179], [230]. It is actually possible to combine the idea of Section 14.3 with techniques from Dory in a way that keeps the commitment size one group element instead of $O(\sqrt{N})$ group elements, see Section 15.4.5—this combination is what we refer to as Dory in Table 16.1, which summarizes the costs of the transparent polynomial commitments that we have covered.

In Section 16.4, we briefly describe additional polynomial commitment schemes inspired by similar techniques, but based on cryptographic assumptions other than hardness of the discrete logarithm problem.