12

Σ-Protocols and Commitments from Hardness of Discrete Logarithm

12.1 Cryptographic Background

12.1.1 A Brief Introduction to Groups

Informally, a group $\mathbb{G}$ is any set equipped with an operation that behaves like multiplication. To be more precise, a group is a collection of elements equipped with a binary operation (which we denote by $\cdot$ and refer to in this monograph as multiplication) that satisfies the following four properties.

- Closure: the product of two elements in $\mathbb{G}$ are also in $\mathbb{G}$, i.e., for all $a, b \in \mathbb{G}$, $a \cdot b$ is also in $\mathbb{G}$.
- Associativity: for all $a, b, c \in \mathbb{G}$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
- Identity: there an element denoted $1_{\mathbb{G}} \in \mathbb{G}$ such that $1_{\mathbb{G}} \cdot g = g \cdot 1_{\mathbb{G}} = g$ for all $g \in \mathbb{G}$.
- Invertibility: For each $g \in \mathbb{G}$, there is an element h in $\mathbb{G}$ such that $g \cdot h = 1_{\mathbb{G}}$. This element h is denoted g^{-1}.

One important example of a group is the set of nonzero elements of any field, which forms a group under the field multiplication operation.

This is referred to as the multiplicative group of the field. Another is the the set of invertible matrices, which forms a group under the matrix multiplication operation. Note that matrix multiplication is not commutative. In cases where the group operation *is* commutative, the group is called abelian.

Sometimes it is convenient to think of the group operation as addition rather than multiplication, in which case the operation is denoted with a $+$ symbol instead of $\cdot$. Whether a group is considered additive or multiplicative is a matter of context, convenience, and convention. As an example, any field is a group under the field's addition operation, and the set of all $n \times n$ matrices over the field form a group under the matrix addition operation. For these groups it is of course natural to denote the group operation with $+$ rather than $\cdot$. Henceforth in this monograph, *with the lone exception of two subsections in the Section 14 and 15 (Sections 14.4 and 15.4)*, we will exclusively refer to multiplicative groups, using $\cdot$ to denote the group operation.

A group $\mathbb{G}$ is said to be *cyclic* if there is some group element g such that all group elements can be generated by repeatedly multiplying g with itself, i.e., if every element of $\mathbb{G}$ can be written as g^i for some positive integer i. Here, in analogy to how exponentiation refers to repeated multiplication in standard arithmetic, g^i denotes $\underbrace{g \cdot g \cdot \cdots \cdot g}_{i \text{ copies of } g}$.[1] Such an element of g is called a generator for $\mathbb{G}$. Any cyclic group is abelian.

The cardinality $|\mathbb{G}|$ is called the *order* of $\mathbb{G}$. A basic fact from group theory is that for any element $g \in \mathbb{G}$, $g^{|\mathbb{G}|} = 1_{\mathbb{G}}$. This implies that when considering any group exponentiation, i.e., g^ℓ for some integer ℓ, reducing the exponent ℓ modulo the group size $|\mathbb{G}|$ does not change anything: for any integer ℓ, if $z \equiv \ell \mod |\mathbb{G}|$, then $g^\ell = g^z$.

A *subgroup* of a group $\mathbb{G}$ is a subset $\mathbb{H}$ of $\mathbb{G}$ that itself forms a group under the same binary operation as $\mathbb{G}$ itself. Another basic fact from group theory states that the order of any subgroup $\mathbb{H}$ of $\mathbb{G}$ divides the order of $\mathbb{G}$ itself. A consequence is that any prime-order group $\mathbb{G}$ is cyclic: in fact, each non-identity element $g \in \mathbb{G}$ is a group generator.

[1] Similarly, g^{-i} denotes the ith power of the inverse of g, i.e, $\left(g^{-1}\right)^i$.

This is because the set $\{g, g^2, g^3, \dots, \}$ of powers of g is easily seen to be a subgroup of $\mathbb{G}$, referred to as the subgroup *generated by* g. Since $g \geq 1_{\mathbb{G}}$, its order is an integer strictly between 1 and $|\mathbb{G}|$, and since $|\mathbb{G}|$ is prime, the order must equal $|\mathbb{G}|$. Hence, the subgroup generated by g in fact equals the entire group $\mathbb{G}$.

12.1.2 The Discrete Logarithm Problem and Background on Elliptic Curves

12.1.2.1 Discrete Log Problem

For a specified group $\mathbb{G}$ the discrete logarithm problem takes as input two group elements g and h, and the goal is to output a positive integer i such that $g^i = h$ (if $\mathbb{G}$ is of prime order then such an i is guaranteed to exist).

The discrete logarithm problem is believed to be computationally intractable in certain groups $\mathbb{G}$. In modern cryptography, the groups used are typically cyclic subgroups of groups defined via elliptic curves over finite fields, or the multiplicative group of integers modulo a very large prime p. An important caveat is that quantum computers can solve the discrete logarithm problem in polynomial time via Shor's algorithm [233]. Hence, cryptosystems whose security is based on the assumed hardness of the discrete logarithm problem are not post-quantum secure.

12.1.2.2 Elliptic Curve Groups

Though it is a fascinating and important topic, we will not go into great detail on elliptic curve cryptography in this monograph, and restrict ourselves to the following comments. Any elliptic curve group is defined with respect to a (finite) field $\mathbb{F}$, called the *base field* of the curve. Group elements correspond to pairs of points $(x, y) \in \mathbb{F} \times \mathbb{F}$ that satisfy an equation of the form $y^2 = x^3 + ax + b$ for designated field elements a and b.[2] Given two elements P and Q of the group, the precise definition of the group product $P \cdot Q$ will not be important in this monograph, but for the interested reader, here is a rough sketch.

[2]The group also contains one extra element known as the *point at infinity*; this detail will not be relevant to this monograph.

Sketch of the Group Operation. Recalling that P and Q each consist of a *pair* of elements of the base field $\mathbb{F}$ satisfying the curve equation, we can visualize P and Q as two points in the two-dimensional plane. Draw a line through these two points. This line typically turns out to intersect the elliptic curve at a third point $R = (x, y)$. The group product $P \cdot Q$ is defined to equal $(x, -y)$. Here, if $R = (x, y)$ is on the curve $y^2 = x^3 + ax + b$, then so is $(x, -y)$, owing to the fact that $y^2 = (-y)^2$.

Algorithms for Computing Discrete Logarithms. The fastest known classical algorithm to solve the Discrete Logarithm problem over most elliptic curve groups used in practice runs in time $O(\sqrt{|\mathbb{G}|})$.[3] Under the assumption that these are in fact the fastest attacks possible, this means that to obtain "λ bits of security" (meaning security against attackers running in time 2^λ, see Footnote 7 in Section 7.3.2.2), one should use an elliptic curve group of order $2^{2\lambda}$. For example, a popular elliptic curve called Curve25519, which is defined over base field $\mathbb{F}$ of size $2^{255} - 19$, defines a cyclic group of order close to 2^{252}; hence, this group provides slightly less than 128 bits of security [52].

One reason Curve25519 is popular is efficiency of group operations: the computational bottleneck in multiplying elliptic curve group elements turns out to be performing multiplications in the base field $\mathbb{F}$. Because $p = 2^{255} - 19$ is a power of two minus a small constant, multiplication in $\mathbb{F}$ can be implemented more efficiently than if p did not have this form. As general (rough) guidance, the time cost of performing one group multiplication in an elliptic curve group defined over field $\mathbb{F}$ is typically about 10 times as expensive as performing one multiplication in $\mathbb{F}$.

Scalar Field vs. Base Field. Elliptic curve groups used in practice are chosen to have large prime order.[4] This is because there are known

[3] See, for example, the wikipedia article on Pollard's rho algorithm https://en.wikipedia.org/wiki/Pollard%27s_rho_algorithm_for_logarithms, introduced in [213].

[4] A subtlety arising in modern elliptic curves used in cryptography is that the group order is typically a small constant—typically 4 or 8—times a prime. For

algorithms, such as the Pohlig-Hellman algorithm [210], that can compute discrete logarithms in group $\mathbb{G}$ in time proportional to the largest prime-order subgroup of $\mathbb{G}$, The field of size equal to the (prime) order of the elliptic curve group $\mathbb{G}$ is typically referred to as the *scalar field* of $\mathbb{G}$.

Recall that prime-order groups $\mathbb{G}$ are cyclic: for any group element $g \neq 1_{\mathbb{G}}$, we can write $\mathbb{G} = \{g^x : x = 0, 1, \ldots, |\mathbb{G}| - 1\}$. Hence, we can think of elements x of the scalar field of $\mathbb{G}$ as *exponents*, when expressing $\mathbb{G}$ as powers of a generator g.

Note that the scalar field of the elliptic curve group is *not* the same as the base field $\mathbb{F}$ over which the curve is defined.[5, 6] This is particularly relevant to the concrete performance of "SNARK composition", a topic discussed at length later in this survey (Section 18.2).

Readers interested in more detailed (and illustrated) introductions to elliptic curve groups are directed to [73], [109].

12.2 Schnorr's Σ-Protocol for Knowledge of Discrete Logarithms

In this section, we describe several perfect honest-verifier zero-knowledge proof systems. These proof systems have a very simple structure, involving only three messages exchanged between prover and verifier. They are *special-purpose*, meaning that, as standalone objects, they do not solve **NP**-complete problems such as circuit satisfiability. Rather, they solve specific problems including (a) establishing that the prover has knowledge of a discrete logarithm of some group element (Section 12.2.2); (b) allowing the prover to cryptographically commit to group elements without revealing the committed group element to the verifier until

example, the order of Curve25519 is 8 times a prime. For this reason, implementations typically work over the prime-order subgroup of the full elliptic curve group, or they add a layer of abstraction that exposes a prime-order interface. The interested reader is directed to [147] and https://ristretto.group/why_ristretto.html for an overview of these details.

[5] However, the sizes of the two fields cannot be too far apart: a result known as Hasse's theorem [148] states that for all elliptic curve groups $\mathbb{G}$ over field $\mathbb{F}$, $|\mathbb{G}| - (|\mathbb{F}| + 1) \leq 2\sqrt{|\mathbb{F}|}$.

[6] The discrete logarithm problem is easy in elliptic curve groups for which the base field and scalar field are the same. So for all curves used in cryptography, the two fields are different.

later (Section 12.3); and (c) establishing product relationships between committed values (Section 12.3.2).

While the protocols covered in this section are special-purpose, we will see (e.g., Section 13.1) that they can be *combined* with general-purpose protocols such as IPs, IOPs, and MIPs to obtain general-purpose zk-SNARKs.

12.2.1 Σ-Protocols

The presentation in this section closely follows other authors [85]. A relation $\mathcal{R}$ specifies a collection of "valid" instance-witness pairs (h, w). For example, given a group $\mathbb{G}$ and a generator g, the discrete logarithm relation $\mathcal{R}_{\mathsf{DL}}(\mathbb{G}, g)$ is the collection of pairs $(h, w) \in \mathbb{G} \times \mathbb{Z}$ such that $h = g^w$.

A Σ-protocol for a relation $\mathcal{R}$ is a 3-message public coin protocol between prover and verifier in which both prover and verifier know a public input h, and the prover knows a witness w such that $(h, w) \in \mathcal{R}$.[7] Let us denote the three messages by (a, e, z), with the prover first sending a, the verifier responding with a challenge e (chosen via public random coin tosses), and the prover replying with z. A Σ-protocol is required to satisfy perfect completeness, i.e., if the prover follows the prescribed protocol then the verifier will accept with probability 1. It is also required to satisfy two additional properties.

Special Soundness: There exists a polynomial time algorithm $\mathcal{Q}$ such that, when given as input a pair of accepting transcripts (a, e, z) and (a, e', z') with $e \neq e'$, $\mathcal{Q}$ outputs a witness w such that $(h, w) \in \mathcal{R}$.

Intuitively, special soundness guarantees that if, after sending its first message in the Σ-protocol, the prover is prepared to answer more than one challenge from the verifier, then the prover much *know* a witness w such that $(h, w) \in \mathcal{R}$.

Honest Verifier Perfect Zero-Knowledge. There must be a randomized polynomial time simulator that takes as input the public input h

[7] The term Σ-protocol was coined because pictorial diagrams of 3-message protocols are vaguely reminiscent of the Greek letter Σ.

from the Σ-protocol, and outputs a transcript (a, e, z) such that the distribution over transcripts output by the simulator is *identical* to the distribution over transcripts produced by the honest verifier in the Σ-protocol interacting with the honest prover.

Remark 12.1. Special soundness implies that, if the verifier in the Σ-protocol were to be given "rewinding access" to the prover, then the Σ-protocol would *not* be zero-knowledge. That is, special soundness says that if the verifier could run the protocol to completion to obtain the transcript (a, e, z), then "rewind" to just after the prover sent its first message a, and restart the protocol with a new challenge e', then, assuming both transcripts lead to acceptance, the verifier would learn a witness (see Section 12.2.3 for additional discussion of this witness-extraction procedure). This clearly violates zero-knowledge if witnesses are assumed to be intractable to compute. Hence, the honest-verifier zero-knowledge property of Σ-protocols only holds if the verifier is never allowed to run the prover more than once with the same first prover message a.

12.2.2 Schnorr's Σ-Protocol for the Discrete Logarithm Relation

Let $\mathbb{G}$ be a cyclic group of prime order generated by g. Recall that in any Σ-protocol for the discrete logarithm relation, $\mathcal{P}$ holds (h, w) such that $h = g^w$ in $\mathbb{G}$, while $\mathcal{V}$ knows h and g.[8]

To convey the intuition behind Schnorr's [223] protocol, we describe a number of progressively more sophisticated attempts at designing a proof of knowledge for the discrete logarithm relation.

Attempt 1. The most straightforward possible proof of knowledge for any relation is to simply have the prover $\mathcal{P}$ send the witness w for the public input h, so the verifier $\mathcal{V}$ can check that $(h, w) \in \mathcal{R}$. However, this reveals w to the verifier, violating zero-knowledge (assuming the verifier could not efficiently compute the witness on her own).

[8] In this monograph we only consider Σ-protocols for groups of prime order. Σ-protocols (and related proof systems) for problems over composite and hidden-order groups have also been studied, see for example [18], [63], [118].

Attempt 2. $\mathcal{P}$ could pick a random value $r \in \{0, \ldots, |\mathbb{G}| - 1\}$ and send $(w + r) \mod |\mathbb{G}|$ to $\mathcal{V}$. This totally "hides" w in that $(w + r) \mod |\mathbb{G}|$ is a uniform random element of the set $\{0, \ldots, |\mathbb{G}| - 1\}$, and hence this message does not violate zero-knowledge. But for the same reason, $(w + r) \mod |\mathbb{G}|$ is useless to $\mathcal{V}$ as far as ensuring soundness goes. It is simply a random number, which $\mathcal{V}$ could have generated on her own.

Attempt 3. To address the issue in Attempt 2 that $(w + r) \mod |\mathbb{G}|$ is useless on its own, $\mathcal{P}$ could first send r, followed by a value z claimed to equal $(w + r) \mod |\mathbb{G}|$. $\mathcal{V}$ checks that $g^r \cdot h = g^z$.

This protocol is complete and sound, but it is not zero-knowledge. Completeness is easy to check, while special soundness holds because if $g^r \cdot h = g^z$, then $g^{z-r} = h$, i.e, $z - r$ is a witness. That is, a witness can be extracted from even a single accepting transcript. Of course, for the same reason, this protocol is not zero-knowledge.

Effectively, Attempt 3 broke w into two pieces, $z \coloneqq w + r \mod |\mathbb{G}|$ and r, such that each piece individually reveals no information to $\mathcal{V}$ (because each is simply a random element of $\{0, 1, \ldots |\mathbb{G}|\}$). But together, the pieces reveal the witness w to the verifier (since $z - r = w$). Hence, this attempt is no closer to satisfying zero-knowledge than Attempt 1.

Attempt 4. We could modify Attempt 3 above so that, rather than $\mathcal{P}$ sending r to $\mathcal{V}$, $\mathcal{P}$ instead sends a group element a claimed to equal g^r, followed by a number z exactly as in Attempt 3, i.e., z is claimed to equal $(w + r) \mod |\mathbb{G}|$. $\mathcal{V}$ checks that $a \cdot h = g^z$.

This attempt turns out to be complete and zero-knowledge, but not special sound. Completeness is easy to verify: if the prover is honest, then $a \cdot h = g^r \cdot h = g^{r+w} = g^z$. It is zero-knowledge because a simulator can choose an element $z \in \{0, 1, \ldots, |\mathbb{G}| - 1\}$ at random, and then set a to be $g^z \cdot h^{-1}$, and output the transcript (a, z). This generates a transcript distributed identically to that generated by the honest prover.

Conceptually, while the honest prover in Attempt 4 chooses a random group element $a = g^r$ and then chooses z to be the unique number

such that the verifier accepts (a, z), the simulator chooses z first at random and then chooses a to be the unique group element causing the verifier to accept (a, z). The two distributions are identical—in both cases, a and z are individually uniformly distributed (a from $\mathbb{G}$ and z from $\{0, 1, \ldots, |\mathbb{G}| - 1\}$), with the value of a determining the value of z and vice versa.

Sadly, Attempt 4 is not special sound for the same reason it is zero-knowledge. The simulator is able to generate accepting transcripts, and since the protocol is totally non-interactive (there is no challenge sent by verifier to prover), the simulator itself acts as a "cheating" prover capable of convincing the verifier to accept despite not knowing a witness.

Comparison of Attempts 3 and 4. The reason Attempt 4 is zero-knowledge while Attempt 3 is not is that whereas Attempt 3 has $\mathcal{P}$ send r "in the clear", Attempt 4 "hides" r in the exponent of g, and accordingly the subtraction of r from z by the verifier in Attempt 4 happens "in the exponent" of g rather than in the clear.

The fact that Attempt 4 is zero-knowledge may seem surprising at first. After all, on an information-theoretic level, r can be derived from g^r, and then the witness $z - r$ can be computed, and this may seem like a violation of zero-knowledge. But the derivation of r requires finding the discrete logarithm of g^r, which is just as hard as deriving a witness w (i.e., a discrete logarithm of h). In summary, the fact that Attempt 4 reveals r to the verifier in an information-theoretic sense does not contradict zero-knowledge, because the public input $h = g^w$ itself information-theoretically specifies w in the same way that $a = g^r$ information-theoretically specifies r. In fact, g^r combined with $(w + r) \bmod |\mathbb{G}|$ does not actually reveal any new information to the verifier beyond what was already revealed by h itself.

Schnorr's Σ-Protocol. Protocol 3 describes Schnorr's Σ-protocol. Essentially, Schnorr's protocol modifies Attempt 4 so that, after $\mathcal{P}$ sends a but before $\mathcal{P}$ sends z, the verifier sends a random challenge e drawn from $\{0, 1, \ldots, |\mathbb{G}| - 1\}$. Compared to Attempt 4, the verifier's check

is modified so that it will pass if $z = w \cdot e + r$ (Attempt 4 above is identical to Schnorr's protocol with the verifier's challenge e fixed to 1).

These modifications to Attempt 4 do not alter the completeness or zero-knowledge properties of the protocol. The intuition for why Schnorr's protocol is special sound is that if $\mathcal{P}$'s first message is $a = g^r$ and $\mathcal{P}$ can produce accepting transcripts (a, e, z) and (a, e', z') with $e \neq e'$, then $\mathcal{V}$'s acceptance criterion implies that $z = w \cdot e + r$ and $z' = w \cdot e' + r$.[9] These are two linearly independent equations in two unknowns, namely w and r. Hence, one can take these two transcripts and efficiently solve for both w and r.

Protocol 3 Schnorr's Σ-protocol for the Discrete Logarithm Relation

1: Let $\mathbb{G}$ be a (multiplicative) cyclic group of prime order with generator g.
2: Public input is $h = g^w$, where only prover knows w.
3: $\mathcal{P}$ picks a random number r in $\{0, \ldots, |\mathbb{G}| - 1\}$ and sends $a \leftarrow g^r$ to the verifier.
4: Verifier responds with a random element $e \in \{0, \ldots, |\mathbb{G}| - 1\}$.
5: Prover responds with $z \leftarrow (we + r) \mod |\mathbb{G}|$.
6: Verifier checks that $a \cdot h^e = g^z$.

We now turn to formally proving that Schnorr's protocol satisfies perfect completeness, special soundness, and honest-verifier zero-knowledge.

Perfect completeness is easy to establish: if $a \leftarrow g^r$ and $z \leftarrow (we + r) \mod |\mathbb{G}|$, then

$$a \cdot h^e = g^r \cdot h^e = g^r \cdot (g^w)^e = g^{r+we} = g^z,$$

so the verifier accepts transcript (a, e, z).

Special soundness: Suppose we are given two accepting transcripts (a, e, z) and (a, e', z') with $e \neq e'$. We must show that a witness w^* can be extracted in polynomial time from these two transcripts.

[9] Similar to Attempt 4, note that $\mathcal{V}$'s check on transcript (a, e, z) in Schnorr's protocol confirms "in the exponent" that $z = w \cdot e + r$. $\mathcal{V}$ is able to perform this check in the exponent despite only knowing z, g^r, and h (in particular, without knowing r and w, which are the discrete logarithms of g^r and h).

Let $(e-e')^{-1}$ denote the multiplicative inverse of $e-e'$ modulo $|\mathbb{G}|$, i.e., $(e-e')^{-1}$ denotes a number ℓ such that $\ell \cdot (e-e') \equiv 1 \mod |\mathbb{G}|$. Since $e \neq e'$, such a multiplicative inverse is guaranteed to exist because $|\mathbb{G}|$ is prime and every nonzero number has a multiplicative inverse modulo any prime, and in fact ℓ can be computed efficiently via the Extended Euclidean algorithm.

Let $w^* = \left((z-z') \cdot (e-e')^{-1}\right) \mod |\mathbb{G}|$. To see that w^* is a witness, observe that since (a, e, z) and (a, e', z') are both accepting transcripts, it holds that $a \cdot h^e = g^z$ and $a \cdot h^{e'} = g^{z'}$. Since $\mathbb{G}$ is cyclic and g is a generator of $\mathbb{G}$, both a and h are powers of g, say, $a = g^j$ and $h = g^w$ for integers j, w. Then the preceding two equations imply that

$$g^{j+we} = g^z$$
$$g^{j+we'} = g^{z'}.$$

Together, these two equations imply that

$$g^{w(e-e')} = g^{z-z'}.$$

Hence, $w(e-e') \equiv z - z' \mod |\mathbb{G}|$, i.e., $w \equiv (z-z') \cdot (e-e')^{-1} \mod |\mathbb{G}| = w^*$. That is, $h^w = h^{w^*}$, meaning that w^* is a witness.

Honest-Verifier Perfect Zero Knowledge. We need to construct a polynomial time simulator that produces a distribution over transcripts (a, e, z) identical to the distribution produced by the honest verifier and prover. The simulator selects e uniformly at random from $\{0, \ldots, |\mathbb{G}|-1\}$ and samples z uniformly at random from $\{0, \ldots, |\mathbb{G}|-1\}$. Finally, the simulator sets $a \leftarrow g^z \cdot (h^e)^{-1}$.

The distribution over transcripts produced by the simulator is identical to that produced by the honest verifier interacting with the prescribed prover. In both cases, the distribution produces a random $e \in \{0, \ldots, |\mathbb{G}|-1\}$, and then chooses a pair (a, z) such that a is chosen uniformly random from $\mathbb{G}$ and z from $\{0, \ldots, |\mathbb{G}|-1\}$, subject to the constraint that $a \cdot h^e = g^z$ (the key observation from which this follows is that, for fixed e, for any $a \in \mathbb{G}$ there is exactly one $z \in \{0, \ldots, |\mathbb{G}|-1\}$ satisfying this equality, and vice versa).

Remark 12.2. Schnorr's protocol is only *honest-verifier* zero knowledge (HVZK) because the simulated distribution over transcripts is identical

to the verifier's view in the actual protocol only if the verifier's message e is a uniformly random element from $\{0, \ldots, |\mathbb{G}| - 1\}$. Two remarks are in order. First, if we render the protocol non-interactive using the Fiat-Shamir transformation (see Section 12.2.3), the distinction between honest-verifier and dishonest-verifier zero-knowledge is eliminated (see Footnote 4 for a brief discussion of this point). Second, it turns out that Schnorr's protocol actually *is* dishonest-verifier zero-knowledge, with the following caveat: the simulation is efficient only if the challenge e is not selected at random from $\{0, \ldots, |\mathbb{G}| - 1\}$, but rather is only permitted to be selected from a designated polynomial-size subset S of $\mathbb{G}$ (this is because the known simulator for an arbitrary dishonest verifier's view has a runtime that grows with $|S|$). To obtain negligible soundness error from such a protocol, one must repeat it $\omega(1)$ many times sequentially, adding additional communication and computation costs. The interested reader is directed to [189, Section 4] for details.

12.2.3 Fiat-Shamir Applied to Σ-Protocols

In this section, we explain that applying the Fiat-Shamir transformation (Section 5.2) to any Σ-protocol (such as Schnorr's) yields a non-interactive argument of knowledge in the random oracle model. This result is originally due to Pointcheval and Stern [211].

For concreteness, we couch the presentation in the context of Schnorr's protocol, where the input is a group element h, and the prover claims to know a witness w such that $h = g^w$, where g is a specified group generator. Recall that in the resulting non-interactive argument, the honest prover aims to produce an accepting transcript (a, e, z) for the Σ-protocol, where $e = R(h, a)$ and R denotes the random oracle.

Let $\mathcal{I}$ refer to the Σ-protocol and $\mathcal{Q}$ refer to the non-interactive argument obtained by applying the Fiat-Shamir transformation to $\mathcal{I}$. Let $\mathcal{P}_{\mathcal{FS}}$ be a prover for $\mathcal{Q}$ that produces a convincing proof on input h with probability at least ε. That is, when $\mathcal{P}_{\mathcal{FS}}$ is run on input h, it outputs a transcript (a, e, z) that, with probability at least ε, is an accepting transcript for $\mathcal{I}$ and satisfies $e = R(h, a)$ (here, the probability is over the choice of random oracle and any internal randomness used by $\mathcal{P}_{\mathcal{FS}}$).

We show that by running $\mathcal{P}_{\mathcal{FS}}$ at most twice, we can, with probability at least $\Omega(\varepsilon^4/T^3)$, "extract" from $\mathcal{P}_{\mathcal{FS}}$ two accepting transcripts for $\mathcal{I}$ of the form (a, e, z) and (a, e', z') with $e \neq e'$.[10] By special soundness of $\mathcal{I}$, these two transcripts can in turn be efficiently transformed into a witness w. If T is polynomial and ε is non-negligible, then $\Omega(\varepsilon^4/T^3)$ is non-negligible, contradicting the assumed intractability of finding a witness.[11]

What We Can Assume About $\mathcal{P}_{\mathcal{FS}}$ Without Loss of Generality. As in the proof of Theorem 5.1, we will assume that $\mathcal{P}_{\mathcal{FS}}$ always makes exactly T queries to the random oracle R, that all queries $\mathcal{P}_{\mathcal{FS}}$ makes are distinct, and that $\mathcal{P}_{\mathcal{FS}}$ always outputs a transcript of the form (a, e, z) with $e = (h, a)$, such that at least one of $\mathcal{P}_{\mathcal{FS}}$'s T queries to R was at point (h, a). See the proof of Theorem 5.1 for an explanation of why these assumptions are without loss of generality.

The Witness Extraction Procedure. There is a natural way to extract from $\mathcal{P}_{\mathcal{FS}}$ two accepting transcripts (a, e, z) and (a, e', z). First, fix the value of any internal randomness used by $\mathcal{P}_{\mathcal{FS}}$. The first transcript is obtained by simply running $\mathcal{P}_{\mathcal{FS}}$ once, generating a random value for R's response to each query $\mathcal{P}_{\mathcal{FS}}$ makes to the random oracle. This yields a transcript (a, e, z) satisfying $e = R(h, a)$ such that with probability at least ε the transcript is an accepting one for $\mathcal{I}$. By assumption, during this execution of $\mathcal{P}_{\mathcal{FS}}$, exactly one of the T queries to R was equal to (h, a). Rewind $\mathcal{P}_{\mathcal{FS}}$ to just before it queries R at (h, a), and change R's response to this query from e to a fresh randomly chosen value e'. Then run $\mathcal{P}_{\mathcal{FS}}$ once again to completion (again generating a random value from R's response to each query made by $\mathcal{P}_{\mathcal{FS}}$), and hope that $\mathcal{P}_{\mathcal{FS}}$ outputs an accepting transcript of the from (a, e', z').

[10] For simplicity, we do not provide a quantitatively tight analysis of the witness extraction procedure.

[11] One can find a witness with constant probability instead of just with non-negligible probability by running the witness-finding procedure $\ell = O(T^3/\varepsilon^4)$ times. The probability that all ℓ invocations of the procedure fail to find a witness is at most $(1 - 1/\ell)^\ell \leq 1/e < 1/2$.

Analysis of the Witness Extraction Procedure. We must show that the probability this procedure outputs two accepting transcripts of the form (a, e, z) and (a, e', z') with $e \neq e'$ is at least $\Omega(\varepsilon^3/T^2)$. Note that e will not equal e' with probability $1 - 1/2^\lambda$, where λ denotes the number of bits output by R on any query. For simplicity, let us assume henceforth that $e \neq e'$, as this will affect the success probability of the extraction procedure by at most $1/2^\lambda$.

Key to the analysis is the following basic result in probability theory.

Claim 12.1. Suppose (X, Y) are jointly distributed random variables and let $A(X, Y)$ be any event such that $\Pr[A(X, Y)] \geq \varepsilon$. Let μ_X be the marginal distribution of X, and for x in the support of μ_X, call x *good* if the conditional probability $\Pr[A(X, Y)|X = x]$ is at least $\varepsilon/2$. Let $p = \Pr_{x \sim \mu_X}[x \text{ is good}]$ denote the probability that an x drawn at random from the distribution μ_X is good. Then $p \geq \varepsilon/2$.

Proof. If x is not good, let us call x bad. We can write:

$$\begin{aligned}\Pr[A(X,Y)] = &\Pr[A(X,Y)|X \text{ is good}] \cdot \Pr[X \text{ is good}] \\ &+ \Pr[A(X,Y)|X \text{ is bad}] \cdot \Pr[X \text{ is bad}] \\ = &\Pr[A(X,Y)|X \text{ is good}] \cdot p + \Pr[A(X,Y)|X \text{ is bad}] \\ &\times (1-p) \leq 1 \cdot p + \varepsilon/2,\end{aligned}$$

where the final inequality holds by the definition of "bad" outcomes x of X. Since $\Pr[A(X, Y)] \geq \varepsilon$, we conclude that $p \geq \varepsilon/2$. □

Say that $\mathcal{P}_{\mathcal{FS}}$ *wins* if the transcript (a, e, z) that $\mathcal{P}_{\mathcal{FS}}$ produces is an accepting one satisfying $e = R(h, a)$. Consider applying Claim 12.1, with X equal to $\mathcal{P}_{\mathcal{FS}}$'s internal randomness, Y equal to the evaluations of the random oracle R, and $A(X, Y)$ equal to the event that $\mathcal{P}_{\mathcal{FS}}$ wins when run with internal randomness X and random oracle Y. Claim 12.1 implies that with probability at least $\varepsilon/2$, $\mathcal{P}_{\mathcal{FS}}$'s internal randomness is "good", which in this context means that when the internal randomness is set to X, the probability over the random oracle R that $\mathcal{P}_{\mathcal{FS}}$ produces an accepting transcript (a, e, z) with $e = R(h, a)$ is at least $\varepsilon/2$. Let E be the event that $\mathcal{P}_{\mathcal{FS}}$'s internal randomness is good. We can write the

probability that the witness extraction procedure succeeds as

$$\begin{aligned}&\Pr[E] \cdot \Pr[\text{witness extraction succeeds}|E] \\ &\quad \geq (\varepsilon/2) \cdot \Pr_R[\text{witness extraction succeeds}|E].\end{aligned}$$

Here, the subscript R indicates that the probability is over the randomness in the random oracle R.

For the remainder of the proof, we bound $\Pr_R[\text{witness extraction succeeds}|E]$. For notational brevity, we will leave the conditioning on E implicit when writing out the probabilities of various events. By conditioning on E, we may henceforth consider $\mathcal{P}_{\mathcal{FS}}$ to be a *deterministic* algorithm (i.e., no internal randomness), that wins with probability at least $\varepsilon/2$ over the random choice of the random oracle R.

Let $Q_1, \ldots, Q_T$ denote the T queries that $\mathcal{P}_{\mathcal{FS}}$ makes to the random oracle (note that these are random variables that depend on R). Next, we claim that there is at least one integer $i^* \in \{1, \ldots, T\}$ such that

$$\Pr_R[\mathcal{P}_{\mathcal{FS}} \text{ wins} \cap Q_{i^*} = (h, a)] \geq \varepsilon/(2T). \tag{12.1}$$

Indeed, if $\Pr_R[\mathcal{P}_{\mathcal{FS}} \text{ wins} \cap Q_i = (h, a)] < \varepsilon/(2T)$ for all $i = 1, \ldots, T$, then since we have assumed that for any transcript (a, e, z) output by $\mathcal{P}_{\mathcal{FS}}$ there is some $i \in \{1, \ldots, T\}$ such that $Q_i = (h, a)$,

$$\Pr_R[\mathcal{P}_{\mathcal{FS}} \text{ wins}] \leq \sum_{i=1}^{T} \Pr_R[\mathcal{P}_{\mathcal{FS}} \text{ wins} \cap Q_i = (h, a)] < T \cdot (\varepsilon/2) = \varepsilon/2,$$

a contradiction.

Let i^* satisfy Equation (12.1). Consider applying Claim 12.1, now with X equal to R's responses to the first $i^* - 1$ queries, and Y equal to R's responses to the remaining $T - i^* + 1$ queries. And now let A be the event that $\mathcal{P}_{\mathcal{FS}}$, when run with random oracle R, produces a winning transcript (a, e, z) with (h, a) equal to $\mathcal{P}_{\mathcal{FS}}$'s (i^*)'th query, namely Q_{i^*}.

For a value of x in the support of X, call x good if $\Pr[A(X, Y)|X = x] \geq \varepsilon/(4T)$. Equation (12.1) asserts that $\Pr[A(X, Y)] \geq \varepsilon/(2T)$. Hence, Claim 12.1 asserts that X is good with probability at least $\varepsilon/(4T)$.

We can think of the process of generating the two transcripts (a, e, z) and (a', e', z') as first selecting X (thereby determining the first i^* queries

$Q_1, \ldots, Q_{i^*}$ made by $\mathcal{P}_{\mathcal{FS}}$), then drawing two independent copies Y' and Y'' of Y. Both (a, e, z) and (a', e', z') are accepting transcripts with $Q_{i^*} = (h, a) = (h, a')$ if (X, Y') and (X, Y'') both satisfy event A. This probability is at least

$$\Pr[X \text{ is good}] \cdot \Pr[A(X, Y')|X \text{ is good}] \cdot \Pr[A(X, Y'')|X \text{ is good}] \geq (\varepsilon/(4T))^3.$$

In conclusion (taking into account that the argument above has conditioned on the event E that the choice of $\mathcal{P}_{\mathcal{FS}}$'s internal randomness is good, an event that happens with probability at least $\varepsilon/2$), we have shown our witness-extraction procedure succeeds with probability at least $\Omega(\varepsilon^4/T^3)$ as claimed.

Remark 12.3. Results lower bounding the success probability of witness extraction procedures related to the one in this section are called *forking lemmas*. The terminology highlights the fact that the witness extraction procedure runs $\mathcal{P}_{\mathcal{FS}}$ twice, once using random oracle responses (X, Y') and once using (X, Y''), where X captures the random oracle's responses to the first i^* queries made by $\mathcal{P}_{\mathcal{FS}}$ and Y' and Y'' capture responses to the remaining queries. One thinks of the random oracle generation process as "forking" into two different paths after the first i^* responses are generated.

Knowledge-Soundness of the Σ-Protocol Itself. We have just seen how to generate a 2-transcript-tree given a convincing prover for any Σ-protocol that has been rendered non-interactive via the Fiat-Shamir transformation. If the Fiat-Shamir transformation has not been applied, generating a 2-transcript-tree for a Σ-protocol is even simpler: run $(\mathcal{P}, \mathcal{V})$ once to generate an accepting transcript for the Σ-protocol, then rewind the Σ-protocol to just after $\mathcal{P}$ sent its first message, and restart the Σ-protocol with a fresh random challenge to generate a new transcript (see Remark 12.1). A similar analysis to the above shows that both generated transcripts will be accepting with probability at least $\Omega(\varepsilon^3)$ where ε is the probability $\mathcal{P}$ passes $\mathcal{V}$'s checks in the Σ-protocol. And with overwhelming probability the two verifier challenges in the two transcripts will be distinct—specifically, with probability at

least $1 - 1/2^{\lambda}$, where 2^{λ} is the size of the set from which the verifier's challenge is chosen. In this event, the two transcripts form a 2-transcript tree. This procedure can be repeated $O(1/\varepsilon^3)$ times to ensure that the probability of successfully generating at least one 2-transcript-tree is at least, say, 9/10.

12.3 A Homomorphic Commitment Scheme

Commitment Schemes. In a commitment scheme, there are two parties, a committer and a verifier. The committer wishes to bind itself to a message without revealing the message to the verifier. That is, once the committer sends a commitment to some message m, it should be unable to "open" to the commitment to any value other than m (this property is called binding). But at the same time the commitment itself should not reveal information about m to the verifier (this is called hiding).

Most properties come in statistical and computational flavors, just like soundness in interactive proofs and arguments. That is, binding can hold statistically, meaning that even computationally unbounded committers are unable to open a commitment to two different messages except with negligible probability of success. Or it can hold only computationally: polynomial-time committers are unable to open commitments to two different messages. Similarly, hiding may be statistical: even computationally unbounded verifiers cannot extract any information about m from the commitment to m. Or it may be computational: polynomial time verifiers are unable to extract information about m from the commitment.

A commitment can be statistically binding and computationally hiding or vice versa, but it cannot be simultaneously statistically hiding and binding. This is because any commitment that statistically binds the committer to a message must by definition reveal the message in a

statistical sense.[12] In this monograph, we will only consider commitment schemes that are computationally binding and perfectly hiding.

Formally, a commitment scheme is specified by three algorithms, KeyGen, Commit, and Verify. KeyGen is a randomized algorithm that generates a commitment key ck and verification key vk that are available to the committer and the verifier respectively (if all keys are public then $\mathsf{ck} = \mathsf{vk}$), while Commit is a randomized algorithm that takes as input the committing key ck and the message m to be committed and outputs the commitment c, as well as possibly extra "opening information" d that the committer may hold onto and only reveal during the verification procedure. Verify takes as input the commitment, the verification key, and a claimed message m' provided by the committer, and any opening information d and decides whether to accept m' as a valid opening of the commitment.

A commitment scheme is *correct* if $\mathsf{Verify}(\mathsf{vk}, \mathsf{Commit}(m, \mathsf{ck}), m)$ accepts with probability 1, for any m (i.e., an honest committer can always successfully open the commitment to the value that was committed). A commitment scheme is perfectly hiding if the distribution of the commitment $\mathsf{Commit}(m, \mathsf{ck})$ is independent of m. Finally, a commitment scheme is computationally binding if for every polynomial time algorithm $\mathcal{Q}$, the probability of winning the game depicted in Protocol 4 is negligible (i.e., inverse-superpolynomial in the security parameter).

Protocol 4 Binding Game for Commitment Schemes

1: $(\mathsf{vk}, \mathsf{ck}) \leftarrow \mathsf{KeyGen}()$
2: $(c, d, m, d', m') \leftarrow \mathcal{Q}(\mathsf{ck})$
   ▷ c should be thought of as a commitment.
   ▷ d and d' should be thought of as opening information, to open c to messages m and m' respectively.
3: $\mathcal{Q}$ wins if $\mathsf{Verify}(\mathsf{vk}, (c, d), m) = \mathsf{Verify}(\mathsf{vk}, (c, d'), m') = 1$ and $m \neq m'$

[12] A computationally unbounded verifier could simulate a computationally unbounded cheating prover's efforts to open the commitment to multiple messages; statistical binding guarantees that these efforts will succeed for only one message except with negligible probability.

A Perfectly Hiding Commitment Scheme from any Σ-Protocol. Informally, a relation $\mathcal{R}$ is said to be hard if there is no efficient algorithm for identifying a witness w such that $(h, w) \in \mathcal{R}$. More precisely, a *hard relation* is one for which there is some efficient randomized algorithm Gen that generates "hard instances" of the relation in the following sense. Gen outputs pairs (h, w), and there is no polynomial time algorithm that, when fed the value h output by Gen, can find a witness w' such that $(h, w') \in \mathcal{R}$ except with negligible probability. For example, for the discrete logarithm relation in prime order groups $\mathbb{G}$ with generator g for which the discrete logarithm problem is believed to be intractable, Gen would pick a random integer $r \in \{0, \ldots, |\mathbb{G}| - 1\}$ and output (h, r) where $h = g^r$.

Damgård [105] showed how to use any Σ-protocol for any hard relation to obtain a perfectly hiding, computationally binding commitment scheme. By instantiating Damgård's construction with Schnorr's Σ-protocol [223] for the discrete logarithm relation, one recovers a well-known commitment scheme due to Pedersen [206] that will play an important role in this monograph. (The typical presentation of Pedersen's commitment scheme differs slightly, in an entirely cosmetic manner, from the version recovered here. See Protocols 5 and 6 for details.)

Actually, to ensure hiding, Damgård's transformation does require the Σ-protocol to satisfy one property that was not mentioned above. The simulator used to establish HVZK must be able to take as input not only the public input h, but also a challenge e^*, and output a transcript (a, e^*, z) such that the distribution over transcripts produced by the simulator is identical to the distribution over transcripts produced by the interaction of the verifier and prescribed prover when the verifier's challenge is fixed to e^*. This property is called *special* honest-verifier perfect zero-knowledge. The simulator for Schnorr's Σ-protocol satisfies this property simply by fixing the challenge chosen by the simulator to e^*, rather than having the simulator choose the challenge at random from the challenge space.

Here is how Damgård's commitment scheme works. The key generation procedure runs the generation algorithm for the hard relation $\mathcal{R}$ to obtain an (instance, witness) pair $(h, w) \leftarrow \mathsf{Gen}$, and declares h to be both the committing key ck and the verification key vk. Note that

the witness w represents "toxic waste" that must be discarded, in the sense that anyone who knows w may be able to break binding of the commitment scheme. To commit to a message m, the committer runs the simulator from the Σ-protocol for $\mathcal{R}$ (whose existence is guaranteed by the special HVZK property of the Σ-protocol) on public input h to generate a transcript in which the challenge is the message m (this is where the property of the simulator described in the previous paragraph is exploited). Let (a, e, z) be the output of the simulator. The committer sends a as the commitment, and keeps $e = m$ and z as opening information. In the verification stage for the commitment scheme, the committer sends the opening information $e = m$ and z to the verifier, who uses the verification procedure of the Σ-protocol to confirm that (a, e, z) is an accepting transcript for public input h.[13]

We need to show that the commitment scheme satisfies correctness, computational binding, and perfect hiding. Correctness is immediate from the fact that the HVZK property of the Σ-protocol guarantees that the simulator only outputs accepting transcripts. Perfect hiding follows from the fact that in any Σ-protocol, the first message a sent by the prover is independent of the verifier's challenge in the Σ-protocol (which equals the message being committed to in the commitment scheme). Computational binding follows from special soundness of the Σ-protocol: if the committer could output a commitment a and two sets of "opening information" (e, z) and (e', z') that both cause the commitment verifier to accept, then (a, e, z) and (a, e', z') must be accepting transcripts for the Σ-protocol, and there is an efficient procedure to take two such transcripts and produce a witness w such that $(h, w) \in \mathcal{R}$. The fact that $\mathcal{R}$ is hard means that this can only be done with non-negligible probability if the committer runs in superpolynomial time.

Note that when applying the transformation to Schnorr's protocol for the discrete logarithm relation, the key generation procedure produces a random power of generator g, which is simply a random group element

[13] If the committed message m contains data that the verifier couldn't compute on its own, then revealing m to the verifier violates zero-knowledge. In our actual zero-knowledge arguments that make use of Pedersen commitments, the prover will never actually open any commitment, but rather will prove in zero-knowledge that it *could* open the commitment if it wanted to. See Protocol 7.

h. Hence, the commitment key and verification key in the resulting commitment scheme can be generated *transparently* (meaning no toxic waste produced). That is, rather than choosing a witness r at random and letting $h = g^r$, thereby producing toxic waste r that could be used to break binding of the commitment scheme, h can be directly chosen to be a random group element. In this way, no one knows the discrete logarithm of h to base g (and by assumption, computing this discrete logarithm given h and g is intractable).

The resulting commitment scheme is displayed in Protocol 6. The traditional (and equivalent, up to cosmetic differences) presentation of Pedersen commitments is given in Protocol 5 for comparison. To maintain consistency with the literature, for the remainder of this monograph we follow the traditional presentation of Pedersen commitments (Protocol 5). In the traditional presentation, to commit to a message m, the committer picks a random exponent z in $\{0, \ldots, |\mathbb{G}| - 1\}$ and the commitment is $g^m \cdot h^z$. One thinks of h^z as a random group element that operates as a "blinding factor": by multiplying g^m by h^z, one ensures that the commitment is a random group element, statistically independent of m.[14]

Protocol 5 Standard presentation of Pedersen commitments in a cyclic group $\mathbb{G}$ for which the Discrete Logarithm problem is intractable.

1: Let $\mathbb{G}$ be a (multiplicative) cyclic group of prime order. The key generation procedure publishes randomly chosen generators $g, h \in \mathbb{G}$, which serve as both the commitment key and verification key.
2: To commit to a number $m \in \{0, \ldots, |\mathbb{G}| - 1\}$, committer picks a random $z \in \{0, \ldots, |\mathbb{G}| - 1\}$ and sends $c \leftarrow g^m \cdot h^z$.
3: To open a commitment c, committer sends (m, z). Verifier checks that $c = g^m \cdot h^z$.

[14] The blinding factor h^z ensures that the Pedersen commitment is *perfectly* (i.e., statistically) hiding. Even if the blinding factor is omitted, the commitment may not reveal m to a polynomial-time receiver. This is because computing m from the "unblinded" commitment g^m requires solving the discrete logarithm problem to base g. If m is itself uniformly distributed, the binding analysis already assumes this is intractable. Intuitively, m is "hidden in the exponent" of g.

Protocol 6 Commitment scheme obtained from Schnorr's protocol via Damgård's transformation. This is the same as Protocol 5 except for the cosmetic difference that the commitment is taken to be $h^{-m} \cdot g^z$ instead of $g^m \cdot h^z$, with the verification procedure modified accordingly (i.e., the roles of g and h are reversed, and m is replaced with $-m$).

1: Let $\mathbb{G}$ be a (multiplicative) cyclic group of prime order.
2: The key generation procedure publishes randomly chosen generators $g, h \in \mathbb{G}$, which serve as both the commitment key and verification key.
3: To commit to a number $m \in \{0, \ldots, |\mathbb{G}| - 1\}$, committer picks a random $z \in \{0, \ldots, |\mathbb{G}| - 1\}$ and sends

$$c \leftarrow h^{-m} \cdot g^z.$$

4: To open a commitment c, committer sends (m, z). Verifier checks that $c \cdot h^m = g^z$.

12.3.1 Important Properties of Pedersen Commitments

Additive Homorphism. One important property of Pedersen commitments is that they are *additively homomorphic*. This means that the verifier can take two commitments c_1 and c_2, to values $m_1, m_2 \in \{0, \ldots, |\mathbb{G}| - 1\}$ (with m_1, m_2 known to the committer but not to the verifier), and the verifier on its own can derive a commitment c_3 to $m_3 := m_1 + m_2$, such that the prover is able to open c_3 to m_3. This is done by simply letting $c_3 \leftarrow c_1 \cdot c_2$. As for "opening information" provided by the prover, if $c_1 = h^{m_1} \cdot g^{z_1}$ and $c_2 = h^{m_2} \cdot g^{z_2}$, then $c_3 = h^{m_1+m_2} \cdot g^{z_1+z_2}$, so the opening information for c_3 is simply $(m_1 + m_2, z_1 + z_2)$. In summary, Pedersen commitments over a multiplicative group $\mathbb{G}$ are additively homomorphic, with addition of messages corresponds to group-multiplication of commitments.

Perfect HVZK Proof of Knowledge of Opening. We will see that in the design of general-purpose zero-knowledge arguments, it will occasionally be useful for the prover to prove that it knows how to open a commitment c to some value, *without* actually opening the

commitment. As observed by Schnorr, Pedersen commitments have this property, using similar techniques to his Σ-protocol for the Discrete Logarithm relation. See Protocol 7.

The idea is that, for $\mathcal{P}$ to prove it knows m, z such that $c = g^m h^z$, in the first round of the proof, the prover sends a group element $a \leftarrow g^d \cdot h^r$ for a random pair of exponents d, r. One should think of a as $\mathsf{Com}(d, r)$, i.e., a commitment to d using randomness r. Then the verifier sends a random challenge e, and the verifier on its own can derive a commitment to $me + d$ via additive homomorphism, and the prover can derive an opening for this commitment. Specifically, $g^{me+d} \cdot h^{ze+r}$ commits to $me + d$, using randomness $ze + r$. Finally, the prover responds with opening information $(me + d, ze + r)$ for this derived commitment. An equivalent description of the protocol using this perspective is given in Protocol 8.

The idea for why the protocol is zero-knowledge is that since the verifier never learns d or r, the quantities $me + d$ and $ze + r$ that the prover sends to the verifier simply appear to be random elements modulo $|\mathbb{G}|$ from the verifier's perspective. The intuition for why this is special sound is that since the committer does not know e before choosing d, there is no way for the prover to open the commitment to $me + d$ unless it knows how to open the commitment to m. In more detail, if the input commitment is $\mathsf{Com}(m, z) = g^m h^z$, and $\mathcal{P}$'s first message in the protocol is $a = g^d h^r$, then if $\mathcal{P}$ can produce two accepting transcripts $(a, e, (m', z'))$ and $(a, e', (m'', z''))$ with $e \neq e'$, $\mathcal{V}$'s acceptance criterion roughly implies that $m' = m \cdot e + d$ and $z' = z \cdot e + r$ while $m'' = m \cdot e' + d$ and $z'' = z \cdot e' + r$. These are four linearly independent equations in four unknowns, namely m, z, d, and r. Hence, one can take these two transcripts and efficiently solve for both m and z, as $m = (m' - m'')/(e - e')$ and $z = (z' - z'')/(e - e')$.

These intuitions are made formal below.

Perfect Completeness. If prover follows the prescribed protocol in Protocol 7 then

$$g^{m'} \cdot h^{z'} = g^{me+r} \cdot h^{ze+r} = c^e \cdot a.$$

Protocol 7 Zero-Knowledge Proof of Knowledge of Opening of Pedersen Commitment

1: Let $\mathbb{G}$ be a (multiplicative) cyclic group of prime order over which the Discrete Logarithm relation is hard, with randomly chosen generators g and h.
2: Input is $c = g^m \cdot h^z$. Prover knows m and z, Verifier only knows c, g, h.
3: Prover picks $d, r \in \{0, \ldots, |\mathbb{G}| - 1\}$ and sends to verifier $a \leftarrow g^d \cdot h^r$.
4: Verifier sends challenge e chosen at random from $\{0, \ldots, |\mathbb{G}| - 1\}$.
5: Prover sends $m' \leftarrow me + d$ and $z' \leftarrow ze + r$.
6: Verifier checks that $g^{m'} \cdot h^{z'} = c^e \cdot a$.

Protocol 8 Equivalent Exposition of Protocol 7 in terms of commitments and additive homomorphism.

1: Let $\mathbb{G}$ be a (multiplicative) cyclic group of prime order over which the Discrete Logarithm relation is hard, with randomly chosen generators g and h.
2: Let $\mathsf{Com}(m, z)$ denote the Pedersen commitment $g^m \cdot h^z$. Prover knows m and z, Verifier only knows $\mathsf{Com}(m, z), g, h$.
3: Prover picks $d, r \in \{0, \ldots, |\mathbb{G}| - 1\}$ and sends to verifier $a \leftarrow \mathsf{Com}(d, r)$.
4: Verifier sends challenge e chosen at random from $\{0, \ldots, |\mathbb{G}| - 1\}$.
5: Let $m' \leftarrow me + d$ and $z' \leftarrow ze + r$, and let $c' \leftarrow \mathsf{Com}(m', z')$. While Verifier does not know m' and z', Verifier can derive c' unaided from $\mathsf{Com}(m, z)$ and $\mathsf{Com}(d, r)$ using additive homomorphism.
6: Prover sends (m', z').
7: Verifier checks that m', z' is valid opening information for c', i.e., that $g^{m'} \cdot h^{z'} = c'$.

Special Soundness. Given two accepting transcripts $(a, e, (m'_1, z'_1))$ and $(a, e', (m'_2, z'_2))$ with $e \neq e'$, we have to extract a valid opening (m, z) for the commitment c, i.e., $g^m \cdot h^z = c$. As in the analysis of the Σ-protocol for the Discrete Logarithm relation, let $(e - e')^{-1}$ denote the multiplicative inverse of $e - e'$ modulo $|\mathbb{G}|$, and define

$$m^* = (m'_1 - m'_2) \cdot (e - e')^{-1} \mod |\mathbb{G}|,$$

$$z^* = (z'_1 - z'_2) \cdot (e - e')^{-1} \mod |\mathbb{G}|.$$

Then

$$\begin{aligned} g^{m^*} \cdot h^{z^*} &= \left(g^{(m'_1 - m'_2)} h^{(z'_1 - z'_2)}\right)^{(e-e')^{-1}} \\ &= \left(c^e \cdot a \cdot \left(c^{e'} \cdot a\right)^{-1}\right)^{(e-e')^{-1}} = c, \end{aligned}$$

where the penultimate equality follows from the fact that $(a, e, (m'_1, z'_1))$ and $(a, e', (m'_2, z'_2))$ are accepting transcripts. That is, (m^*, z^*) is a valid (message, opening information) pair for the commitment c.

Perfect HVZK. The simulator samples e, m', z' uniformly at random from $\{0, \ldots, |\mathbb{G}| - 1\}$ and then sets

$$a \leftarrow g^{m'} \cdot h^{e'} \cdot c^{-e},$$

and outputs

$$(a, e, (m', z')).$$

This ensures that e is uniformly distributed, and a, and (m', z') are also uniformly distributed over $\mathbb{G}$ and $\{0, \ldots, |\mathbb{G}| - 1\}^2$ under the constraint that $g^{m'} \cdot h^{e'} = c^e \cdot a$. This is the same distribution as that generated by the honest verifier interacting with the prescribed prover.

Perfect HVZK Proof of Knowledge of Opening to A Specific Value. The above protocol allows the prover to establish it knows how to open a Pedersen commitment c to *some* value. A variant we will also find useful allows the prover to establish in zero-knowledge that it knows how to open c to a specific public value y. Since a Pedersen commitment c to public value y is of the form $g^y h^r$ for a random $r \in \mathbb{G}$, proving that knowledge of how to open c to y is equivalent to proving knowledge of a value r such that $h^r = c \cdot g^{-y}$. This amounts to proving knowledge of the discrete logarithm of $c \cdot g^{-y}$ in base h, which can be done using Protocol 3.

A Final Perspective on Protocol 7. Protocol 7 asks the prover *not* to open c itself (which would violate zero-knowledge), but instead to open a different commitment c', to random group element that is derived

homomorphically from both c and a commitment to random value d that the prover sends via its first message. Both the prover and verifier "contribute randomness" to the value $m' = me + d$ committed by c'. The randomness contributed by the prover (namely d) is used to ensure that m' is statistically independent of m, which ensures that the opening m' for c' reveals no information about m. The verifier's contribution e to m' is used to ensure special soundness: the prover cannot open c' for more than one value of the verifier's challenge e unless the prover knows how to open c.

We will see more twists on this paradigm (in Sections 12.3.2 and 14.2), in contexts where the prover wants to establish in zero-knowledge that various committed values satisfy certain relationships. Directly opening the commitments would enable to verifier to easily check the claimed relationship, but violate zero-knowledge. So instead the prover opens *derived* commitments, with both the prover and verifier contributing randomness to the derived commitments in a manner such that the derived commitments satisfy the same property that the prover claims is satisfied by the original commitments.

12.3.2 Establishing A Product Relationship Between Committed Values

We have already seen the Pedersen commitments are additively homomorphic, meaning the verifier can take two commitments c_1 and c_2 to values m_1 and m_2 in $\{0, \ldots, |\mathbb{G}| - 1\}$, and without any help from the committer, the verifier can derive a commitment to $m_1 + m_2$ (despite the fact that the verifier has no idea what m_1 and m_2 are, owing to the hiding property of the commitments).

Unfortunately, Pedersen commitments are not *multiplicatively* homomorphic: there is no way for the verifier to derive a commitment to $m_1 \cdot m_2$ without help from the committer. But suppose the committer *sends* a commitment c_3 that is claimed to be a commitment to value $m_1 \cdot m_2$ (meaning that the prover knows how to open up c_3 to the value $m_1 \cdot m_2$). Is it possible for the prover to prove to the verifier that c_3 indeed commits to $m_1 \cdot m_2$, without actually opening up c_3 and thereby

revealing $m_1 \cdot m_2$? The answer is yes, using a somewhat more complicated variant of the Σ-protocols we have already seen. The Σ-protocol is depicted in Protocol 9, with an equivalent formulation in terms of commitments and additive homomorphism given in Protocol 10.

The rough idea of the protocol is that if m_3 indeed equals $m_1 \cdot m_2$, then c_3 can be thought of not only as a Pedersen commitment to $m_1 \cdot m_2$ using group generators g and h, i.e., $c_3 = \mathsf{Com}_{g,h}(m_1 \cdot m_2, r_3)$, but also as a Pedersen commitment to m_2 using group generators $c_1 = g^{m_1} h^{r_1}$ and h. That is, if $m_3 = m_1 \cdot m_2$, it can be checked that

$$c_3 = \mathsf{Com}_{c_1,h}(m_2, r_3 - r_1 m_2).$$

Equivalently, c_3 is a commitment to the same message m_2 as c_2, just using a different generator (c_1 in place of g) and a different blinding factor ($r_3 - r_1 m_2$ in place of r_2). The protocol roughly enables the prover to establish in zero-knowledge that it knows how to open c_3 as a commitment of this form.

Similar to Protocol 7, the idea is to have the prover send commitments to random values b_1 and b_3, the latter being committed twice, once using generators (g, h) and once using generators (c_1, h). The verifier then derives commitments to $em_1 + b_1$ and $em_2 + b_3$ using additive homomorphism (with two commitments derived for the latter quantity, one under the pair of generators (g, h) and the other under (c_1, h)), and then the prover opens these derived commitments. Roughly speaking, the protocol is zero-knowledge since the random choice of b_1 and b_3 ensures that the revealed opening is a random group element independent of m_1 and m_2.

In more detail, the prover first sends three values α, β, γ, where $\alpha = \mathsf{Com}_{g,h}(b_1, b_2)$ and $\beta = \mathsf{Com}_{g,h}(b_3, b_4)$ are commitments to random values $b_1, b_3 \in \{0, \ldots, |\mathbb{G}| - 1\}$ using random blinding factors $b_2, b_4 \in \{0, \ldots, |\mathbb{G}| - 1\}$. Here, the group generators used to produce the two commitments are g and h. γ on the other hand is *another* commitment to b_3 (just as β is), but using group generators c_1 and h rather than g and h. That is, γ is set to $\mathsf{Com}_{c_1,h}(b_3, b_5)$ for a randomly chosen b_5.

From these three values, and despite not knowing m_1, m_2, r_1, r_2, r_3, or $b_1, \ldots, b_5$, the verifier can, for any value $e \in \mathbb{G}$, use additive homomorphism to derive commitments $c'_1 = \mathsf{Com}_{g,h}(b_1 + em_1, b_2 +$

$er_1)$, $c'_2 = \mathsf{Com}_{g,h}(b_3 + em_2, b_4 + er_2)$, and $c'_3 = \mathsf{Com}_{c_1,h}(b_3 + em_2, b_5 + e(r_3 - r_1 m_2))$. After the verifier sends a random challenge e, the prover responds with five values $z_1, \ldots, z_5$ such that (z_1, z_2), (z_3, z_4) and (z_3, z_5) are opening information for c'_1, c'_2 and c'_3 respectively.

Completeness, Special Soundness, and Honest-Verifier Zero-Knowledge. Completeness holds by design. For brevity, we merely sketch the intuition for why special soundness and zero-knowledge hold (though the formal proofs are not difficult and can be found in [189] or [244, Appendix A]).

The intuition for why the protocol is honest-verifier zero-knowledge is that the blinding factors b_2, b_4, b_5 ensure that the prover's first message (α, β, γ) leaks no information about the random committed values b_1, b_3, and this in turn ensures that the prover's second message $(z_1, \ldots, z_5)$ reveal no information about m_1 and m_2.

The intuition for why the protocol is special-sound is that if (a, e, z) and (a, e', z') are two accepting transcripts, where $a = (\alpha, \beta, \gamma)$, $z = (z_1, \ldots, z_5)$, and $z' = (z'_1, \ldots, z'_5)$ then the verifier's checks roughly ensure that:

- $b_1 + em_1 = z_1$ and $b_1 + e'm_1 = z'_1$.
- $b_2 + er_1 = z_2$ and $b_2 + e'r_1 = z'_2$.
- $b_3 + em_2 = z_3$ and $b_3 + e'm_2 = z'_3$.
- $b_4 + er_2 = z_4$ and $b_4 + e'r_2 = z'_4$.
- $b_5 + e(r_3 - r_1 m_2) = z_5$ and $b_5 + e'(r_3 - r_1 m_2) = z'_5$

The first two bullet points refer to the fact that (z_1, z_2) opens $\mathsf{Com}_{g,h}(b_1 + em_1, b_2 + er_1)$ and (z'_1, z'_2) open $\mathsf{Com}_{g,h}(b_1 + e'm_1, b_2 + e'r_1)$. As in the special soundness analysis of Protocol 7, if $e \neq e'$ then the first bullet point represents two linearly independent equations in the unknown m_1 and hence enables solving for m_1 as $(z_1 - z'_1) \cdot (e - e')^{-1}$. Similarly, the second bullet point enables solving for r_1 as $(z_2 - z'_2) \cdot (e - e')^{-1}$. Formally, one can show that $((z_1 - z'_1) \cdot (e - e')^{-1}, (z_2 - z'_2) \cdot (e - e')^{-1})$ is a valid opening for c_1 using generators g and h.

The next two bullet points refer to the fact that (z_3, z_4) opens $\mathsf{Com}_{g,h}(b_3 + em_2, b_4 + er_2)$ and (z'_3, z'_4) open $\mathsf{Com}_{g,h}(b_3 + e'm_2, b_4 + e'r_2)$, and enable solving for m_2 and r_2 as $(z_3 - z'_3) \cdot (e - e')^{-1}$ and $(z_4 - z'_4) \cdot (e - e')^{-1}$. Formally, one can show that $((z_3 - z'_3) \cdot (e - e')^{-1}, (z_4 - z'_4) \cdot (e - e')^{-1})$ is a valid opening for c_2 using generators g and h.

The final bullet point refers to the fact that (z_3, z_5) opens $\mathsf{Com}_{c_1,h}(b_3 + em_2, b_5 + e(r_3 - r_1 m_2))$ and (z'_3, z'_5) opens $\mathsf{Com}_{c_1,h}(b_3 + e'm_2, b_5 + e'(r_3 - r_1 m_2))$. Since r_1 and m_2 have already been derived from the preceding bullet points, the two equations in the final bullet point enable solving for r_3 as $(z_5 - z'_5) \cdot (e - e')^{-1} + r_1 m_2$. Formally, one can show that $(m_1 \cdot m_2, (z_5 - z'_5) \cdot (e - e')^{-1} + r_1 m_2)$ is a valid opening for c_3 using generators g and h.

Protocol 9 Zero-Knowledge PoK of Opening of Pedersen Commitments Satisfying Product Relationship

1: Let $\mathbb{G}$ be a (multiplicative) cyclic group of prime order over which the Discrete Logarithm relation is hard.
2: Input is $c_i = g^{m_i} \cdot h^{r_i}$ for $i \in \{1, 2, 3\}$ such that $m_3 = m_1 \cdot m_2 \bmod |\mathbb{G}|$.
3: Prover knows m_i and r_i for all $i \in \{1, 2, 3\}$, Verifier only knows c_1, c_2, c_3, g, h.
4: Prover picks $b_1, \ldots, b_5 \in \{0, \ldots, |\mathbb{G}| - 1\}$ and sends to verifier three values:
$$\alpha \leftarrow g^{b_1} \cdot h^{b_2},\ \beta \leftarrow g^{b_3} \cdot h^{b_4},\ \gamma \leftarrow c_1^{b_3} \cdot h^{b_5}.$$
5: Verifier sends challenge e chosen at random from $\{0, \ldots, |\mathbb{G}| - 1\}$.
6: Prover sends $z_1 \leftarrow b_1 + e \cdot m_1$, $z_2 \leftarrow b_2 + e \cdot r_1$, $z_3 \leftarrow b_3 + e \cdot m_2$, $z_4 \leftarrow b_4 + e \cdot r_2$, $z_5 \leftarrow b_5 + e \cdot (r_3 - r_1 m_2)$.
7: Verifier checks that the following three equalities hold:
$$g^{z_1} \cdot h^{z_2} = \alpha \cdot c_1^e,$$
$$g^{z_3} \cdot h^{z_4} = \beta \cdot c_2^e,$$
and
$$c_1^{z_3} \cdot h^{z_5} = \gamma \cdot c_3^e.$$

Protocol 10 Equivalent description of Protocol 9 in terms of commitments and additive homomorphism. The notation $\mathsf{Com}_{g,h}(m,z) \coloneqq g^m h^z$ indicates that the group generators used to produce the Pedersen commitment to m with blinding factor z are g and h.

1: Let $\mathbb{G}$ be a (multiplicative) cyclic group of prime order over which the Discrete Logarithm relation is hard.
2: Input is $c_i = g^{m_i} \cdot h^{r_i} = \mathsf{Com}_{g,h}(m_i, r_i)$ for $i \in \{1,2,3\}$ such that $m_3 = m_1 \cdot m_2 \mod |\mathbb{G}|$.
3: Prover knows m_i and r_i for all $i \in \{1,2,3\}$, Verifier only knows c_1, c_2, c_3, g, h.
4: Prover picks $b_1, \ldots, b_5 \in \{0, \ldots, |\mathbb{G}| - 1\}$ and sends to verifier three values:

$$\alpha \leftarrow \mathsf{Com}_{g,h}(b_1, b_2),\ \beta \leftarrow \mathsf{Com}_{g,h}(b_3, b_4),\ \gamma \leftarrow \mathsf{Com}_{c_1,h}(b_3, b_5).$$

5: Verifier sends challenge e chosen at random from $\{0, \ldots, |\mathbb{G}| - 1\}$.
6: Let $z_1 \leftarrow b_1 + e \cdot m_1$, $z_2 \leftarrow b_2 + e \cdot r_1$, $z_3 \leftarrow b_3 + e \cdot m_2$, $z_4 \leftarrow b_4 + e \cdot r_2$, $z_5 \leftarrow b_5 + e \cdot (r_3 - r_1 m_2)$.
7: While Verifier does not know $z_1, \ldots, z_5$, using additive homomorphism Verifier can derive the following three commitments unaided using additive homomorphism:

$$\begin{aligned} c_1' &= \mathsf{Com}_{g,h}(z_1, z_2) = \alpha \cdot c_1^e, \\ c_2' &= \mathsf{Com}_{g,h}(z_3, z_4) = \beta \cdot c_2^e, \\ c_3' &= \mathsf{Com}_{c_1,h}(z_3, z_5) = \gamma \cdot c_3^e. \end{aligned}$$

This final equality for c_3' exploits that

$$c_3^e = g^{em_1m_2} h^{er_3} = c_1^{em_2} h^{er_3 - er_1m_2} = \mathsf{Com}_{c_1,h}(em_2, er_3 - er_1m_2).$$

8: Prover sends $z_1, \ldots, z_5$.
9: Verifier checks that:

- (z_1, z_2) is valid opening information for c_1' using generators g, h.
- (z_3, z_4) is valid opening information for c_2' using generators g, h.
- (z_3, z_5) is valid opening information for c_3' using generators c_1, h.