15

Polynomial Commitments from Pairings

This section explains how to use a cryptographic primitive called pairings (also referred to as bilinear maps) to give polynomial commitment schemes with different cost profiles that those of the previous section. The two major pairing-based schemes that we cover are called KZG commitments and Dory.

KZG commitments are named after Kate *et al.* [164], the authors of the work in which they were introduced. A major benefit of them is that commitments and openings consist of only a constant number of group elements. A downside is that it requires a *structured reference string* (SRS) that is as long as the number of coefficients in the polynomial being committed to. This string must be generated in a specified manner and made available to any party that wishes to commit to a polynomial. The generation procedure produces "toxic waste" (also called a trapdoor) that must be discarded. That is, whatever party generates the reference string knows a piece of information that would let the party break the binding property of the polynomial commitment scheme, and thereby destroy soundness of any argument system that uses the commitment scheme. The generation of such an SRS is also called a *trusted setup*.

As discussed in Section 14.4.2, Dory is transparent. This means that, although there is a pre-processing phase that takes time linear in the number of coefficients of the polynomial to be committed, there is no toxic waste produced. However, Dory's proof size and verification time are logarithmic in the number of coefficients, rather than constant.

15.1 Cryptographic Background

The following background material on pairings builds on Section 12.1, which introduced cryptographic groups and the discrete logarithm problem.

The Decisional Diffie-Helman Assumption. The Decisional Diffie-Helman (DDH) assumption in a cyclic group $\mathbb{G}$ with generator g states that, given g^a and g^b for a, b chosen uniformly and independently from $|\mathbb{G}|$, the value g^{ab} is computationally indistinguishable from a random group element. Formally, the assumption is that the following two distributions cannot be distinguished, except for negligible advantage over random guessing, by any efficient algorithm:

- (g, g^a, g^b, g^{ab}) where a and b are chosen uniformly at random from $\{0, \ldots, |\mathbb{G}| - 1\}$ and g from $\mathbb{G}$.
- (g, g^a, g^b, g^c) where a, b, and c are chosen uniformly at random from $\{0, \ldots, |\mathbb{G}| - 1\}$ and g from $\mathbb{G}$.

If one could compute discrete logarithms efficiently in $\mathbb{G}$, then one could break the DDH assumption in that group: given as input a triple of group elements (g, g_1, g_2, g_3), one could compute the discrete logarithms a, b, c of g_1, g_2, g_3 in base g, and check whether $c = a \cdot b$, outputting "yes" if so. This algorithm would always output yes under draws from the first distribution above, and output yes with probability just $1/|\mathbb{G}|$ under draws from the second distribution.

Hence, the DDH assumption is a stronger assumption than hardness of the Discrete Logarithm problem. In fact, there are groups in which the DDH assumption is false, yet the discrete logarithm problem is nonetheless believed to be hard.

A close relative of DDH is the *computational Diffie-Helman* (CDH) assumption, which states that given g, g^a, and g^b, no efficient algorithm can compute g^{ab}. CDH is a weaker assumption than DDH in the sense that if one can compute g^{ab} given g, g^a and g^b, then one can also solve the DDH problem of distinguishing (g, g^a, g^b, g^{ab}) from (g, g^a, g^b, g^c) for random $a, b, c \in \mathbb{F}_p$. Given a tuple (g, g^a, g^b, g^c), one simply computes g^{ab} and outputs 1 if and only if $g^c = g^{ab}$.

As we will see, there are groups in which CDH is believed to hold but DDH does not.

Pairing-Friendly Groups and Bilinear Maps. Let $\mathbb{G}$ and $\mathbb{G}_t$ be two cyclic groups of the same order. A map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$ is said to be bilinear if for all $u, v \in \mathbb{G}$ and $a, b \in \{0, \ldots, |\mathbb{G}| - 1\}$, $e(u^a, v^b) = e(u, v)^{ab}$.[1] If a bilinear map e is also non-degenerate (meaning, it does not map all pairs in $\mathbb{G} \times \mathbb{G}$ to the identity element 1_{G_t}) and e is efficiently computable, then e is called a pairing. This terminology refers to the fact that e associates each pair of elements in $\mathbb{G}$ to an element of $\mathbb{G}_t$.

Note that any two cyclic groups $\mathbb{G}$ and $\mathbb{G}_t$ of the same order are in fact isomorphic, meaning there is a bijective mapping $\pi\colon \mathbb{G} \mapsto \mathbb{G}_t$ that preserves group operations, i.e., $\pi(a \cdot b) = \pi(a) \cdot \pi(b)$ for all $a, b \in \mathbb{G}$. But just because $\mathbb{G}$ and $\mathbb{G}_t$ are isomorphic does not mean they are equivalent from a computational perspective; elements of $\mathbb{G}$ and $\mathbb{G}_t$ and the respective group operations can be represented and computed in very different ways.

Not all cyclic groups $\mathbb{G}$ for which the discrete logarithm problem is believed to be hard are "pairing-friendly", i.e., come with a bilinear map e mapping $\mathbb{G} \times \mathbb{G}$ to $\mathbb{G}_t$. For example, as elaborated upon shortly, the popular Curve25519, which is believed to yield an elliptic curve group in which the discrete logarithm problem is hard, is not pairing-friendly. As a result, group operations of pairing-friendly elliptic curves

[1] In general, the domain of a bilinear map typically consists of pairs of elements from two different cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ of the same order as $\mathbb{G}_t$, rather than pairs of elements from the same cyclic group $\mathbb{G}$. In the general case that $\mathbb{G}_1 \neq \mathbb{G}_2$, the pairing is said to be *asymmetric*, while the case that $\mathbb{G}_1 = \mathbb{G}_2$ is called *symmetric*. Asymmetric pairings are much more efficient in practice than symmetric pairings. But for simplicity in this monograph we will primarily consider the symmetric case in which $\mathbb{G}_1 = \mathbb{G}_2$.

tend to be concretely slower than preferred groups that need not be pairing-friendly.

In more detail, in practice, if $\mathbb{G}$ is an elliptic curve group defined over field $\mathbb{F}_p$, then $\mathbb{G}_t$ is typically a multiplicative subgroup of an extension field $\mathbb{F}_{p^k}$ for some positive integer k (recall from Section 2.1.5 that $\mathbb{F}_{p^k}$ denotes the finite field of size p^k). That is, $\mathbb{G}_t$ consists of (a subgroup of) the nonzero elements of $\mathbb{F}_{p^k}$, with the group operation being field multiplication. As the multiplicative subgroup of $\mathbb{F}_{p^k}$ has size $p^k - 1$, and the order $|\mathbb{H}|$ of any subgroup $\mathbb{H}$ of a group $\mathbb{G}'$ divides $|\mathbb{G}'|$, k is chosen to be the smallest integer such that $|\mathbb{G}|$ divides $p^k - 1$; this value of k is called the *embedding degree* of $\mathbb{G}$. To efficiently implement pairings in this manner, $\mathbb{G}$ must have low embedding degree. This is because elements of $\mathbb{F}_{p^k}$ are k times bigger than elements of the field $\mathbb{F}_p$ over which $\mathbb{G}$ is defined, and multiplication within $\mathbb{F}_{p^k}$ is at least k times slower than within $\mathbb{F}_p$. So if k is large, $\mathbb{G}_t$ elements will be much more expensive to write down and operate on than elements of $\mathbb{G}$.[2] There are often ways of reducing this naive representation size of $\mathbb{G}_t$ elements by a constant factor such as 3, with corresponding speed improvements when multiplying two $\mathbb{G}_t$ elements (see, e.g., [196]), but this will not help much if the embedding degree k is enormous.

Unfortunately, popular groups in which the Discrete Logarithm problem is believed intractable, such as Curve25519, have enormous embedding degree. This is why arithmetic in pairing-friendly groups tends to be concretely slower then in non-pairing-friendly groups. At the time a writing, a popular pairing-friendly curve for use in SNARKs is called BLS12-381, which has embedding degree 12 and targets roughly 120 bits of security.[3]

[2] Since $\mathbb{G}_t$ is a subgroup of $\mathbb{F}_{p^k}$ of size only p, information-theoretically speaking each element of $\mathbb{G}_t$ can be uniquely represented with only $\log_2 p$ bits. But there will likely not be an efficient algorithm for performing $\mathbb{G}_t$ operations if using such a space-optimal representation of the $\mathbb{G}_t$ elements.

[3] See https://electriccoin.co/blog/new-snark-curve/ and https://hackmd.io/@benjaminion/bls12-381 for discussion of BLS12-381. As a general ballpark, operations in this pairing-friendly group may be about 4× slower than in Curve25519—of course, a precise comparison will depend on implementation details and the hardware on which a comparison is performed. BLS12-381 was designed to work over a field

Note that in any group $\mathbb{G}$ equipped with a symmetric pairing, the Decisional Diffie-Hellman assumption does not hold. This is because one can distinguish triples (g, g_1, g_2, g_3) of the form $(g, g_1 = g^a, g_2 = g^b, g_3 = g^{ab})$ from $(g, g_1 = g^a, g_2 = g^b, g_3 = g^c)$ for randomly chosen $c \in \{0, \ldots, |\mathbb{G}| - 1\}$ by checking whether $e(g, g_3) = e(g_1, g_2)$. In the case where $g_3 = g^{ab}$, this check will always pass by bilinearity of e, while if e is non-degenerate, this check will fail with overwhelming probability if g_3 is a random group element in $\mathbb{G}$. Nonetheless, even in groups equipped with a symmetric pairing, it is often assumed that the *computational* Diffie-Hellman assumption holds.

Intuition for Why Bilinear Maps are Useful. Recall that an additively homomorphic commitment scheme such as Pedersen commitments allows any party to perform addition "underneath commitments". That is, despite the fact that the commitments perfectly hide the value that is committed, it is possible for anyone to take two commitments c_1, c_2 to values m_1, m_2, and compute a commitment c_3 to $m_1 + m_2$, despite not actually knowing anything about m_1 or m_2. However, Pedersen commitments are not multiplicatively homomorphic: while we gave an efficient interactive protocol (Protocol 9) for a prover (that knows how to open c_1 and c_2) to prove that c_3 commits to $m_1 \cdot m_2$, it is not possible for a party that does not know m_1 or m_2 to compute a commitment to $m_1 \cdot m_2$ on its own.

Bilinear maps effectively convey the power of multiplicative-homomorphism, but only for *one* multiplication operation. To be more concrete, let us think of a group element $g^{m_i} \in \mathbb{G}$ as a commitment to m_i (if m_i is chosen at random, the commitment is computationally hiding if the discrete logarithm problem is hard in $\mathbb{G}$, meaning it is hard to determine m_i from g^{m_i}). Then bilinear maps allow any party, given commitments c_1, c_2, c_3 to check whether the values m_1, m_2, m_3 inside the commitments satisfy $m_3 = m_1 \cdot m_2$. This is because by bilinearity of the map $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$, $e(g^{m_1}, g^{m_2}) = e(g^{m_3}, g)$ if and only if $m_3 = m_1 \cdot m_2$.

---

that supports efficient FFT algorithms, so that its use is compatible with SNARKs in which the prover must perform an FFT (see Section 19.3.1 for further dicussion).

It turns out that the power to perform a single "multiplication check" of committed values is enough to obtain a polynomial commitment scheme. This is because Lemma 9.3 implies that for any degree-D univariate polynomial p, the assertion "$p(z) = v$" is equivalent to the assertion that there exists a polynomial w of degree at most $D-1$ such that

$$p(X) - v = w(X) \cdot (X - z). \tag{15.1}$$

Equation (15.1) can be probabilistically verified by evaluating the two polynomials on the left hand side and right hand side at a randomly chosen point τ. This intuitively means that the committer can commit to p by sending a commitment c_3 to $m_3 \coloneqq p(\tau)$, and then convince a verifier that Equation (15.1) holds by sending a commitment c_2 to $m_2 \coloneqq w(\tau)$. If the verifier can compute a commitment c_1 to $m_1 \coloneqq \tau - z$ on its own, then the verifier can use the bilinear map to check that indeed the $m_3 = m_1 \cdot m_2$ (i.e., Equation (15.1) holds at input τ). This entire approach assumes that the committer does not know τ, since if it did, it could choose the polynomial w so that Equation (15.1) does not hold as an equality of polynomials, but does hold at τ.[4]

The following section makes the above high-level outline formal.

15.2 KZG: Univariate Polynomial Commitments from Pairings and a Trusted Setup

A Binding Scheme. Let e be a bilinear map pairing groups $\mathbb{G}, \mathbb{G}_t$ of prime order p, and $g \in \mathbb{G}$ be a generator, and D be an upper bound on the degree of the polynomials we would like to support commitments to. The structured reference string consists of encodings in $\mathbb{G}$ of all powers of a random nonzero field element $\tau \in \mathbb{F}_p$. That is, τ is an integer chosen at random from $\{1, \ldots, p-1\}$, and the SRS equals $(g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^D})$.

[4] One may wonder if one can instead use a non-pairing-friendly group, and use Protocol 9 rather than the bilinear map to check that $m_3 = m_1 \cdot m_2$. This would work, but the proofs would be considerably longer, though still a constant number of group elements. Also, to render the argument system non-interactive, one would need to apply the Fiat-Shamir transformation, forcing the verifier to perform hashing operations. All told, this yields more expensive verification than KZG commitments. The raison d'être of KZG commitments is the remarkable efficiency of evaluation-proof verification.

The value τ is toxic waste that must be discarded because it can be used to destroy binding.

To commit to a polynomial q over $\mathbb{F}_p$, the committer sends a value c claimed to equal $g^{q(\tau)}$. Note that while the committer does not know τ, it is still able to compute $g^{q(\tau)}$ using the SRS and additive homomorphism: if $q(Z) = \sum_{i=0}^{D} c_i Z^i$, then $g^{q(\tau)} = \prod_{i=0}^{D} (g^{\tau^i})^{c_i}$, which can be computed given the values g^{τ_i} for all $i = 0, \ldots, D$ even without knowing τ.[5]

To open the commitment at input $z \in \{0, \ldots, p-1\}$ to some value v, i.e., to prove that $q(z) = v$, the committer computes a "witness polynomial"

$$w(X) \coloneqq (q(X) - v)/(X - z),$$

and sends a value y claimed to equal $g^{w(\tau)}$ to the verifier. Again, since w has degree at most D, $g^{w(\tau)}$ can be computed from the SRS despite the fact that the prover does not know τ. The verifier checks that

$$e(c \cdot g^{-v}, g) = e(y, g^{\tau} \cdot g^{-z}). \tag{15.2}$$

Note that this requires the verifier to know c, v, y, z, and g^{τ}. The first three values are provided by the prover, the opening query z is determined by the verifier itself, and g^{τ} is an entry of the SRS. Note that g^{τ} and g are the only entries of the SRS needed for verification. For this reason, some works refer to the entire SRS as the *proving key* and (g, g^{τ}) as the *verification key*, and think of the verifier as only downloading the verification key, not the entire proving key.

Analysis of Correctness and Binding. Correctness is easy to establish: if $c = g^{q(\tau)}$ and $y = g^{w(\tau)}$ then

$$\begin{aligned} e(c \cdot g^{-v}, g) &= e(g^{q(\tau)-v}, g) = e(g^{w(\tau)\cdot(\tau-z)}, g) = e(g^{w(\tau)}, g^{\tau-z}) \\ &= e(y, g^{\tau} \cdot g^{-z}). \end{aligned}$$

[5] As in Section 14.4, one can think of g^{τ^i} as a Pedersen commitment to τ^i (Protocol 5), but without the blinding factor h^z. This yields a commitment that is perfectly binding but at best computationally hiding: g^{τ^i} information-theoretically specifies τ^i, but deriving τ^i from g^{τ^i} requires computing the discrete logarithm of g^{τ^i} to base g. The ability of the committer to compute $g^{q(\tau)}$ from the SRS without knowing τ follows from the fact that this modified Pedersen commitment is additively homomorphic.

Here, the first inequality holds because $c = g^{q(\tau)}$, the second holds by definition of w as $(q(X) - v)/(X - z)$, the third holds by bilinearity of e, and the fourth holds because $y = g^{w(\tau)}$.

The intuition for why binding holds is as follows. If $q(z) \neq v$, then passing the verifier's check (Equation (15.2)) requires computing $g^{w(\tau)}$, where $w(X) = (q(X) - v)/(X - z)$ is *not* a polynomial in X. Rather, it is a polynomial in X, multiplied by the rational function $1/(X - z)$. The SRS, by containing g raised to all positive powers of τ, provides enough information for the prover to evaluate "in the exponent of g" any desired degree-D polynomial at τ, despite not knowing τ. But, intuitively, this information should not be enough to allow the prover to then "divide in the exponent" by $\tau - z$, as appears to be required to compute $g^{w(\tau)} = g^{(q(\tau)-v)/(\tau-z)}$.

To make this intuition precise, we show that binding follows from a cryptographic assumption, called the *D-strong Diffie-Hellman (SDH) assumption*, that essentially just asserts that "dividing in the exponent" by $\tau - z$ is intractable, even for an adversary given the SRS used by the KZG commitment scheme. That is, SDH assumes that, given the SRS that consists of the generator g raised to all powers of τ up to power-D, there is no efficient algorithm $\mathcal{A}$ that outputs a pair $(z, g^{1/(\tau-z)})$ except with negligible probability. The SDH assumption was introduced by Boneh and Boyen [62]. It is closely related to an earlier assumption called the Strong RSA assumption [20], the main difference being that SDH refers to (pairing-friendly) cyclic groups while the Strong RSA assumption refers to the (non-cyclic) groups arising in the RSA cryptosystem.

Note that the SDH assumption in $\mathbb{G}$ implies that the Discrete Logarithm problem is hard in $\mathbb{G}$, because if discrete logarithms are easy to compute, then τ can be efficiently computed from g^{τ}, and given τ and z it is easy to compute $g^{1/(\tau-z)}$. Indeed, this can be done by computing the multiplicative inverse ℓ modulo p of $\tau - z$ using the Extended Euclidean algorithm. Since $\mathbb{G}$ has order p, and hence $g^{ip} = 1_{\mathbb{G}}$ for all integers i, $g^{\ell} = g^{1/(\tau-z)}$.[6]

[6] There are known algorithms that use the group elements g^{τ^i} for $i > 1$ given in the SRS to speed up the computation of τ, relative to the fastest known algorithms that

Formally, to establish binding of KZG commitments assuming SDH, one must show that if one can open a commitment c at point $z \neq \tau$ to two different values v, v', then one can efficiently compute $g^{1/(\tau - z)}$, thereby violating the SDH assumption.

Some More Intuition. Recall that, if $c = g^{q(\tau)}$ and $y = g^{w(\tau)}$, then the verifier's check in Equation (15.2) confirms "in the exponent of g" that $q(\tau) - v = w(\tau) \cdot (\tau - z)$. So opening $c = g^{q(\tau)}$ to two different values v, v' intuitively requires identifying two different exponents $w(\tau)$ and $w'(\tau)$ such that

$$q(\tau) - v = w(\tau) \cdot (\tau - z)$$

and

$$q(\tau) - v' = w'(\tau) \cdot (\tau - z).$$

Subtracting these two equations from each other implies that

$$v' - v = (w(\tau) - w'(\tau))(\tau - z).$$

Since $v - v' \neq 0$, and assuming $\tau \neq z$, one can divide both sides by $(v - v') \cdot (\tau - z)$ to conclude that

$$1/(\tau - z) = (w(\tau) - w'(\tau))/(v - v').$$

Thus, one has solved for $1/(\tau - z)$ "in the exponent" of g, contradicting the SDH assumption. The following analysis makes this formal.

Formal Binding Analysis. To open c to values v and v' the committer must identify values $y, y' \in \mathbb{G}$ such that:

$$e(c \cdot g^{-v}, g) = e(y, g^{\tau - z})$$

and

$$e(c \cdot g^{-v'}, g) = e(y', g^{\tau - z}).$$

For simplicity, let us write $c = g^{r_1}$, $y = g^{r_2}$, and $y' = g^{r_3}$ (although the committer may not know r_1, r_2, or r_3). By bilinearity of e, these

solve for τ given only g and g^{τ}. However, the speedups are modest, i.e., solving for τ given $g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^D}$ is believed to require super-polynomial time for appropriately chosen cryptographic groups. See [98, Appendix A.5] for details.

two equations imply that

$$g^{r_1} \cdot g^{-v} = g^{r_2 \cdot (\tau - z)}$$

and

$$g^{r_1} \cdot g^{-v'} = g^{r_3 \cdot (\tau - z)}.$$

Together, these two equations imply that:

$$g^{v - v'} = g^{(r_3 - r_2)(\tau - z)}.$$

In other words,

$$\left((y' \cdot y^{-1})^{1/(v-v')}\right)^{(\tau - z)} = g. \tag{15.3}$$

Here, $(y' \cdot y^{-1})^{1/(v-v')}$ denotes the value obtained raising the group element $y' \cdot y^{-1} \in \mathbb{G}$ to the power x, where x is the multiplicative inverse of $v - v'$ modulo p; note that x can be computed efficiently (in time $\tilde{O}(\log p)$) via the Extended Euclidean algorithm. Equation (15.3) states that $\left((y' \cdot y^{-1})^{1/(v-v')}\right)$ equals $g^{1/(\tau - z)}$. Since this value can be computed efficiently given v, v', y, y' provided by the committer, the committer must have broken the SDH assumption.

An Extractable Scheme. Recall (Section 7.4) that an extractable polynomial commitment scheme guarantees that for every "efficient committer adversary $\mathcal{A}$" that takes as input the public parameters of the commitment scheme and a degree bound D and outputs a polynomial commitment c, there is an efficient algorithm E (which depends on $\mathcal{A}$) that produces a degree-D polynomial p explaining all of $\mathcal{A}$'s answers to evaluation queries. That is, if $\mathcal{A}$ is able to successfully answer evaluation query z with value v, then $p(z) = v$. Since E is efficient, it cannot know anything more than $\mathcal{A}$ does (since $\mathcal{A}$ can afford to run E), and E clearly knows p by virtue of outputting it. This captures the intuition that $\mathcal{A}$ must "know" a polynomial p that $\mathcal{A}$ is using to answer evaluation queries.

The binding analysis above shows that, once the prover sends a KZG commitment, it is bound to some function. This means that for each possible evaluation query z, there is at most one value v

that the committer can successfully answer the query with. But it *doesn't* establish that the function the prover is bound to is a degree-D polynomial. To make this polynomial commitment scheme extractable rather than simply binding, it must be modified and/or additional cryptographic assumptions are required (see for example the discussion of the Generic Group Model later in this section).

Here is one method of achieving extractability that does require modifying the scheme, as well as an additional cryptographic assumption. Recall that $\mathbb{G}$ is a cyclic group of order p with public generator g. In the modified scheme, the SRS doubles in size. Specifically, for τ and α chosen at random from $\mathbb{F}_p$, the modified SRS consists of the pairs

$$\{(g, g^{\alpha}), (g^{\tau}, g^{\alpha\tau}), (g^{\tau^2}, g^{\alpha\tau^2}), \ldots, (g^{\tau^D}, g^{\alpha\tau^D})\}.$$

That is, the SRS consists not only of powers of τ in the exponent of g, but also the same quantities raised to the power α. Note that neither τ nor α are included in the SRS—they are "toxic waste" that must be discarded after the SRS is generated, as any party that knows these quantities can break extractability or binding of the polynomial commitment scheme.

The Power Knowledge of Exponent (PKoE) assumption [141] posits, roughly, that for any polynomial time algorithm $\mathcal{A}$ given access to the SRS, whenever the algorithm outputs any two group elements $g_1, g_2 \in \mathbb{G}$ such that $g_2 = g_1^{\alpha}$, the algorithm must "know" coefficients $c_1, \ldots, c_D$ that "explain" g_2, in the sense that $g_2 = \prod_{i=1}^{D} g^{c_i \cdot \tau^i}$. The idea is that, given access to the SRS, it is easy to compute a pair (g_1, g_2) with $g_2 = g_1^{\alpha}$ in the following manner: let g_1 equal any product of quantities in the first half of the SRS raised to constant powers, e.g.,

$$g_1 \coloneqq \prod_{i=0}^{D} g^{c_i \tau^i},$$

and let g_2 be the result of applying the same operations to the second half of the SRS, i.e., $g_2 \coloneqq \prod_{i=0}^{D} g^{c_i \alpha \tau^i}$. The PKoE essentially assumes that this is the *only way* that an efficient party is capable of computing two group elements with this relationship to each other. It formalizes this by assuming that, for any efficient adversary $\mathcal{A}$ that takes as input the SRS and produces such pairs of group elements, there is an efficient

procedure E (which can depend on $\mathcal{A}$) that actually produces such c_i values. Since E is efficient, it cannot "know" anymore than A does, and E obviously knows the c_i values.

Whereas in the original commitment scheme, which was binding but not necessarily extractable, the commitment to polynomial q was $g^{q(\tau)}$, in the modified scheme the commitment is the pair $(g^{q(\tau)}, g^{\alpha q(\tau)})$. The committer can compute this pair using the modified SRS. To open a commitment $c = (U, V)$ at $z \in \mathbb{F}_p$ to value y, the committer computes the degree-$(D-1)$ polynomial $w(X) := (q(X) - v)/(X - z)$ and sends a value y claimed to equal $w(\tau)$ exactly as in the original scheme. The verifier checks not only that

$$e(U \cdot g^{-v}, g) = e(y, g^{\tau} \cdot g^{-z}),$$

but also that $e(U, g^{\alpha}) = e(V, g)$.

Completeness for the first check holds exactly as in the unmodified scheme. Completeness for the verifier's second check holds because if U and V are provided honestly then $V = U^{\alpha}$ (despite the fact that neither the prover nor the verifier know α), and hence by bilinearity of e, $e(U, g^{\alpha}) = e(V, g)$.

To prove that the modified scheme is extractable, we use the extractor E whose existence is asserted by the PKoE assumption to construct an extractor $\mathcal{E}$ for the polynomial commitment scheme. Specifically, the second check made by the verifier during opening ensures that $V = U^{\alpha}$, despite the fact that the verifier does not know α. The PKoE assumption therefore asserts the existence of an efficient extraction procedure E that outputs quantities $c_1, \ldots, c_D$ such that $U = \prod_{i=1}^{D} g^{c_i \cdot \tau^i}$. We define the extractor $\mathcal{E}$ to run E to produce these $c_1, \ldots, c_D$ values, and then output the polynomial $s(X) = \sum_{i=1}^{D} c_i X^i$.

Clearly, $g^{s(\tau)} = U$, so (U, V) is indeed a commitment to the polynomial s. In particular, U is a commitment to s under the original unmodified commitment scheme. Since we showed that the original scheme is binding under the SDH assumption, this means that the committer is bound to s in the modified scheme.[7]

[7] The SRS of the modified commitment scheme contains additional group elements $g^{\alpha\tau}, g^{\alpha\tau^2}, \ldots, g^{\alpha\tau^D}$ for a random $\alpha \in \mathbb{F}$. Any adversary $\mathcal{A}$ for the SDH assumption

In more detail, suppose that the committer in the modified scheme is able to open $c = (U, V)$ to value v at point z, and $v \neq s(z)$. Then the committer in the unmodified scheme can in fact open U to both v and $s(z)$. For example, to open U to $s(z)$, the committer can let $w'(X) = (s(X) - s(z))/(X - z)$, which is a polynomial of degree at most $D - 1$, and send $g^{w'(\tau)}$ during the opening procedure. This quantity will pass the verifier's first check by the completeness analysis of the unmodified commitment scheme.

Discussion of the PKoE Assumption. The PKoE assumption is qualitatively different from all other cryptographic assumptions that we have discussed thus far in this monograph, including the DDH and CDH assumptions, the discrete logarithm assumption, and the existence of collision-resistant families of hash functions. Specifically, all of these other assumptions satisfy a property called *falsifiability*. Falsifiability is a technical notion formalized by Naor [197]: a cryptographic assumption is said to be falsifiable if it can be captured by defining an interactive game between a polynomial-time challenger and an adversary, at the conclusion of which the challenger can decide in polynomial-time whether the adversary won the game. A falsifiable assumption must be of the form "every efficient adversary has a negligible probability of winning the game".

For example, the assumption that a hash family is collision-resistant can be modeled by having the challenger send the adversary a hash function h chosen at random from the family, and challenging the adversary to find a collision, i.e., two distinct strings x, y such that $h(x) = h(y)$. Clearly, the challenger can efficiently check whether the adversary won the game, by evaluating h at x and y and confirming that indeed $h(x) = h(y)$. In contrast, a knowledge-of-exponent assumption such as PKoE is not falsifiable: if the adversary computes a pair (g_1, g_1^{α}), it is not clear how the challenger could determine whether the adversary broke the assumption. That is, since the challenger does not have access to

that is given access to the original SRS can efficiently simulate these extra group elements itself by picking α at random and raising every element of the unmodified SRS to the power α. Hence, these extra group elements do not give the SDH adversary any extra power.

the internal workings of the adversary, it is not clear how the challenger could determine whether or not the adversary computed (g_1, g_1^{α}) without the adversary "knowing in its own head" coefficients $c_1, \ldots, c_D$ such that $g_1 = \prod_{i=1}^{D} g^{c_i \cdot \tau^i}$. The issue is that in claiming to have broken the assumption, the adversary is claiming to *not know* certain information, namely, the coefficients $c_1, \ldots, c_D$ of the previous sentence. This entire manuscript is devoted to efficiently proving knowledge to an untrusting party, but there is no way to prove *lack* of knowledge.

Theoretical cryptographers generally prefer falsfiable assumptions because they seem easier to reason about and are arguably more concrete, as there is an efficient process to check whether an adversarial strategy falsifies the assumption. That said, not all falsifiable assumptions are "superior" to all non-falsifiable ones: indeed, some falsifiable assumptions proposed in the research literature have turned out to be false! And cryptographers certainly do believe that the PKoE assumption holds in many groups.

We have presented some succinct *interactive* arguments for circuit satisfiability in this monograph that are based on falsifiable assumptions (e.g., the 4-message argument argument obtained by combining PCPs with Merkle trees from Section 9.2). But none of the *non-interactive* succinct arguments of knowledge (SNARKs) for circuit satisfiability that we present are based on falsifiable assumptions; they are either based on knowledge-of-exponent assumptions such as PKoE, or they are sound in the random oracle model.[8] This is because it is not known how to base a SNARK for circuit satisfiability on a falsifiable assumption, and indeed barriers to achieving this are known [126].

In summary, while assumptions like PKoE are slightly controversial in the theoretical cryptography community, many researchers and practitioners are nonetheless confident in their veracity. It is perhaps reasonable to expect that any given deployed SNARK is more likely

[8] We did explain a succinct non-interactive argument for circuit *evaluation* based on falsifiable assumptions in Section 5.2, by instantiating the Fiat-Shamir transformation of the GKR protocol with a correlation-intractable hash family in place of the random oracle.

to be broken for mundane reasons such as unnoticed flaws in the security proofs or bugs in the implementation, than because the PKoE assumption turns out to be false in the group used by the SNARK.

Generic Group Model and Algebraic Group Model. The unmodified polynomial commitment scheme covered in this section is also known to be extractable in the so-called *Generic Group model* (GGM), as well as in a variant model called the *Algebraic Group model* (AGM) [117]. The Generic Group model is similar in spirit to the random oracle model (see Section 5.1). Recall that the random oracle model models cryptographic hash functions as truly random functions. In contrast, "real-world" implementations of protocols in the random oracle model must instantiate the random oracle with concrete hash functions, and real-world attackers trying to "break" the protocol can try to exploit properties of the concrete hash function. Accordingly, the random oracle model only captures "attacks" that do not exploit any structure in the concrete hash functions. The rationale for why this is reasonable is that real-world cryptographic hash functions are designed to (hopefully) "look random to efficient adversaries"; hence we generally do not know real-world attacks that exploit structure in concrete hash functions, though contrived protocols are known for which no real-world instantiation of the protocol with a concrete hash function is secure.

Similarly, the GGM considers adversaries that are only given access to cryptographic groups $\mathbb{G}, \mathbb{G}_t$ via an oracle that computes the group multiplication operation. The pairing operation $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$ is modeled as an additional oracle. In the real-world, attackers are actually given explicit representations of group elements, as well as efficient computer code that, given the representation of two group elements, computes the representation of the product of those two elements. But we generally do not know of attacks on real-world protocols that exploit these explicit representations.

The AGM is a model that lies in between the GGM and the real world. The AGM has a similar flavor to knowledge-of-exponent assumptions like PKoE, in that it assumes whenever an efficient algorithm $\mathcal{A}$ outputs a "new" group element $g \in \mathbb{G}$, it also outputs an "explanation" of g as a combination of "known" group elements $L = (L_1, \ldots, L_t)$

that were previously given to $\mathcal{A}$, i.e., numbers $c_1, \ldots, c_t$ such that $g = \prod_{i=1}^{t} L_i^{c_i}$. Any attacker operating in the GGM can also be implemented in the AGM [201], so the known extractability of the KZG polynomial commitment in the AGM [98] is a strictly stronger result than security in the GGM.[9]

15.3 Extension of KZG to Multilinear Polynomials

The previous section gave a polynomial commitment scheme based on pairings, for univariate polynomials over the field $\mathbb{F}_p$ where p is the order of the groups involved in the pairing. In this section, we wish to give a similar commitment scheme for *multilinear* polynomials q over $\mathbb{F}_p$, proposed by Papamanthou *et al.* [202]. Let ℓ denote the number of variables of q, so $q\colon \mathbb{F}_p^{\ell} \to \mathbb{F}_p$. In applications of multilinear polynomial commitment schemes (namely, to transforming IPs and MIPs to succinct arguments for circuit satisfiability), it is convenient to work with polynomials specified over the Lagrange basis (see Lemma 3.8 for a definition of the Lagrange basis polynomials), so we present the commitment scheme in this setting, though the scheme works just as well over any basis of multilinear polynomials.

The structured reference string (SRS) now consists of encodings in $\mathbb{G}$ of all powers of all Lagrange basis polynomials evaluated at a randomly chosen input $r \in \mathbb{F}^{\ell}$. That is, if $\chi_1, \ldots, \chi_{2^{\ell}}$ denotes an enumeration of the 2^{ℓ} Lagrange basis polynomials, the SRS equals $(g^{\chi_1(r)}, \ldots, g^{\chi_{2^{\ell}}(r)})$. Once again, the toxic waste that must be discarded because it can be used to destroy binding is the value r.

As in the univariate commitment scheme, to commit to a multilinear polynomial q over $\mathbb{F}_p$, the committer sends a value c claimed to equal $g^{q(r)}$. Note that while the committer does not know r, it is still able to compute $g^{q(r)}$ using the SRS: if $q(X) = \sum_{i=0}^{2^{\ell}} c_i \chi_i(X)$, then $g^{q(r)} = \prod_{i=0}^{2^{\ell}} (g^{\chi_i(r)})^{c_i}$, which can be computed given the values $g^{\chi_i(r)}$ for all $i = 0, \ldots, 2^{\ell}$ even without knowing r.

[9] To be precise, there are a couple of variants of the GGM considered in the literature [188], [234]. See [167] for a discussion of subtleties regarding the relationship between these different versions of the GGM and the AGM.

To open the commitment at input $z \in \mathbb{F}_p^\ell$ to some value v, i.e., to prove that $q(z) = v$, the committer computes a series of ℓ "witness polynomials" $w_1, \ldots, w_\ell$, defined in the following fact.

Fact 15.1 (Papamanthou *et al.* [202]). For any fixed $z = (z_1, \ldots, z_\ell) \in \mathbb{F}_p^\ell$ and any multilinear polynomial q, $q(z) = v$ if and only if there is a unique set of ℓ multilinear polynomials $w_1, \ldots, w_\ell$ such that

$$q(X) - v = \sum_{i=1}^{\ell} (X_i - z_i) w_i(X). \tag{15.4}$$

Proof. If $q(X) - v$ can be expressed as the right hand side of Equation (15.4), then clearly $q(z) - v = 0$, and hence $q(z) = v$.

On the other hand, suppose that $q(z) = v$. Then by dividing the polynomial $q(X) - v$ by the polynomial $(X_1 - z_1)$, we can identify multilinear polynomials w_1 and s_1 such that

$$q(X) - v = (X_1 - z_1) \cdot w_1(X_1, X_2, \ldots, X_\ell) + s_1(X_2, X_3, \ldots, X_\ell),$$

where $s_1(X_2, X_3, \ldots, X_\ell)$ is the remainder term, and does not depend on variable X_1. Iterating this process, we can divide s_1 by the polynomial $(X_2 - Z_2)$ to write

$$\begin{aligned} q(X) - v = (X_1 - z_1) \cdot w_1(X_1, X_2, \ldots, X_\ell) + (X_2 - z_2) \cdot w_2(X_2, \ldots, X_\ell) \\ + s_2(X_3, X_4, \ldots, X_\ell) \end{aligned}$$

and so forth until we have written

$$q(X) - v = \sum_{i=1}^{\ell} (X_i - z_i) \cdot w_i(X_1, X_2, \ldots, X_\ell) + s_\ell,$$

where s_ℓ depends on no variables, i.e., s_ℓ is simply an element in $\mathbb{F}_p$. Since $q(z) - v = 0$, it must hold that $s_\ell = 0$, completing the proof. □

To open the commitment at input $z \in \mathbb{F}_p^\ell$ to value v, the prover computes $w_1, \ldots, w_\ell$ as per Fact 15.1 and sends to the verifier values $y_1, \ldots, y_\ell$ claimed to equal $g^{w_i(r)}$ for $i = 1, \ldots, \ell$. Again, since each w_i is multilinear, $g^{w_i(r)}$ can be computed from the SRS despite the fact that the prover does not know r. The verifier checks that

$$e(c \cdot g^{-v}, g) = \prod_{i=1}^{\ell} e(y_i, g^{r_i} \cdot g^{-z_i}).$$

Note that the verifier is able to perform this check so long as the verification key includes g^{r_i} for each i (the verification key is a subset of the SRS, as each dictator function $(X_1, \ldots, X_\ell) \mapsto X_i$ is a Lagrange basis polynomial).

Correctness is clear: if $c = g^{q(r)}$ and $y_i = g^{w(r_i)}$ for $i = 1, \ldots, \ell$, then

$$\begin{aligned} e(c \cdot g^{-v}, g) &= e(g^{q(r)-v}, g) = e(g^{\sum_{i=1}^{\ell} w_i(r)\cdot(r_i - z_i)}, g) \\ &= \prod_{i=1}^{\ell} e(g^{w_i(r)}, g^{r_i - z_i}) = e(y_i, g^{r_i} \cdot g^{-z_i}). \end{aligned}$$

Here, the first inequality holds because $c = g^{q(r)}$, the second holds by Equation (15.4), the third holds by bilinearity of e, and the fourth holds because $y_i = g^{w_i(r)}$.

The proof of binding and techniques to achieve extractability are similar to the previous section and we omit them for brevity.

Costs. Like the univariate pairing-based polynomial commitment scheme of the previous section (Section 15.2), the ℓ-variate multilinear polynomial commitment consists of a constant number of group elements. However, whereas evaluation proofs for the univariate protocol also consisted of a constant number of group elements, evaluation proofs for the multilinear polynomial protocol are ℓ group elements rather than $O(1)$, with a corresponding increase in verification time from $O(1)$ group operations and bilinear map evaluations, to $O(\ell)$.

In terms of committer runtime, Zhang *et al.* [259, Appendix G] show that the committer in the protocol for multilinear polynomials can compute the polynomials $w_1, \ldots, w_\ell$ with just $O(2^\ell)$ field operations in total. And once these polynomials are computed, the prover can compute all ℓ necessary values $g^{w_i(r)}$ with $O(2^\ell)$ many group exponentiations in total.

15.4 Dory: Transparent Scheme with Logarithmic Verification Costs

This section describes a polynomial commitment scheme called Dory with similar (logarithmic) verification costs to the polynomial commitment scheme of the previous section, but that does not rely on a

trusted setup. It does require a pre-processing phase that requires time square root in the size of the polynomial to be committed, but this pre-processing phase does not produce any "toxic waste" that must be discarded to guarantee no one can break the scheme's binding property. Asymptotically, its verifier costs compare favorably to Bulletproofs (Section 14.4): like Bulletproofs, it is transparent with logarithmic-sized proofs, but unlike Bulletproofs it has logarithmic rather than linear verifier time. However, concretely its proofs are larger than Bulletproofs by a significant constant factor.

Dory builds on beautiful ideas and building blocks developed over a variety of works [2], [75], [84], especially so-called AFGHO commitments [2]. While Dory itself is arguably somewhat complicated, the building blocks that comprise it are simpler and useful in their own right.

15.4.1 Commitments to Vectors of Group Elements via Inner Pairing Products

Let $\mathbb{F}_p$ be the field of prime order p and $\mathbb{G}$ be a multiplicative cyclic group of order p. Let $\mathbf{h} = (h_1, \ldots, h_n)$ be a public vector of (randomly chosen) generators of $\mathbb{G}$. Recall that an (unblinded) Pedersen vector commitment is a compressing commitment to a vector of field elements $v \in \mathbb{F}_p^n$, given by $\mathsf{Com}(v) = \prod_{i=1}^n h_i^{v_i}$ (see Section 14.2). In other words, this commitment takes (unblinded) Pedersen commitments to each entry of v, and multiplies them together to get a single commitment to the whole vector. The commitment is a single element of the group $\mathbb{G}$.

Now let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ be a pairing-friendly triple of groups of order p. In the previous paragraph, we expressed $\mathbb{G}$ as a multiplicative group, but for the remainder of this section, we will express $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ as *additive* groups. As in Section 14.4 where we described Bulletproofs, this allows us to express commitments as an inner product between the vector to be committed and the commitment key.

Using pairings, one can define an analog of Pedersen vector commitments that allows one to commit to vectors of *group elements*, i.e., vectors in $\mathbb{G}_1^n$, rather than vectors of field elements. Specifically, for $w \in \mathbb{G}_1^n$, and for a fixed vector $\mathbf{g} = (g_1, \ldots, g_n) \in \mathbb{G}_2^n$ of public, randomly

chosen group elements, define

$$\text{IPPCom}(w) = \sum_{i=1}^{n} e(w_i, g_i). \tag{15.5}$$

Note that $\text{IPPCom}(w)$ is a single element of the target group $\mathbb{G}_t$. We use the notation IPPCom as short-hand for the term *inner-pairing-product commitment*. This refers to the fact that $\text{IPPCom}(w)$ can be thought of as the inner product $\langle w, \mathbf{g} \rangle = \sum_{i=1}^{n} w_i \cdot g_i$, where the "multiplication" of w_i and g_i is defined via the pairing $e(w_i, g_i)$. From now on, for $w \in \mathbb{G}_1^n, \mathbf{g} \in \mathbb{G}_2^n$, we use $\langle w, \mathbf{g} \rangle$ to denote the inner-pairing-product $\sum_{i=1}^{n} e(w_i, g_i)$.

Intuitively, $e(w_i, g_i)$ acts as a Pedersen commitment to w_i despite the fact that w_i is a group element in $\mathbb{G}_1$ rather than a field element in $\mathbb{F}_p$. The sum $\sum_{i=1}^{n} e(w_i, g_i)$ of all such entry-wise Pedersen commitments is the natural compressing commitment to the vector w.

The above commitment scheme for vectors of group elements originated in work of Abe *et al.* [2], [143] and are often called AFGHO-commitments. We use the terms AFGHO-commitments and inner-pairing-product commitments interchangeably.

Rendering the Commitment Perfectly Hiding. Recall that a Pedersen vector commitment in group $\mathbb{G}_1$ can be made perfectly hiding by having an extra randomly chosen public parameter $g \in \mathbb{G}_1$, and having the committer pick a random $r \in \mathbb{F}_p$ and include a blinding factor g^r in the commitment. Analogously, $\text{IPPCom}(w)$ can be made perfectly hiding by having the committer pick a random $r \in \mathbb{G}_1$ and including a blinding term $e(r, g)$ in the commitment, i.e., defining

$$\text{IPPCom}(w) = e(r, g) + \sum_{i=1}^{n} e(w_i, g_i).$$

For simplicity, we omit this blinding factor from the remainder of our treatment.

Computational Binding. Recall from Section 15.1 that the Decisional Diffie-Hellman (DDH) assumption for an additive group $\mathbb{G}_1$ states that

no efficient algorithm can meaningfully distinguish between a random tuple of the form

$$(g, a \cdot g, b \cdot g, c \cdot g)$$

with a, b, c chosen at random from $\mathbb{F}_p$ versus one of the form

$$(g, a \cdot g, b \cdot g, (ab) \cdot g).$$

We show that assuming DDH holds in $\mathbb{G}_1$, IPPCom(w) is a computationally binding commitment to $w \in \mathbb{G}_1^n$. For expository clarity, our presentation assumes that $n = 2$.

Given a DDH challenge $(g, a \cdot g, b \cdot g, c \cdot g)$ in $\mathbb{G}_1$, we must explain how to use an efficient prover $\mathcal{P}$ that breaks binding of the commitment scheme to give an efficient algorithm $\mathcal{A}$ that breaks the DDH assumption. $\mathcal{A}$ sets $\mathbf{g} = (g, a \cdot g) \in \mathbb{G}_1 \times \mathbb{G}_1$; note that by definition of the DDH challenge distributions, both entries of $\mathbf{g}$ are uniform random group elements. $\mathcal{A}$ then runs $\mathcal{P}$ to identify a nonzero commitment $c^* \in \mathbb{G}_t$ and two openings $u = (u_1, u_2), w = (w_1, w_2)$ of c^*, meaning that

$$c^* = e(u_1, g) + e(u_2, a \cdot g) = e(w_1, g) + e(w_2, a \cdot g).$$

Let $v = u - w$, and write $v = (v_1, v_2) \in \mathbb{G}_1 \times \mathbb{G}_1$. Since $c^* \neq 0$, it follows that v is not the zero-vector. Moreover, by bilinearity of e,

$$e(v_1, g) + e(v_2, a \cdot g) = 0. \tag{15.6}$$

The DDH adversary $\mathcal{A}$ then outputs 1 if and only if

$$e(v_1, b \cdot g) + e(v_2, c \cdot g) = 0. \tag{15.7}$$

That $\mathcal{A}$ breaks the DDH assumption in $\mathbb{G}_1$ can be seen as follows. First, if $c = a \cdot b$, then the left hand side of Equation (15.7) equals:

$$\begin{aligned} e(v_1, b \cdot g) + e(v_2, (a \cdot b) \cdot g) &= e(v_1, b \cdot g) + e(v_2, b \cdot (a \cdot g)) \\ &= b \cdot \left(e(v_1, b \cdot g) + e(v_2, c \cdot g)\right). \end{aligned}$$

This last expression equals 0 by Equation (15.6). Hence, in this case $\mathcal{A}$ outputs 1. Meanwhile, if c is chosen at random from $\mathbb{F}_p$, then since v is not the zero-vector, Equation (15.7) holds with probability just $1/p$.

Hence, $\mathcal{A}$ succeeds in distinguishing tuples of the form $(g, a \cdot g, b \cdot g, c \cdot g)$, with a, b, c chosen at random from $\mathbb{F}_p$, from those of the form $(g, a \cdot g, b \cdot g, (ab) \cdot g)$.

15.4.2 Committing to Field Elements Using Pairings

One can also use inner-pairing-product commitments in place of Pedersen-vector commitments to commit to vectors of field elements. Let h be any element of $\mathbb{G}_1$ and $\mathbf{g} = (g_1, \ldots, g_n) \in \mathbb{G}_2^n$ be a random vector of $\mathbb{G}_2$ elements. For a vector $v \in \mathbb{F}_p^n$, consider the vector $w(v)$ of entry-wise (unblinded) Pedersen commitments to v in $\mathbb{G}_1$ with commitment key h, i.e., $w(v)_i = v_i \cdot h$, and define IPPCom(v) as

$$\text{IPPCom}(v) = \text{IPPCom}(w(v)) = \langle w(v), \mathbf{g} \rangle = \sum_{i=1}^{n} e(v_i \cdot h, g_i). \quad (15.8)$$

Since IPPCom$(w(v))$ is a binding commitment to $w(v)$, and the map $v \mapsto w(v)$ is bijective, IPPCom(v) is a binding commitment to $v \in \mathbb{F}_p^n$. It will be important both for concrete efficiency of computing commitments, and for the polynomial commitment scheme that we develop (Sections 15.4.4 and 15.4.5), that the *same* group element h is used to compute every entry of $w(v)$.

Efficiency Comparison. There are two natural ways to compute IPPCom(v). One is by computing the right hand side of Expression (15.8) directly, which requires n evaluations of the bilinear map e and n group operations in the target group $\mathbb{G}_t$. The other (faster) way is by computing a Pedersen-vector commitment $c = \sum_{i=1}^{n} v_i g_i$ in $\mathbb{G}_2$, and then applying the bilinear map just once to compute $e(h, c)$. By bilinearity of e,

$$e(h, c) = e(h, \sum_{i=1}^{n} v_i \cdot g_i) = \sum_{i=1}^{n} e(h, v_i \cdot g_i) = \sum_{i=1}^{n} e(v_i \cdot h, g_i) = \text{IPPCom}(v).$$

Both methods of computing IPPCom(v) are concretely slower than computing a Pedersen vector commitment to v in $\mathbb{G}_1$. This is because group operations in $\mathbb{G}_2$ and $\mathbb{G}_t$ are typically slower than group operations in $\mathbb{G}_1$, and computing an evaluation of the bilinear map e is slower still. For example, according to microbenchmarks in [179], if using the popular pairing-friendly curve BLS12-381, a $\mathbb{G}_t$ operation is about 4 times slower than one in $\mathbb{G}_1$ while a $\mathbb{G}_2$ operation is about 2 times slower than one in $\mathbb{G}_1$. Moreover, operations in a pairing-friendly group

$\mathbb{G}_1$ such as BLS12-381 are perhaps 2-3 times slower than operations in the fastest non-pairing-friendly groups.

On top of this slow commitment computation, the commitment size IPPCom(v) is concretely larger—for BLS12-381, representations of elements of the target group $\mathbb{G}_t$ are four times bigger than those of elements of $\mathbb{G}_1$. In summary, while the asymptotic costs of computing and sending IPPCom(w) are similar to Pedersen vector commitments, the concrete costs are worse.

A Simplification in the Remainder of the Presentation. As discussed in Section 15.1, the DDH assumption *cannot* hold in $\mathbb{G}_1$ if $\mathbb{G}_1 = \mathbb{G}_2$, i.e., if the pairing is a symmetric pairing. This is because, given tuple (g, ag, bg, cg), one can check whether $c = a \cdot b$ by checking whether $e(g, c \cdot g) = e(a \cdot g, b \cdot g)$.

However, the DDH assumption can hold in $\mathbb{G}_1$ and $\mathbb{G}_2$ if the two groups are not equal, i.e., if there is no efficiently computable mapping ϕ group $\mathbb{G}_1$ to $\mathbb{G}_2$ or vice versa that preserves group structure in the sense that $\phi(a+b) = \phi(a)+\phi(b)$. This is (believed to be) the case for pairings used in practice, such as BLS12-381. The assumption that DDH holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$ is called the *symmetric external Diffie-Hellman assumption*, abbreviated SXDH.

Despite the fact that IPPCom(w) is *not* necessarily a binding commitment to w if $\mathbb{G}_1 = \mathbb{G}_2$, we will nonetheless assume for the remainder of our presentation of Dory that $\mathbb{G}_1 = \mathbb{G}_2$, denoting both groups by $\mathbb{G}$.[10] This simplifies the presentation of the Dory protocol. The changes required to the protocol in the case where $\mathbb{G}_1 \neq \mathbb{G}_2$ are straightforward, but they introduce a notational burden that we prefer to avoid.

Outline for the Remainder of the Presentation. In order to highlight in a modular fashion the main ideas that go into the Dory polynomial commitment scheme, we describe progressively more sophisticated protocols. First, in Section 15.4.3, we explain how a prover $\mathcal{P}$ can convince

[10] See [143] for a modification of the commitment scheme that is binding under a plausible assumption for symmetric pairings.

a verifier that $\mathcal{P}$ knows how to open some inner-pairing-product commitment $c_u = \text{IPPCom}(u)$ to some vector $u \in \mathbb{G}^n$. The proof size and verification cost of this protocol are $O(\log^2 n)$, after a transparent linear-time pre-processing phase. Second, in Section 15.4.4 we explain how to extend this protocol to give a transparent polynomial commitment scheme in which the verifier's runtime is $O(\log^2 n)$. Third, in Section 15.4.5, we explain a modification of the protocol of Section 15.4.3 that reduces the runtime of the pre-processing phase from linear in n to $O(n^{1/2})$. Finally, in Section 15.4.6, we present a more complicated variant of the protocol of Section 15.4.3 that reduces verification costs from $O(\log^2 n)$ to $O(\log n)$.

15.4.3 Proving Knowledge of Opening with $O(\log^2 n)$ Verification Cost

Let $u \in \mathbb{G}^n$, and assume for simplicity that n is a power of 2. Given public input $c_u = \text{IPPCom}(u)$, the following protocol is transparent and allows the prover to prove knowledge of an opening u of c_u. The verifier runs in $O(\log^2 n)$ time, after a transparent pre-processing phase that is independent of u.

Recap of Bulletproofs. Let us briefly recall the Bulletproofs (Section 14.4) protocol for establishing knowledge of an opening $u^{(0)} \in \mathbb{F}_p^n$ of a *Pedersen* vector commitment

$$c_{u^{(0)}} = \langle u^{(0)}, \mathbf{g}^{(0)} \rangle = \sum_{i=1}^{n} u_i \cdot g_i.$$

Conceptually, $\mathbf{g}^{(0)}$ is broken into two halves $\mathbf{g}_L^{(0)}$ and $\mathbf{g}_R^{(0)}$, and likewise $u^{(0)}$ is written as $(u_L^{(0)}, u_R^{(0)})$. The prover sends two commitments v_L and v_R claimed to equal $\langle u_L^{(0)}, \mathbf{g}_R^{(0)} \rangle$ and $\langle u_R^{(0)}, \mathbf{g}_L^{(0)} \rangle$. The verifier picks a random $\alpha_1 \in \mathbb{F}_p$ and sends it to $\mathcal{P}$, and uses v_L and v_R to homomorphically update the commitment $c_{u^{(0)}}$ to

$$c_{u^{(1)}} \coloneqq c_{u^{(0)}} + \alpha_1^2 v_L + \alpha_1^{-2} v_R.$$

If the prover is honest, $c_{u^{(1)}}$ is a commitment to the to length-$(n/2)$ vector

$$u^{(1)} = \alpha_1 u_L^{(0)} + \alpha_1^{-1} u_R^{(0)}.$$

using the commitment key

$$\mathbf{g}^{(1)} := \alpha_1^{-1}\mathbf{g}_L^{(0)} + \alpha_1\mathbf{g}_R^{(0)}. \tag{15.9}$$

$\mathcal{P}$ and $\mathcal{V}$ then proceed to the next round, in which $\mathcal{P}$ recursively establishes knowledge of an opening $u^{(1)}$ to $c_{u^{(1)}}$. This continues for $\log n$ rounds, at which point the recursion bottoms out: $u^{(\log n)}$ and $\mathbf{g}^{(\log n)}$ have length 1, and hence (if zero-knowledge is not a consideration) $\mathcal{P}$ can afford to prove knowledge of $u^{(\log n)}$ by sending it explicitly to $\mathcal{V}$, who can check that $u^{(\log n)} \cdot \mathbf{g}^{(\log n)} = c_{u^{(\log n)}}$.

Why is the verifier runtime in the above protocol linear rather than logarithmic? The answer is: in each round, the verifier needs to update the commitment key from $\mathbf{g}^{(i-1)}$ to $\mathbf{g}^{(i)} = \alpha_i^{-1}\mathbf{g}_L^{(i-1)} + \alpha\mathbf{g}_R^{(i-1)}$. This takes time at least $O(n/2^i)$.

The Pre-Processing Procedure. Now let $u^{(0)}$ be a vector in $\mathbb{G}^n$, and let $c_{u^{(0)}} = \mathrm{IPPCom}(u^{(0)})$. To avoid linear verifier time in our protocol for establishing knowledge of an opening $u^{(0)} \in \mathbb{G}^n$ for $c_{u^{(0)}}$, we rely on a pre-processing phase that is independent of $u^{(0)}$, depending only on the public commitment key $\mathbf{g}$. For exposition, we describe the preprocessing as occurring over $\log n$ iterations, with two inner-pairing-product commitments (i.e., elements of $\mathbb{G}_t$) produced per iteration. We refer to the party doing the pre-processing as the verifier—in fact, any entity willing to invest the effort can perform the pre-processing and distribute the resulting (logarithmically-many) commitments to the world. Any entity willing to invest the effort can also validate the distributed commitments, raising an alarm if a discrepancy is found.

- (First iteration of pre-processing): Let $\mathbf{g}^{(0)} = (\mathbf{g}_L^{(0)}, \mathbf{g}_R^{(0)}) \in \mathbb{G}^{n/2} \times \mathbb{G}^{n/2}$ be the commitment key used to compute the initial commitment $c_{u^{(0)}} = \langle u^{(0)}, \mathbf{g}^{(0)} \rangle$. The first pre-processing iteration outputs inner-pairing-product commitments $\Delta_L^{(1)}$ and $\Delta_R^{(1)}$ to $\mathbf{g}_L^{(0)}$ and to $\mathbf{g}_R^{(0)}$ respectively, using public, randomly chosen commitment key $\Gamma^{(1)} \in \mathbb{G}^{n/2}$.[11] That is, $\Delta_L^{(1)} = \langle \mathbf{g}_L^{(0)}, \Gamma^{(1)} \rangle$ and $\Delta_R^{(1)} = \langle \mathbf{g}_R^{(0)}, \Gamma^{(1)} \rangle$.

[11] $\Gamma^{(1)}$ need not be independent of $\mathbf{g}^{(0)}$, e.g., it is fine for $\Gamma^{(0)}$ to equal $\mathbf{g}_L^{(0)}$.

- (Second iteration): Write $\Gamma^{(1)}$ itself as $(\Gamma_L^{(1)}, \Gamma_R^{(1)}) \in \mathbb{G}^{n/4} \times \mathbb{G}^{n/4}$. The second iteration computes commitments $\Delta_L^{(2)}$ and $\Delta_R^{(2)}$ to $\Gamma_L^{(1)}$ and to $\Gamma_R^{(1)}$ respectively, using public, randomly chosen commitment key $\Gamma^{(2)} \in \mathbb{G}^{n/4}$.[12]

- In general, in iteration $i > 1$: compute commitments $\Delta_L^{(i)}$ and $\Delta_R^{(i)}$ to the left and right halves of the commitment key $\Gamma^{(i-1)}$ that was used to compute the previous iteration's commitments $\Delta_L^{(i-1)}$ and $\Delta_R^{(i-1)}$.

The pre-processing ends after iteration $i = \log(n)$. At that point, $\Gamma^{(i)}$ has length 1, so the pre-processing just outputs $\Gamma^{(i)}$ explicitly. Note that after the pre-processing is done, a logarithmic-time verifier does have time to read and store the two commitments $\Delta_L^{(i)}$ and $\Delta_R^{(i)}$ output by each iteration of pre-processing, but does *not* have the time to read or store the corresponding *commitment key* $\Gamma^{(i)}$ used to produce $\Delta_L^{(i)}$ and $\Delta_R^{(i)}$. This is because $\Gamma^{(i)}$ has size $n/2^i$ and hence is super-logarithmic for all $i \leq \log(n) - \log\log(n)$. As we will see, this means the verifier in the knowledge-of-opening protocol described below will have to somehow "check" that the prover knows how to open many different commitments $\Delta_L^{(i)}$ and $\Delta_R^{(i)}$ *without the verifier even knowing the keys* used to produce those commitments.

The Knowledge of Opening Protocol. Let $u^{(0)}$ be a vector in $\mathbb{G}^n$. The verifier begins the protocol knowing a commitment $c_{u^{(0)}}$. If the prover is honest,

$$c_{u^{(0)}} = \langle u^{(0)}, \mathbf{g}^{(0)} \rangle, \tag{15.10}$$

and the prover needs to prove that it knows a vector $u^{(0)}$ satisfying Equation (15.10). Recall that a core difficulty here is that the verifier doesn't know $\mathbf{g}^{(0)}$ as in Bulletproofs, but rather only some "pre-processed" information about $\mathbf{g}^{(0)}$, namely commitments to $\mathbf{g}_L^{(0)}$ and $\mathbf{g}_R^{(0)}$ under some different commitment key $\Gamma^{(1)}$.

The key idea is that, in each round i, rather than explicitly computing $\mathbf{g}^{(i)}$ from $\mathbf{g}^{(i-1)}$ as per Equation (15.9), $\mathcal{V}$ instead uses homomorphism

[12] As in Footnote 11, $\Gamma^{(2)}$ need not be independent of $\Gamma^{(1)}$, e.g., it is fine for $\Gamma^{(2)}$ to equal $\Gamma_L^{(1)}$.

of the commitments output by the pre-processing procedure to compute in constant time a *commitment* to $\mathbf{g}^{(i)}$ under a suitable commitment key. Roughly speaking, the verifier can use this commitment to force the prover to do the hard work of computing $\mathbf{g}^{(i)}$ explicitly. The prover will only ever explicitly reveal to the verifier the final "fully collapsed" commitment key $\mathbf{g}^{(\log n)}$, which consists of a single group element. Protocol details follow.

Round 1 **procedure**: As in Bulletproofs, the prover begins by sending two commitments v_L and v_R claimed to equal $\langle u_L^{(0)}, \mathbf{g}_R^{(0)} \rangle$ and $\langle u_R^{(0)}, \mathbf{g}_L^{(0)} \rangle$. The verifier picks a random $\alpha_1 \in \mathbb{F}_p$ and sends it to $\mathcal{P}$. The verifier uses v_L and v_R to homomorphically update the commitment $c_{u^{(0)}}$ to

$$c_{u^{(1)}} \coloneqq c_{u^{(0)}} + \alpha_1^2 v_L + \alpha_1^{-2} v_R.$$

Recall that unlike in Bulletproofs, our $\mathcal{V}$ does not know $\mathbf{g}^{(0)}$, but does know pre-processing commitments $\Delta_L^{(1)} = \langle g_L^{(0)}, \Gamma^{(1)} \rangle$ and $\Delta_R^{(1)} = \langle g_R^{(0)}, \Gamma^{(1)} \rangle$. Via homomorphism, $\mathcal{V}$ can compute a commitment

$$c_{\mathbf{g}^{(1)}} = \alpha_1^{-1} \Delta_L^{(1)} + \alpha_1 \Delta_R^{(1)}$$

to $\mathbf{g}^{(1)} = \alpha_1^{-1} \mathbf{g}_L^{(0)} + \alpha_1 \mathbf{g}_R^{(0)}$ under commitment key $\Gamma^{(1)}$.

The above Round 1 leaves the prover and verifier in the following situation. Unlike in Bulletproofs, at the start of Round 2, our verifier does not know $\mathbf{g}^{(1)}$. All that our verifier knows is a *commitment* $c_{\mathbf{g}^{(1)}}$ to $\mathbf{g}^{(1)}$ under commitment key $\Gamma^{(1)} = (\Gamma_L^{(1)}, \Gamma_R^{(1)}) \in \mathbb{G}^{n/4} \times \mathbb{G}^{n/4}$.

Because the information known to $\mathcal{V}$ is so limited, at the start of Round 2, $\mathcal{P}$ needs to prove that it knows vectors $u^{(1)}$ and $\mathbf{g}^{(1)}$ in $\mathbb{G}^{n/2}$ such that

$$c_{u^{(1)}} = \langle u^{(1)}, \mathbf{g}^{(1)} \rangle \tag{15.11}$$

and

$$c_{\mathbf{g}^{(1)}} = \langle \mathbf{g}^{(1)}, \Gamma^{(1)} \rangle. \tag{15.12}$$

In summary, the first round of our protocol started with a claim about *one* inner product equation involving two vectors $u^{(0)}$ and $\mathbf{g}^{(0)}$ of length n (Equation (15.10)), in which $\mathcal{V}$ only knew "pre-computed commitments" $\Delta_L^{(1)}, \Delta_R^{(1)}$ to $\mathbf{g}^{(0)}$. And it reduced it to a claim about *two*

inner product equations involving *three* vectors of length $n/2$, namely $u^{(1)}$, $\mathbf{g}^{(1)}$, and $\Gamma^{(1)}$, in which $\mathcal{V}$ only knows pre-computed commitments $\Delta_L^{(2)}, \Delta_R^{(2)}$ to $\Gamma_L^{(1)}$ and $\Gamma_R^{(1)}$.

Round 2 Procedure.

- *A trivial case:* If $n = 2$, so the pre-processing phase output $\Gamma^{(1)}$ explicitly, then a trivial way for the prover to establish both of these claims is to explicitly reveal $u^{(1)}$ and $\mathbf{g}^{(1)}$ to $\mathcal{V}$, who can then check Equations (15.11) and (15.12) explicitly. But this does not work if $n \geq 4$.

- *What to do if* $n \geq 4$. Fortunately, Equations (15.11) and (15.12) are both of a form that Bulletproofs was designed to handle. Namely, $\mathcal{P}$ is claiming to know some vector satisfying an inner-product relation with some other vector ($u^{(1)}$ with $\mathbf{g}^{(1)}$ in Equation (15.11) and $\mathbf{g}^{(1)}$ with $\Gamma^{(1)}$ in Equation (15.12)). So $\mathcal{P}$ and $\mathcal{V}$ can apply the Bulletproofs scheme in parallel to both claims, using the same random verifier-chosen $\alpha_2 \in \mathbb{F}_p$ for both, to reduce each claim to an equivalent one involving vectors of half the length.

  That is, $\mathcal{P}$ sends commitments $v_L, v_R, w_L, w_R \in \mathbb{G}_t$ claimed to equal $\langle u_L^{(1)}, \mathbf{g}_R^{(1)} \rangle$, $\langle u_R^{(1)}, \mathbf{g}_L^{(1)} \rangle$, $\langle \mathbf{g}_L^{(1)}, \Gamma_R^{(2)} \rangle$ and $\langle \mathbf{g}_R^{(1)}, \Gamma_L^{(2)} \rangle$. $\mathcal{V}$ then sends $\alpha_2 \in \mathbb{F}$ to $\mathcal{P}$. $\mathcal{V}$ sets

$$c_{u^{(2)}} \coloneqq c_{u^{(1)}} + \alpha_2^2 v_L + \alpha_2^{-2} v_R.$$

  If $\mathcal{P}$ is honest, then

$$c_{u^{(2)}} = \langle u^{(2)}, \mathbf{g}^{(2)} \rangle,$$

  where

$$u^{(2)} = \alpha_2 u_L^{(1)} + \alpha_2^{-1} u_R^{(1)},$$

  and

$$\mathbf{g}^{(2)} \coloneqq \alpha_2^{-1} \mathbf{g}_L^{(1)} + \alpha_2 \mathbf{g}_R^{(1)}.$$

  Likewise, $\mathcal{V}$ sets

$$c_{\mathbf{g}^{(2)}} \coloneqq c_{\mathbf{g}^{(1)}} + \alpha_2^{-2} w_L + \alpha_2^2 w_R.$$

If $\mathcal{P}$ is honest, then $c_{\mathbf{g}^{(2)}}$ is a commitment to $\mathbf{g}^{(2)}$ under commitment key $\Gamma' = \alpha_2 \Gamma_L^{(1)} + \alpha_2^{-1} \Gamma_R^{(1)}$, i.e.,

$$c_{\mathbf{g}^{(2)}} = \langle \alpha_2^{-1} \mathbf{g}_L^{(1)} + \alpha_2 \mathbf{g}_R^{(1)}, \alpha_2 \Gamma_L^{(1)} + \alpha_2^{-1} \Gamma_R^{(1)} \rangle.$$

Finally, $\mathcal{V}$ computes a commitment $c_{\Gamma'}$ to Γ' under commitment key $\Gamma^{(2)}$ homomorphically given the pre-processing commitments $\Delta_L^{(2)}$ and $\Delta_R^{(2)}$, via $c_{\Gamma'} = \alpha_2 \Delta_L^{(2)} + \alpha_2^{-1} \Delta_R^{(2)}$.

The above Round 2 procedure reduces the task of proving knowledge of $u^{(1)}, \mathbf{g}^{(1)} \in \mathbb{G}^{n/2}$ satisfying Equations (15.11) and (15.12) to proving knowledge of $u^{(2)}, \mathbf{g}^{(2)}, \Gamma' \in \mathbb{G}^{n/4}$ satisfying:

$$c_{u^{(2)}} = \langle u^{(2)}, \mathbf{g}^{(2)} \rangle, \tag{15.13}$$

$$c_{\mathbf{g}^{(2)}} = \langle \mathbf{g}^{(2)}, \Gamma' \rangle, \tag{15.14}$$

$$c_{\Gamma'} = \langle \Gamma', \Gamma^{(2)} \rangle. \tag{15.15}$$

That is, as discussed in the "sketch of the knowledge extractor" several paragraphs hence, $\mathcal{P}$'s knowledge of $u^{(2)}, g^{(2)}, \Gamma' \in \mathbb{G}^{n/4}$ satisfying Equations (15.13)–(15.15) (for at least 3 values of $\alpha_2 \in \mathbb{F}_p$) implies knowledge of $u^{(1)}$ and $\mathbf{g}^{(1)}$ satisfying Equations (15.11) and (15.12).

In summary, Round 2 started with a claim about *two* inner product equations involving *three* vectors of length $n/2$, namely $u^{(1)}, \mathbf{g}^{(1)}, \Gamma^{(1)}$, in which $\mathcal{V}$ only knows pre-computed commitments $\Delta_L^{(2)}, \Delta_R^{(2)}$ to $\Gamma^{(1)}$. And it reduced it to a claim about *three* inner product equations involving *four* vectors of length $n/4$, in which $\mathcal{V}$ only knows pre-computed commitments to the fourth vector $\Gamma^{(2)}$.

Round $i > 2$. The above Round-2 procedure can be iterated, to ensure that at the start of Round i, the prover has to establish knowledge of $i+1$ vectors, each of length $n/2^i$, satisfying i inner product equations. In more detail, denote the $i+1$ vectors $\mathcal{P}$ claims to know at the start of round i by $v_1, \ldots, v_{i+1}$. Then $v_1 = u^{(i-1)}$, $v_2 = \mathbf{g}^{(i-1)}$, and $v_{i+1} = \Gamma^{(i-1)}$. And the j'th equation at the start of Round i is of the form $\langle v_j, v_{j+1} \rangle = c_j$ for some commitment c_j.

The prover operates analogously to Round 2, sending commitments to two cross-terms per equation, with the verifier then picking a random

$\alpha_i \in \mathbb{F}_p^n$ and sending it to $\mathcal{P}$. For each odd $j \in \{1, \ldots i+1\}$, the verifier uses homomorphism to derive a commitment to $\alpha_i \cdot v_{j,L} + \alpha_i^{-1} \cdot v_{j,R}$. Likewise, for each even j, $\mathcal{V}$ derives a commitment to $\alpha_i^{-1} \cdot v_{j,L} + \alpha_i \cdot v_{j,R}$. For $v_{i+1} = \Gamma^{(i-1)}$, the appropriate commitment is derived homomorphically from the pre-processing commitments $\Delta_L^{(i)}$ and $\Delta_R^{(i)}$. Round $i+1$ is then devoted to proving that the prover indeed knows how to open each of the $i+1$ derived commitments, which entails $i+1$ inner-product equations involving $i+2$ vectors.

The iterations stop after round $\log(n)$; at that point, the vectors involved in each of the $\log(n)+1$ equations have length just 1. If zero-knowledge is not a consideration, the prover can establish that it knows such vectors by simply sending them explicitly to $\mathcal{V}$, and $\mathcal{V}$ can directly check that the received values indeed satisfy all the equations claimed.

Verification Costs. The total number of commitments sent by the prover in Round i is $O(i)$, leading to a communication cost $O(\sum_{i=1}^{\log(n)} i) = O(\log^2 n)$. The verifier's total runtime is also $O(\log^2 n)$ scalar multiplications in $\mathbb{G}_t$.

Sketch of the Knowledge Extractor. The knowledge extractor for the above protocol proceeds similarly to that for Bulletproofs (Section 14.4). After first generating a 3-transcript tree via the forking lemma (Theorem 14.1), the extractor proceeds from the leaves toward the root, and at each vertex i layers below the root it constructs the $i+1$ vectors satisfying the i inner-product equations that the prover claims to know at the point in the protocol corresponding to that vertex. For example, when the extractor comes to the root itself, the extractor has already constructed, for each child of the root, vectors $u^{(1)}$ and $\mathbf{g}^{(1)}$ such that Equations (15.11) and (15.12) hold, i.e., $\langle u^{(1)}, \mathbf{g}^{(1)} \rangle = c_{u^{(1)}}$ and $\langle \mathbf{g}^{(1)}, \Gamma^{(1)} \rangle = c_{\mathbf{g}^{(1)}}$. Note the extractor knows $\Gamma^{(1)}$, because $\Gamma^{(1)}$ is public and, unlike the verifier, the extractor need only run in polynomial time, not logarithmic time.

The Bulletproofs extractor (Section 14.4) gives a procedure to take the $\mathbf{g}^{(1)}$ vectors for all three children of the root and reconstruct a $\mathbf{g}^{(0)}$ that "explains" all three children's $\mathbf{g}^{(1)}$ vectors, in the sense that for

each child of the root, $\mathbf{g}^{(1)} = \alpha_1^{-1}\mathbf{g}_L^{(0)} + \alpha_1\mathbf{g}_R^{(0)}$, where α_1 is label of the edge connecting the root to the child under consideration. Once $\mathbf{g}^{(0)}$ is identified, the same Bulletproofs extraction procedure identifies a vector $u^{(0)}$ that explains all three children's $u^{(1)}$ vectors, i.e., such that at each child of the root, $u^{(1)} = \alpha_1 u_L^{(0)} + \alpha_1^{-1} u_R^{(0)}$, and moreover $c_{u^{(0)}} = \langle u^{(0)}, \mathbf{g}^{(0)} \rangle$.

15.4.4 Extending to a Polynomial Commitment

Extending the knowledge-of-opening protocol of Section 15.4.3 to a polynomial commitment scheme follows the outline provided at the start of the section: let $a \in \mathbb{F}_p^n$ be the coefficient vector of the polynomial $q(X) = \sum_{i=0}^{n-1} a_i X^i$ to be committed, of degree $n-1$. Then the commitment to q is just an AFGHO-commitment to a (Section 15.4.2).[13]

Recall this means the following. If $\mathbf{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $h \in \mathbb{G}$ are public commitment keys, and $w(a) \in \mathbb{G}^n = (a_1 \cdot h, \ldots, a_n \cdot h)$ is the vector of Pedersen commitments to a with key $h \in \mathbb{G}$, then the commitment to the polynomial is $c_q \coloneqq \text{IPPCom}(w(a)) = \prod_{i=1}^n e(a_i \cdot h, g_i)$.

Suppose the verifier then requests to evaluate the committed polynomial at input $z \in \mathbb{F}_p$, and the prover claims that $q(z) = v$. Then the prover has to establish that it knows a vector a such that two equalities hold: (1) $c_q = \text{IPPCom}(w(a))$, and (2) $q(r) = v$. Recall that the latter claim is equivalent to $\langle a, y \rangle = v$ where $y = (1, r, r^2, \ldots, r^{n-1})$.

Claim (1) is established by applying the protocol of Section 15.4.3 establish that $\mathcal{P}$ knows $u \coloneqq w(a)$ opening c_q. Claim (2) is established in parallel with this protocol, in close analogy to the Bulletproofs polynomial commitment scheme (Protocol 13). In slightly more detail, there are two differences from Protocol 13, aside from using the protocol of Section 15.4.3 in place of Protocol 12 to establish that $\mathcal{P}$ knows an opening of c_q:

- In Protocol 13, the verifier explicitly computes the vector $y^{(i)}$ in each round. This requires time $O(n/2^i)$ in round i, which is

[13] Recall that the same approach applies to multilinear polynomials; we restrict our attention to univariate polynomials in this section for brevity.

far too large for the polylogarithmic time verifier we set out to achieve in this section. To address this, the key observation is that $\mathcal{V}$ does not actually need to know $y^{(i)}$ for $i < \log n$. The only information the verifier actually needs to perform its checks in the protocol is the final-round value $y^{(\log n)}$. Moreover, not only does $y = (1, r, \ldots, r^{n-1})$ have "tensor structure", but so do the round-by-round updates to y. This enables $\mathcal{V}$ to compute $y^{(\log n)}$ in logarithmic time.

Specifically, for $(a, b) \in \mathbb{F}_p^2$ and a vector $v \in \mathbb{F}_p^n$, define the vector $v \otimes (a, b)$ to be $(a \cdot v, b \cdot v) \in \mathbb{F}_p^{2n}$. Then it can be checked that $y = \otimes_{i=0}^{\log(n)-1}(1, r^{2^i})$. For example,

$$(1, r) \otimes (1, r^2) = (1, r, r^2, r^3)$$

and

$$(1, r) \otimes (1, r^2) \otimes (1, r^4) = (1, r, r^2, \ldots, r^7).$$

For $i \in \{1, \ldots, \log n\}$, let $\bar{i} = \log(n) - i$. It can be checked that the above tensor structure in the vector y leads to the following equality:

$$y^{(\log n)} = \prod_{i=1}^{\log n} (\alpha_i + \alpha_i^{-1} r^{2^{\bar{i}}}). \tag{15.16}$$

Clearly, the right hand side of Equation (15.16) can be computed in $O(\log n)$ time.

For example, if $n = 4$, then

$$y^{(1)} = (1, r, r^2, r^3),$$
$$y^{(2)} = (\alpha_1 \cdot 1 + \alpha_1^{-1} \cdot r^2, \alpha_1 r + \alpha_1^{-1} r^3),$$

and

$$y^{(3)} = \alpha_2 y_L^{(2)} + \alpha_2^{-1} y_R^{(2)} = \alpha_2 \cdot \alpha_1 \cdot 1 + \alpha_2^{-1} \cdot \alpha_1 \cdot r + \alpha_2 \cdot \alpha_1^{-1} r^2 + \alpha_2^{-1} \alpha_1^{-1} r^3 = \left(\alpha_1 + \alpha_1^{-1} r^2\right)\left(\alpha_2 + \alpha_2^{-1} r\right).$$

- In the final round, round $\log n$, of the protocol of Section 15.4.3, the prover should reveal not only the group element $u^{(\log n)}$ such

that $e(u^{(\log n)}, \mathbf{g}^{(\log n)}) = c_{u^{(\log n)}}$, but also the discrete logarithm $a^* \in \mathbb{F}_p$ to base h, i.e., the field element to which $u^{(\log n)}$ is a Pedersen commitment. This enables the verifier to confirm that indeed $\langle a^*, y^{(\log n)} \rangle = a^* \cdot y^{(\log n)} = v^{(\log n)}$, where $y^{(i)}$ and $v^{(i)}$ denote the round-i values of the vectors y and v from Protocol 13.

15.4.5 Reducing Pre-Processing Time to $O(\sqrt{n})$ via Matrix Commitments

Analogy to Hyrax Polynomial Commitment. Recall that Section 14.2 used Pedersen vector commitments to give a constant-size polynomial commitment, but with linear-size evaluation proofs and verification time. Section 14.3 reduced the evaluation proof size and verification time to square root, but at the cost of increasing commitment size from constant to square root. It worked by expressing any polynomial evaluation query $q(z)$ as a vector-matrix-vector product $b^T \cdot u \cdot a$ and having the prover commit to each column of the matrix u separately. The verifier could then use homomorphism of the column commitments to compute a commitment to $u \cdot a$, and the prover could then invoke the evaluation procedure of the commitment scheme to reveal $b^T \cdot (u \cdot a)$.

Unlike the commitment scheme of Section 14.2, the schemes of this section *already* have (poly)logarithmic verification costs, following a linear-time pre-processing phase. It is nonetheless possible to combine this section's commitment schemes with the vector-matrix-vector multiplication structure in evaluation queries, to improve other costs. Specifically, the pre-processing time can be reduced from linear to square root in the degree of the committed polynomial. And unlike in Section 14.3, this improvement does not come at the cost of increasing the size of the commitment. This technique has the added benefit that polynomial evaluation queries can be answered with $O(n^{1/2})$ rather than $O(n)$ group operations by the prover (on top of the $O(n)$ $\mathbb{F}_p$ operations required to simply evaluate the polynomial at the requested point).

Commitment Phase of the Improved Protocol. Let $u \in \mathbb{F}_p^{m \times m}$ be the matrix such that for any $z \in \mathbb{F}_p$, $q(z) = b^T \cdot u \cdot a$ for some vectors $b, a \in \mathbb{F}_p^m$. The prover "in its own head" computes a Pedersen-vector

commitment c_i in $\mathbb{G}$ to each column i of u, using random generator vector $\mathbf{h} = (h_1, \ldots, h_m) \in \mathbb{G}^m$ as the commitment key. Rather than sending $(c_1, \ldots, c_m) \in \mathbb{G}^m$ explicitly to the verifier, the prover instead just sends an inner-pairing-product commitment c^* to $(c_1, \ldots, c_m)$ using $\mathbf{g} = (g_1, \ldots, g_m)$ as the commitment key. In other words, the commitment is

$$c^* := \sum_{j=1}^{m} e\left(\sum_{i=1}^{m} u_{i,j} \cdot h_i, g_j\right) = \sum_{j=1}^{m}\sum_{i=1}^{m} e\left(u_{i,j} \cdot h_i, g_j\right).$$

This is just a single element of $\mathbb{G}_t$.

Evaluation Phase of the Improved Protocol. If the verifier requests the evaluation $q(z)$, let $w, x \in \mathbb{F}_p^m$ be vectors such that $q(z) = w^T \cdot u \cdot x$, and let v denote the claimed value of $q(z)$.

Both vectors w and x themselves have the same tensor structure exploited to maintain polylogarithmic verifier time in Section 15.4.4. Recall that that protocol allowed a prover to establish knowledge of an opening $\mathbf{t} \in \mathbb{G}^m$ of a given inner-pairing-product commitment such that $\langle \mathbf{t}, y \rangle = v$, where $y \in \mathbb{F}^m$ is any public vector that can be written as a $(\log m)$-dimensional tensor product of length-2 vectors.

Denote $u \cdot x \in \mathbb{F}_p^m$ by $\mathbf{d}$. The prover first sends the verifier a Pedersen-vector commitment $c' \in \mathbb{G}$, whose prescribed value is $\langle \mathbf{d}, \mathbf{h} \rangle$. It suffices for the prover to establish three things.

- The prover knows an opening $\mathbf{t} = (t_1, \ldots, t_m) \in \mathbb{G}^m$ of the inner-pairing-product commitment c^*.
- $\langle \mathbf{t}, x \rangle = c'$. Combined with the first bullet above, this means that, if u is the matrix with column commitments given by the entries of $\mathbf{t}$, then c' is a Pedersen-vector commitment to $u \cdot x$ using key $\mathbf{h}$, i.e., $c' = \sum_{i=1}^{m} (u \cdot x)_i \cdot h_i$.
- The prover knows an opening $\mathbf{d} = (d_1, \ldots, d_m) \in \mathbb{F}_p^m$ of c' such that $\langle w, \mathbf{d} \rangle = v$. Combined with the bullet above, this means that $w^T \cdot u \cdot x = v$ as claimed by $\mathcal{P}$.

The first two items together can be established directly by the protocol of Section 15.4.4.

Since w also has tensor structure, it is tempting to also apply the protocol of Section 15.4.4 to establish that the third bullet point holds. This does not work directly because c' is not an AFGHO-commitment to $\mathbf{d}$ but rather a Pedersen-vector commitment to $\mathbf{d}$. To address this, the verifier simply transforms c' into an AFGHO-commitment as follows: for a public, randomly chosen $g \in \mathbb{G}$, the verifier lets $c'' = e(c', g)$. Then $c'' \in \mathbb{G}_t$ is an AFGHO-commitment to $\mathbf{d}$ with commitment keys $\mathbf{h} \in \mathbb{G}^n$ and $g \in \mathbb{G}$. Indeed, by bilinearity of e:

$$c'' = e(\sum_{i=1}^{m} d_i h_i, g) = \sum_{i=1}^{n} e(d_i \cdot h_i, g) = \sum_{i=1}^{n} e(h_i, d_i \cdot g).$$

Hence, the prover can invoke the protocol of Section 15.4.4 to prove it knows an opening $\mathbf{d}$ of c'' such that $\langle w, \mathbf{d} \rangle = v$.

15.4.6 Achieving $O(\log n)$ Communication and Verification Time

Recall that in Round 1 of the protocol of Section 15.4.3, the verifier is able to use the pre-processing outputs $\Delta_L^{(1)}$ and $\Delta_R^{(1)}$ to compute a commitment $c_{\mathbf{g}^{(1)}}$ to the newly-updated vector $\mathbf{g}^{(1)} = \alpha_1^{-1}\mathbf{g}_L^{(0)} + \alpha_1 \mathbf{g}_R^{(0)}$ under key $\Gamma^{(1)}$. However, the verifier is *not* able to derive a commitment to $\mathbf{u}^{(1)} = \alpha_1 u_L^{(0)} + \alpha_1^{-1} u_R^{(0)}$ under key $\Gamma^{(1)}$. Rectifying this turns out to lead to improved verification costs, of $O(\log n)$ instead of $O(\log^2 n)$.

The protocol below conceptually proceeds in $\log n$ *rounds* in close analogy with Bulletproofs and the protocol of Section 15.4.3, with each round halving the lengths of the vectors under consideration. But each round in the protocol below actually consists of 4 messages. So the number of actual rounds of communication in the final protocol is $2 \log n$ rather than $\log n$.

The Setup. Suppose at the start of Round i, the prover has claimed to know two vectors $u^{(i-1)}, \mathbf{g}^{(i-1)} \in \mathbb{G}^{n \cdot 2^{-(i-1)}}$ satisfying the following three equations:

$$\langle u^{(i-1)}, \mathbf{g}^{(i-1)} \rangle = c_1, \tag{15.17}$$

$$\langle u^{(i-1)}, \Gamma^{(i-1)} \rangle = c_2 \tag{15.18}$$

$$\langle \mathbf{g}^{(i-1)}, \Gamma^{(i-1)} \rangle = c_3. \tag{15.19}$$

In Round 1, the following choices ensure that the three equations above capture $\mathcal{P}$'s claim that the protocol as a whole is meant to verify, namely that $\mathcal{P}$ knows $u^{(0)}$ satisfying

$$\langle u^{(0)}, \mathbf{g}^{(0)} \rangle = c_{u^{(0)}}. \tag{15.20}$$

Set $\Gamma^{(0)} = \mathbf{g}^{(0)}$, and set $c_1 = c_2 = c_{u^{(0)}}$. Finally, set $c_3 = \langle \mathbf{g}^{(0)}, \mathbf{g}^{(0)} \rangle$, which is a quantity that can be included in the output of pre-processing, as it is independent of $u^{(0)}$.[14] Under the above settings of $\Gamma^{(0)}$, c_1, c_2, and c_3, the validity of Equations (15.17)-(15.19) for $i = 1$ is equivalent to the validity of Equation (15.20).

Description of Round i. The purpose of Round i is to reduce the three equations above to three equations of the same form, but over vectors $u^{(i)}$ and $g^{(i)}$ that are shorter than $u^{(i-1)}, \mathbf{g}^{(i-1)}$ by a factor of 2. This is done as follows (below, we intermingle the description of each step of the protocol with intuitive justification for the step. A standalone protocol description is in Protocol 14).

- The prover begins by sending four quantities $D_{1L}, D_{1R}, D_{2L}, D_{2L} \in \mathbb{G}_t$ claimed to be AFGHO-commitments to the left and right halves of $u^{(i-1)}$ and $\mathbf{g}^{(i-1)}$ under key $\Gamma^{(i)}$. Conceptually, the point of sending these four commitments is to enable the verifier to homomorphically compute commitments under $\Gamma^{(i)}$ to $u^{(i)}$ and $\mathbf{g}^{(i)}$ (defined later in the protocol).

- The verifier chooses a random $\beta \in \mathbb{F}_p$ and sends it to $\mathcal{P}$. Conceptually, β is used to "take a random linear combination" of the three equations, yielding a single "equivalent" equation. Specifically, observe that if Equations (15.17)–(15.19) hold, then so does the following equation:

$$\begin{aligned} &\langle u^{(i-1)} + \beta \Gamma^{(i-1)}, \mathbf{g}^{(i-1)} + \beta^{-1} \Gamma^{(i-1)} \rangle \\ &\quad = c_1 + \beta^{-1} c_2 + \beta c_3 + \langle \Gamma^{(i-1)}, \Gamma^{(i-1)} \rangle. \end{aligned} \tag{15.21}$$

[14] More generally, Round i of the protocol will require that the pre-processing also output the quantity $\langle \Gamma^{(i-1)}, \Gamma^{(i-1)} \rangle$.

Meanwhile, it turns out that if the prover can establish that it knows $u^{(i-1)}$ and $\mathbf{g}^{(i-1)}$ such that Equation (15.21) holds even for just three values of β, then in fact $u^{(i-1)}$ and $\mathbf{g}^{(i-1)}$ satisfy Equations (15.17)–(15.19). This is because, if any one of Equations (15.17)–(15.19) fail to hold, then Equation (15.21) can only hold for at most 2 values of $\beta \in \mathbb{F}_p$, as the left hand and right hand sides of Equation (15.21) are distinct Laurent polynomials in β, and hence can agree on at most 2 points.

So, conceptually speaking, the remainder of Round i will be devoted to proving that Equation (15.21) holds for the verifier's random choice of β. As we detail below, the remainder of the round accomplishes this essentially by applying the standard Bulletproofs iteration to Equation (15.21), i.e., of randomly choosing an $\alpha \in \mathbb{F}_p$ and using it to "randomly combine" the left and right halves of $u^{(i-1)}$ and $\mathbf{g}^{(i-1)}$ respectively to obtain new vectors $u^{(i)}$ and $\mathbf{g}^{(i)}$of half the length.

To this end, let

$$w_1 = u^{(i-1)} + \beta\Gamma^{(i-1)} \tag{15.22}$$

and

$$w_2 = \mathbf{g}^{(i-1)} + \beta^{-1}\Gamma^{(i-1)}, \tag{15.23}$$

and let w_{1L} and w_{1R} denote the left and right halves of w_1 and similarly for w_{2L} and w_{2R}.

- The prover sends cross-terms v_L and v_R claimed to equal $\langle w_{1L}, w_{2R}\rangle$ and $\langle w_{1R}, w_{2L}\rangle$.
- The verifier chooses a random $\alpha \in \mathbb{F}_p$ and sends it to the prover.
- The prover defines:

$$u^{(i)} = \alpha \cdot w_{1L} + \alpha^{-1} w_{1R} \tag{15.24}$$

$$\mathbf{g}^{(i)} = \alpha^{-1} \cdot w_{2L} + \alpha w_{2R}. \tag{15.25}$$

- The verifier does not know $u^{(i)}$ or $\mathbf{g}^{(i)}$ but can homomorphically compute the following three quantities:

– $\langle u^{(i)}, \mathbf{g}^{(i)} \rangle$. Indeed, if Equations (15.17)–(15.19) hold and v_L, v_W are prescribed, then a straightforward consequence of Equation (15.21) is that

$$\langle u^{(i)}, \mathbf{g}^{(i)} \rangle = c_1 + \beta^{-1} c_2 + \beta c_3 + \langle \Gamma^{(i-1)}, \Gamma^{(i-1)} \rangle + \alpha v_L + \alpha^{-1} v_R. \tag{15.26}$$

Note that the verifier has access to all terms on the right hand side of Equation (15.26) (as mentioned in Footnote 14, $\langle \Gamma^{(i-1)}, \Gamma^{(i-1)} \rangle$ can be computed in pre-processing.)

– $\langle u^{(i)}, \Gamma^{(i)} \rangle$. Indeed, if D_{1L} and D_{1R} are as prescribed, then Equations (15.22) and (15.24) imply that:

$$\langle u^{(i)}, \Gamma^{(i)} \rangle = \alpha D_{1L} + \alpha^{-1} D_{1R} + \alpha\beta \Delta_L^{(i)} + \alpha^{-1}\beta \Delta_R^{(i)}. \tag{15.27}$$

Recall that, here, $\Delta_L^{(i)} = \langle \Gamma_L^{(i-1)}, \Gamma^{(i)} \rangle$ and $\Delta_R^{(i)} = \langle \Gamma_R^{(i-1)}, \Gamma^{(i)} \rangle$ are commitments to the left and right halves of $\Gamma^{(i-1)}$ computed during pre-processing.

– $\langle \mathbf{g}^{(i)}, \Gamma^{(i)} \rangle$. If D_{2L} and D_{2R} are as prescribed, then Equations (15.23) and (15.25) imply that

$$\langle \mathbf{g}^{(i)}, \Gamma^{(i)} \rangle = \alpha^{-1} D_{2L} + \alpha D_{2R} + \alpha^{-1}\beta^{-1} \Delta_L^{(i)} + \alpha\beta^{-1} \Delta_R^{(i)}. \tag{15.28}$$

- Let c'_1, c'_2, and c'_3 denote the above three quantities that the verifier homomorphically computes. Round $i+1$ is then devoted to showing that indeed the prover knows $u^{(i)}, \mathbf{g}^{(i)}$ such that the following three equations indeed hold:

$$\langle u^{(i)}, \mathbf{g}^{(i)} \rangle = c'_1$$
$$\langle u^{(i)}, \Gamma^{(i)} \rangle = c'_2,$$
$$\langle \mathbf{g}^{(i)}, \Gamma^{(i)} \rangle = c'_3.$$

The above protocol is summarized in Protocol 14.

Sketch of the Extraction Analysis. The knowledge extractor proceeds similarly to the one for the protocol of Section 15.4.3. First, it constructs a 3-transcript tree for the protocol, of depth $2 \log n$ (the 2 comes in because each of the $\log n$ "conceptual rounds" in the protocol description actually consists of 2 communication rounds). For the remainder of this sketch, we use the phrase "round" to refer to a conceptual round.

As usual, the extractor proceeds from the leaves toward the root. At each node of distance $2i$ from the root (capturing the start of round $i+1$ of the protocol), it constructs vectors $u^{(i)}$ and $\mathbf{g}^{(i)}$ that "explain" the prover's messages in all transcripts of the sub-tree rooted at that node.

Suppose by way of induction that the extractor has already succeeded in reconstructing such vectors for all vertices in the tree corresponding to rounds $i+1$ and later. Consider a tree vertex corresponding to round $i+1$ of the protocol, and let α, β denote the random group elements selected by the verifier in round i. Because AFGHO commitments are computationally binding and the extractor is efficient, one can argue that for the vector $u^{(i+1)}$ reconstructed by the extractor for that vertex, there is an (efficiently computable) vector $z = (z_L, z_R)$ such that

$$u^{(i+1)} = \alpha z_L + \alpha^{-1} z_R + \alpha\beta\Gamma_L^{(i-1)} + \alpha^{-1}\beta\Gamma_R^{(i-1)}.$$

This is because Equation (15.27) ensures that the commitment to $u^{(i+1)}$ is a commitment to a vector of this form. Similarly, one can argue that there is an (efficiently computable) vector $t = (t_L, t_R)$ such that the reconstructed vector $\mathbf{g}^{(i+1)}$ is of the form

$$\alpha^{-1} t_L + \alpha t_R + \alpha^{-1}\beta^{-1}\Gamma_L^{(i-1)} + \alpha\beta\Gamma_R^{(i-1)}.$$

Now consider any node of the transcript tree capturing the second message of round i of the protocol. Because the second message of Round i applies the Bulletproofs update procedure to commitments to w_1 and w_2 under commitment key $\Gamma^{(i)}$, the Bulletproofs extractor is able to reconstruct vectors w_1 and w_2 such that Equations (15.24) and (15.25) hold, i.e., w_1 and w_2 "explain" the reconstructed vectors $u^{(i+1)}$ and $\mathbf{g}^{(i+1)}$ for all child vertices of the node. Moreover, the previous paragraph implies that there are efficiently computable vectors $u^{(i-1)}$ and $\mathbf{g}^{(i-1)}$

such that Equations (15.22) and (15.23) hold, i.e., $w_1 = u^{(i-1)} + \beta\Gamma^{(i-1)}$ and $w_2 = \mathbf{g}^{(i-1)} + \beta^{-1}\Gamma^{(i-1)}$.

All that is left is to argue that $u^{(i-1)}$ and $\mathbf{g}^{(i-1)}$ satisfy Equations (15.17)–(15.19). That $u^{(i-1)}$ and $\mathbf{g}^{(i-1)}$ satisfy Equation (15.21) for the three values of β appearing in the 3-transcript tree follows from the following three facts. First, the verifier's commitment computation in Equation (15.26) applies the Bulletproofs update procedure to confirm that the prover knows vectors $w_1 = u^{(i-1)} + \beta\Gamma^{(i-1)}$ and $w_2 = \mathbf{g}^{(i-1)} + \beta^{-1}\Gamma^{(i-1)}$ such that Equation (15.21) holds. Second, as argued above, w_1 and w_2 "explain" the reconstructed vectors $u^{(i+1)}$ and $\mathbf{g}^{(i+1)}$ and hence they are exactly the vectors that the Bulletproofs knowledge extractor would compute if the Bulletproofs update procedure were applied to Equation (15.21). This means that the vectors computed by the extractor satisfy Equation (15.21). Finally, as explained in the protocol description itself, if $u^{(i-1)}$ and $\mathbf{g}^{(i-1)}$ satisfy Equation (15.21) for three or more values of β, then they also satisfy Equations (15.17)–(15.19).

Protocol 14 A transparent argument of knowledge of vectors $u, \mathbf{g} \in \mathbb{G}^n$ such that $\langle u, \mathbf{g}\rangle = c_1$, $\langle u, \Gamma\rangle = c_2$, and $\langle \mathbf{g}, \Gamma\rangle = c_3$, where $\Gamma \in \mathbb{G}^n$ is a public vector summarized via a linear-time pre-processing phase (Section 15.4.3). Here, $\mathbb{G}$ is an additive group and $\langle u, \mathbf{g}\rangle = \sum_{i=1}^{n} e(u_i, g_i)$ denotes the inner-pairing-product between u and $\mathbf{g}$. The protocol consists of $2\log_2 n$ communication rounds, with 6 $\mathbb{G}_t$ elements communicated from prover to verifier per round. Total verifier time is dominated by $O(\log n)$ scalar multiplications in $\mathbb{G}_t$. For simplicity, we present the protocol for a symmetric pairing, though inner-pairing-product commitments are only binding for asymmetric pairings (in particular, under the SXDH assumption, which only holds for asymmetric pairings).

1: Let $\mathbb{G}$ be an additive cyclic group of prime order p with with bilinear map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_t$.

2: Input is $c_1 = \langle u, \mathbf{g}\rangle$, $c_2 = \langle u, \Gamma\rangle$, and $c_3 = \langle \mathbf{g}, \Gamma\rangle$, where $u, \mathbf{g}, \Gamma \in \mathbb{G}^n$. Verifier only knows c_1, c_2, c_3 and the following quantities computed during pre-processing: $\langle \Gamma, \Gamma\rangle$, as well as AFGHO commitments $\Delta_L = \langle \Gamma_L, \Gamma'\rangle$ and $\Delta_R = \langle \Gamma_R, \Gamma'\rangle$ to the left and right halves of Γ under public commitment key $\Gamma' \in \mathbb{G}^{n/2}$.

3: If $n = 1$, Prover sends u, $\mathbf{g}$, and Γ, to the verifier and the verifier checks that $c_1 = \langle u, \mathbf{g} \rangle$, $c_2 = \langle u, \Gamma \rangle$, $c_3 = \langle \mathbf{g}, \Gamma \rangle$, $\Delta_L = \langle \Gamma_L, \Gamma' \rangle$ and $\Delta_R = \langle \Gamma_R, \Gamma' \rangle$.
4: Otherwise, $\mathcal{P}$ sends $D_{1L}, D_{1R}, D_{2L}, D_{2L} \in \mathbb{G}_t$ claimed to be AFGHO-commitments to the left and right halves of u and $\mathbf{g}$ under key Γ'.
5: $\mathcal{V}$ chooses a random $\beta \in \mathbb{F}_p$ and sends it to $\mathcal{P}$.
6: $\mathcal{P}$ sets $w_1 = u + \beta\Gamma$ and $w_2 = \mathbf{g} + \beta^{-1}\Gamma$. Let w_{1L} and w_{1R} denote the left and right halves of w_1 and similarly for w_{2L} and w_{2R}.
7: $\mathcal{P}$ sends $v_L, v_R \in \mathbb{G}_t$ claimed to equal $\langle w_{1L}, w_{2R} \rangle$ and $\langle w_{1R}, w_{2L} \rangle$.
8: $\mathcal{V}$ chooses a random $\alpha \in \mathbb{F}_p$ and sends it to $\mathcal{P}$
9: $\mathcal{P}$ sets $u' = \alpha \cdot w_{1L} + \alpha^{-1} \cdot w_{1R}$ and $\mathbf{g}' = \alpha^{-1} \cdot w_{2L} + \alpha \cdot w_{2R}$.
10: $\mathcal{V}$ sets:

$$c'_1 = c_1 + \beta^{-1}c_2 + \beta c_3 + \langle \Gamma, \Gamma \rangle + \alpha v_L + \alpha^{-1} v_R.$$
$$c'_2 = \alpha D_{1L} + \alpha^{-1} D_{1R} + \alpha\beta\Delta_L + \alpha^{-1}\beta\Delta_R.$$
$$c'_3 = \alpha^{-1} D_{2L} + \alpha D_{2R} + \alpha^{-1}\beta^{-1}\Delta_L + \alpha\beta^{-1}\Delta_R.$$

11: $\mathcal{V}$ and $\mathcal{P}$ apply the protocol recursively to prove that $\mathcal{P}$ knows vectors $u', \mathbf{g}' \in \mathbb{G}^{n/2}$ such that $\langle u', \mathbf{g}' \rangle = c_1$, $\langle u', \Gamma' \rangle = c_2$, and $\langle \mathbf{g}', \Gamma' \rangle = c_3$.