13

Zero-Knowledge via Commit-And-Prove and Masking Polynomials

Historically, the first zero-knowledge argument for an **NP**-complete problem was given by Goldreich, Micali, and Wigderson (GMW) [131]. GMW designed a zero-knowledge argument with a polynomial-time verifier for the Graph 3-Coloring problem. This yields a zero-knowledge argument with a polynomial time verifier for any language $\mathcal{L}$ in **NP** (including arithmetic circuit satisfiability), because any instance of $\mathcal{L}$ can first be transformed into an equivalent instance of Graph 3-Coloring with a polynomial blowup in instance size, and then GMW's zero-knowledge argument for Graph 3-Coloring can be applied. However, this does not yield a practical protocol for two reasons. First, GMW's construction works by first designing a "basic" protocol that has large soundness error $(1 - 1/|E|$, where $|E|$ denotes the number of edges in the graph) and hence needs to be repeated a polynomial number of times to ensure negligible soundness error. Second, for problems in **NP** that are relevant in practice, reductions to Graph 3-Coloring can introduce large (polynomial) overheads. That is, we saw in Section 6 that arbitrary non-deterministic RAMs running in time T can be transformed into equivalent circuit satisfiability instances of size $\tilde{O}(T)$, but an analogous result is not known for Graph 3-Coloring. For this

reason, our focus in this monograph is on directly giving zero-knowledge arguments for circuit satisfiability and related problems, rather than for other **NP**-complete problems. The interested reader can learn more about GMW's seminal zero-knowledge argument from any standard text on zero-knowledge (e.g., [130, Section 4.4.2]).

Commit-and-Prove Zero-Knowledge Arguments. In this section, we describe our first zero-knowledge arguments for circuit satisfiability. These are based on a technique often called *commit-and-prove*.[1] The idea is as follows. Suppose that for some agreed-upon circuit $\mathcal{C}$, the prover wants to establish that it knows a witness w such that $\mathcal{C}(w) = 1$,[2] and consider the following naive, information-theoretically secure and non-interactive proof system, which is (perfectly) sound but not zero-knowledge. The prover sends w to the verifier, along with the value of every gate of $\mathcal{C}$ when $\mathcal{C}$ is evaluated on input w. The verifier simply checks that the claimed value of the output gate is 1, and checks gate-by-gate that the claimed value of the gate is accurate (i.e., for any multiplication (respectively, addition) gate, the value the prover sends for that gate is indeed the product (respectively, sum) of the two in-neighbors of the gate). Clearly, this proof system is information-theoretically sound, but is not zero-knowledge because the verifier learns the witness w.

To obtain a zero-knowledge argument, the prover will instead send a hiding commitment to each gate, and prove in zero-knowledge that the

[1] An important warning: some papers use the phrase "commit-and-prove SNARKs", e.g., [87], which is related to but different than our use of the term commit-and-prove in this survey. Commit-and-prove SNARKs are SNARKs in which the verifier is given a compressing commitment to an input vector (e.g., using the generalized Pedersen commitment we describe later in Section 14.2), and the SNARK is capable of establishing that the prover knows an opening w for the commitment such that w satisfies a property of interest. Hence, commit-and-prove SNARKs are SNARKs for a particular type of statement. In contrast, we use commit-and-prove to refer to a particular *design approach* for zero-knowledge arguments.

[2] In previous sections of this survey, we have considered arithmetic circuits that take as input a public input x and witness w, and the prover wants to establish knowledge of a w such that $\mathcal{C}(x, w) = 1$. In this section we omit the public input x for brevity. It is easy to modify the arguments given here to support a public input x in addition to a witness w.

committed values satisfy the checks that the verifier in the naive (non-zero-knowledge) proof system performs. This way the argument system verifier learns nothing about the committed values, but nonetheless confirms that the committed values would have satisfied the verifier within the information-theoretically secure protocol.

The next section contains additional details of this approach when the commitment scheme used is Pedersen commitments.

13.1 Proof Length of Witness Size Plus Multiplicative Complexity

Section 12.3.2 explained that Pedersen commitments satisfy the following properties: (a) they are additively homomorphic, meaning that given commitments c_1, c_2 to values m_1, m_2, the verifier can compute a commitment to $m_1 + m_2 \mod |\mathbb{G}|$ directly from c_1, c_2, even though the verifier does not know m_1 or m_2 (b) given commitments c_1, c_2, c_3 to values m_1, m_2, m_3 there is a Σ-protocol (Protocol 9) for which the prover can establish in (honest verifier) zero-knowledge that c_3 is a commitment to $m_1 \cdot m_2 \mod |\mathbb{G}|$.

Addition and multiplication are a universal basis, meaning that with these two operations alone, one can compute *arbitrary* functions of any input. Hence, properties (a) and (b) together effectively mean that a verifier is able to do arbitrary computation over committed values, without making the prover ever reveal the committed values.

In more detail, we have the following zero-knowledge argument for arithmetic circuit satisfiability. While conceptually appealing, this argument is not succinct—the communication complexity is linear in the witness size $|w|$ plus the number M of multiplication gates of the circuit, leading to very large proofs.

Let $\mathcal{C}$ be an arithmetic circuit over $\mathbb{F}$ of prime order, and let $\mathbb{G}$ be a cyclic group of the same order as $\mathbb{F}$ in which the Discrete Logarithm relation is assumed to be hard. Let us suppose that multiplication gates in $\mathcal{C}$ have fan-in 2 (the zero-knowledge argument in this section naturally supports addition gates of unbounded fan-in, in which case we can assume without loss of generality that the in-neighbors of any addition gate consist entirely of multiplication gates). Suppose the prover claims that it knows a w such that $\mathcal{C}(w) = 1$.

At the start of the protocol, the prover sends Pedersen commitments to each entry of w, as well as Pedersen commitments to the value of every multiplication gate in $\mathcal{C}$. Then, for each entry of witness w, the prover proves via Protocol 7 that the prover knows an opening of the commitment to that entry. Next, for each multiplication gate in the circuit, the prover uses Protocol 9 to prove that the committed values respect the operations of the multiplication gates. That is, if a multiplication gate g_1 computes the product of gates g_2 and g_3, the verifier can demand that the prover prove in zero-knowledge that the commitment c_1 to the value of gate g_1 equals the product of the commitments c_2 and c_3 to the value of gates g_2 and g_3. Addition gates are handled within the protocol without any communication between prover and verifier by using the additive homomorphism property of Pedersen commitments: if an addition gate g_1 computes the sum of gates g_2 and g_3, the verifier can on its own, via Property (a), compute a commitment to the value of g_1 given commitments to the values of gates g_2 and g_3. Finally, at the end of the protocol, the prover uses Protocol 3 to prove knowledge of how to open the commitment to the value of the output gate of $\mathcal{C}$ to value $y = 1$.

The resulting proof system is clearly complete because each of the subroutines (Protocols 3, 7, and 9) is complete. To show it is perfect honest-verifier zero-knowledge, one must construct an efficient simulator whose output is distributed identically to the honest verifier's view in the protocol. The idea of the construction is simply that the protocol is comprised entirely of $|w| + M + 1$ sequential invocations of Σ-protocols that are themselves perfect honest-verifier zero-knowledge. The simulator for the entire protocol can simply run the simulator for each of these subroutines in sequence and concatenate the transcripts that it generates.

13.1.1 Establishing Knowledge-Soundness

To establish our argument system is an argument of knowledge for arithmetic circuit satisfiability, we need to show that if the prover convinces the verifier to accept with non-negligible probability, then it is possible to efficiently extract from the prover a witness w such

that $\mathcal{C}(w) = 1$. Formally, we must show that for any prover $\mathcal{P}$ that convinces the argument system verifier to accept with non-negligible probability, there is a polynomial-time algorithm $\mathcal{E}$ that, given access to a rewindable transcript generator for the prover-verifier pair $(\mathcal{P}, \mathcal{V})$ for the argument system, outputs a w such that $\mathcal{C}(w) = 1$.

Naturally, the procedure to extract the witness w relies on the fact that each of the $|w| + M + 1$ subroutines used in the argument system protocol themselves satisfies special soundness. Recall that this means the protocols consist of three messages, and given access to two accepting transcripts that share a first message and differ in their second message, there is an efficient procedure to extract a witness for the statement being proven. We call such a set of transcripts a 2-transcript-tree for the subroutine.

Using its access to a rewindable transcript generator for $(\mathcal{P}, \mathcal{V})$, $\mathcal{E}$ can in polynomial time identify a 2-transcript-tree for each subroutine with high probability.

By special soundness of Protocol 7, given such a set of 2-transcript-trees for all of the subroutines of the argument system, $\mathcal{E}$ can extract an opening of the commitment to each entry i of the witness, and output the vector w of extracted values. We now explain that the vector w that is output in this manner indeed satisfies $\mathcal{C}(w) = 1$.

Just as $\mathcal{E}$ extracted an opening for each entry of w from the 2-transcript-trees for each invocation of Protocol 7, given the 2-transcript-trees for the invocation of Protocol 9 to the ith multiplication gate of $\mathcal{C}$, there is an efficient procedure to extract openings to the commitment for multiplication gate i and the commitments to its two in-neighbor gates such that the values opened respect the multiplication operation (one or both of these in-neighbors may be addition gates, the commitments for which are derived via additive homomorphism from the commitments to multiplication gates sent by the prover). Similarly, given the 2-transcript-tree for the lone invocation of Protocol 3, there is an efficient procedure to extract an opening of the commitment to the output gate value to 1.

Observe that a value for any particular gate g in $\mathcal{C}$ may be extracted multiple times by these extraction procedures. For example, the value of a gate g will be extracted via the 2-transcript-tree for any invocation of Protocol 9 to a gate g' for which g is an in-neighbor. And if g is

itself a multiplication gate, its value will be extracted an additional time from the application of Protocol 9 to g itself. And the output gate of $\mathcal{C}$ will have an opening of its commitment to value 1 extracted due to the invocation of Protocol 3.

For any gate whose value is extracted multiple times, the extracted values must all be the same, for if this were not the case, the extraction procedure would have identified two different openings of the same commitment. This would violate the binding property of the commitment scheme, since all the 2-transcript-trees were constructed in polynomial time and the extraction procedure from each 2-transcript-tree is also efficient.

In summary, we have established the following properties of the extracted values:

- A unique value is extracted for every gate of $\mathcal{C}$ and entry of the witness w.
- The extracted values for all multiplication gates respect the multiplication operation of the gate (this holds by the special soundness of Protocol 9).
- The extracted values of the gates also respect the addition operations computed by all addition gates of $\mathcal{C}$ (this holds by the additive homomorphism of the commitment scheme).
- The extracted value for the output gate is 1.

This four properties together imply that $\mathcal{C}(w) = 1$ where w is the extracted witness.

13.1.2 A Final Perspective on Commit-and-Prove

The commit-and-prove argument described above is conceptually related to *fully homomorphic encryption* (FHE). An FHE scheme allows for computation over encrypted data. Specifically, let c_1 and c_2 by ciphertexts with corresponding plaintexts m_1 and m_2. An FHE scheme allows anyone given c_1 and c_2 (but not the corresponding plaintexts) to compute encryptions of $m_1 \cdot m_2$ and $m_1 + m_2$. This allows any arithmetic circuit to be evaluated gate-by-gate over encrypted data.

For example, a computationally limited user can offload sensitive computation to a cloud computing service by encrypting their data with an FHE scheme, and asking the cloud computing service to evaluate an arithmetic circuit over their encrypted data. The cloud service can proceed gate-by-gate through the circuit. For each addition gate and multiplication gate in the circuit, the service can apply the addition or multiplication operation to the plaintexts "inside" the ciphertexts of the gate's in-neighbors, without ever "opening" the ciphertexts. In this way, the cloud service obtains an encryption of the circuit output, which it can send to the user, who decrypts it. The use of FHE here avoids leaking the user's information to the cloud.[3]

The commit-and-prove zero-knowledge argument is conceptually similar, with the argument *prover* playing the role of user, and the *verifier* playing the role of the cloud service. To preserve zero-knowledge, the prover wishes to keep the elements of the witness hidden from the verifier. So the prover *commits* to the witness elements using an additively homomorphic commitment scheme (Pedersen commitments)– these commitments are analogs of the ciphertexts in the FHE scenario above. The verifier seeks to obtain a commitment to the output of the circuit, analogously to how the cloud server seeks to obtain an encryption of the output. The key difference in the commit-and-prove argument is that the commitment scheme is only additively homomorphic rather than fully homomorphic. This means that the verifier on its own can "add two committed values" without ever opening the commitments, but cannot multiply them. So for every multiplication gate in the circuit, the prover in the commit-and-prove argument *helps* the verifier compute the multiplication, by sending a commitment to the product and proving in zero-knowledge that indeed it can open that commitment to the appropriate product of committed values. This is why the proof length grows linearly with the number of multiplication gates in the circuit, but has no dependence on the number of addition gates.

[3]Note that FHE does not provide a guarantee that the cloud correctly evaluated the designated arithmetic circuit on the user's data. One would need to combine FHE with a proof or argument system to obtain such a guarantee.

13.1.3 Commit-and-Prove with Other Commitment Schemes

We used Pedersen commitments in the commit-and-prove zero-knowledge argument system above. However, the only properties of Pedersen commitments we needed were: perfect hiding, computational binding, additive homomorphism, and zero-knowledge arguments of knowledge for opening information and product relationships. One can replace Pedersen commitments with any other commitment scheme satisfying these properties. To this end, several works [23], [107], [248] essentially replace Pedersen commitments with a commitment scheme derived from a primitive called *vector oblivious linear evaluation* (VOLE) [8]. This has the following benefits over Pedersen commitments. First, using Pedersen commitments to implement commit-and-prove leads to proofs containing 10 elements of a cryptographic group per multiplication gate. The use of VOLE-based commitments can reduce this communication to as low as 1 or 2 field elements per multiplication gate. Second, the computational binding property of Pedersen commitments is based on the intractability of the discrete logarithm problem, and since quantum computers can efficiently compute discrete logarithms, the resulting commit-and-prove arguments are not quantum-sound. In contrast, VOLE-based commitments are believed to be quantum-sound (they are based on variants of the so-called Learning Parity with Noise (LPN) assumption).

However, the use of VOLE-based commitments comes with significant downsides as well. Specifically, these commitments currently require an interactive pre-processing phase Unlike commit-and-prove with Pedersen-based commitments, the interaction cannot be fully removed with the Fiat-Shamir transformation, and accordingly the resulting arguments for circuit satisfiability are not publicly verifiable.

13.2 Avoiding Linear Dependence on Multiplicative Complexity: zk-Arguments from IPs

The proof length in the zero-knowledge argument of the previous section is linear in the witness length and number of multiplication gates of $\mathcal{C}$. Moreover, the verifier's runtime is linear in the size of $\mathcal{C}$ (witness length

plus number of addition and multiplication gates), as the verifier effectively applies every gate operation in $\mathcal{C}$ "underneath the commitments" (i.e., the verifier evaluates every gate on committed values, without ever asking the prover to open any commitments).

It is possible to reduce the communication complexity and verifier runtime to $O(|w| + d \cdot \text{polylog}(|\mathcal{C}|))$, where d is the depth of $|\mathcal{C}|$, by combining the ideas of the previous section with the GKR protocol. The idea is to start with our first, naive protocol for circuit satisfiability, (Section 7.1), which is not zero-knowledge, and combine it with the ideas of the previous section to render it zero-knowledge without substantially increasing any of its costs (communication, prover runtime, or verifier runtime). Specifically, recall that in the naive protocol of Section 7.1 we had the prover explicitly send w to the verifier, and then applied the GKR protocol to prove that $\mathcal{C}(w) = 1$. This was not zero-knowledge because the verifier learns the witness w.

To render it zero-knowledge, we can have the prover send Pedersen commitments to each element of w and use Protocol 7 to prove knowledge of openings of each commitment, exactly as the prover did at the start of the zero-knowledge argument from the previous section. Then we can apply the GKR protocol to the claim that $\mathcal{C}(w) = 1$. However, the prover's messages within the GKR protocol also leak information about the witness w to the verifier, as the prover's messages all consist of low-degree univariate polynomials whose coefficients are derived from $\mathcal{C}$'s gate values when evaluated on w. To address this issue, we do not have the prover send the coefficients of these polynomials to the verifier "in the clear", but rather have the prover $\mathcal{P}$ send Pedersen commitments to these coefficients and engage for each one in an invocation of Protocol 7 to prove that $\mathcal{P}$ knows an opening of the commitment. In sum, when the argument system prover and verifier have finished simulating the GKR protocol, the argument system prover has sent Pedersen commitments to all entries of the witness w and all entries of the GKR prover's messages.

We now have to explain how the argument system verifier can confirm in zero-knowledge that the values inside these commitments would have convinced the GKR verifier to accept the claim that $\mathcal{C}(w) = 1$. The idea is roughly that there is a circuit $\mathcal{C}'$ that takes as input the

prover's messages in the GKR protocol (including the witness w), and such that (1) all of $\mathcal{C}'$ outputs are 1 if and only if the prover's messages would convince the GKR verifier to accept, and (2) $\mathcal{C}'$ contains $O(d \log |\mathcal{C}| + |w|)$ addition and multiplication gates. Hence, we can apply the zero-knowledge argument of the previous section to the claim that $\mathcal{C}'$ outputs the all-1s vector. Recall that at the start of the argument system of the previous section applied to the claim $\mathcal{C}'(w') = 1$, the prover sent commitments to each entry of w'. In this case, w' consists of the witness w for $\mathcal{C}$ and the prover's messages within the GKR protocol, and the argument system has already committed to these values (via the last sentence of the previous paragraph). By Property (2) of $\mathcal{C}'$, the total communication cost and verifier runtime of the zero-knowledge argument applied to $\mathcal{C}'$ is $O(|w| + d \log |\mathcal{C}|)$.

The argument for $\mathcal{C}$ is easily seen to be complete and honest-verifier zero-knowledge (since it consists of the sequential application of honest-verifier zero-knowledge argument systems). To formally prove that it is knowledge sound, one needs to show that, given any argument system prover $\mathcal{P}$ that runs in polynomial time and causes the argument system verifier to accept with non-negligible probability, one can extract a witness w and a prover strategy $\mathcal{P}'$ for the GKR protocol applied to the claim $\mathcal{C}(w) = 1$ that causes the GKR verifier to accept with high probability. Soundness of the GKR protocol then implies that $\mathcal{C}(w) = 1$. The witness w can be extracted from $\mathcal{P}$ as in the previous section via the special soundness of Protocol 7. In each round of the GKR protocol, the message to be sent by the GKR prover $\mathcal{P}$' in response to the GKR verifier's challenge can also be extracted from the commitments sent by the argument system prover $\mathcal{P}$ in response to the same challenge, because Protocol 7 was invoked in each round of the argument system to prove that $\mathcal{P}'$ knows openings to every commitment sent.

The probability that the GKR verifier accepts when interacting with $\mathcal{P}'$ is analyzed by exploiting the fact that the GKR verifier's checks on the committed messages sent by $\mathcal{P}'$ are performed by applying the zero-knowledge proof of knowledge of the previous section to $\mathcal{C}'$. Specifically, soundness of the argument system applied to $\mathcal{C}'$ ensures that whenever $\mathcal{P}$ convinces the argument system verifier to accept, $\mathcal{P}'$ convinces the GKR verifier to accept (up to the negligible probability with which

a polynomial time adversary is able to break binding property of the commitment scheme).

To summarize, in the above argument system, we essentially applied the commit-and-prove zero-knowledge argument of Section 13.1 not to $\mathcal{C}$ itself, but rather to the verifier in the GKR protocol applied to check the claim that $\mathcal{C}(w) = 1$.

Reducing the Dependence on Witness Size Below Linear. The argument system just described has communication complexity that grows linearly with $|w|$, because the prover sends a hiding commitment to each entry of w and proves in zero-knowledge that it knows openings to each commitment. The next section describes several practical polynomial commitment schemes. Rather than committing to each entry of w individually, the prover could commit to the multilinear extension $\widetilde{w}$ of w using an extractable polynomial commitment scheme as outlined in Section 7.3, and thereby reduce the dependence on the proof length on $|w|$ from linear to sublinear or even logarithmic. (More precisely, to ensure zero-knowledge, the polynomial commitment scheme should be hiding, and during its evaluation phase it should reveal to the verifier a hiding *commitment* to $\widetilde{w}(z)$ for any point z chosen by the verifier. See for example the multilinear polynomial commitment scheme in Section 14.3.)

This same approach also transforms the succinct MIP-derived argument of Section 8.3 into a zero-knowledge one. Specifically, after having the argument system prover first commit to the multilinear polynomial Z claimed to extend a valid circuit transcript (with a hiding commitment scheme), the prover and verifier then simulate the MIP verifier's interactions with the first MIP prover, but with the prover sending Pedersen commitments to the MIP prover's messages rather than the messages themselves, and proves in zero-knowledge that it knows openings for the commitments. The argument system verifier then confirms in zero-knowledge that the values inside these commitments would have convinced the MIP verifier to accept the claim.

The ideas in this section and the previous section were introduced by Cramer and Damgård [104] and first implemented and rendered practical via optimizations in [226], [244], [257].

13.3 Zero-Knowledge via Masking Polynomials

The preceding section (Section 13.2) gave a generic technique for transforming any IP into a zero-knowledge argument: the argument system prover mimics the IP prover, but rather than sending the IP prover's messages in the clear, it sends hiding commitments to those messages, and proves in zero-knowledge that it knows how to open the commitments. At the end of the protocol, the argument system prover establishes in zero-knowledge that the committed messages would have caused the IP verifier to accept; this is done efficiently by exploiting homomorphism properties of the commitments.

In this section, we discuss another technique for transforming any IP into a zero-knowledge argument. The technique makes use of any extractable polynomial commitment scheme, meaning that we assume the prover is able to cryptographically bind itself to a desired polynomial p, and later the verifier can force the prover to reveal the evaluation $p(r)$ for a random input r to p of the verifier's choosing. Suppose further that the polynomial commitment scheme is zero-knowledge, meaning that the verifier learns nothing about p from the commitment, and the evaluation phase reveals no information about p to the verifier other than the evaluation $p(r)$. One benefit of this technique is that if the polynomial commitment scheme is binding even against quantum cheating provers, then the resulting zero-knowledge argument is also plausibly post-quantum sound. For example, the FRI-based polynomial commitment scheme of Section 10.4.2 is plausibly sound against cheating provers that can run polynomial time quantum algorithms.[4] In contrast, any protocol (such as that of the last section) that makes use of Pedersen commitments is not post-quantum sound, because Pedersen commitments are only binding if the discrete logarithm problem is intractable, and quantum computers can compute discrete logarithms in polynomial time.

[4] FRI-derived polynomial commitments are not zero-knowledge, but can be rendered zero-knowledge using techniques similar to those in this section.

Another Zero-Knowledge Sum-Check Protocol. Consider applying the sum-check protocol to an ℓ-variate polynomial g over $\mathbb{F}$ to check the prover's claim that $\sum_{x \in \{0,1\}^\ell} g(x)$ equals some value G. Let us assume that the verifier has oracle access to g, in the sense that for any point $r \in \mathbb{F}^\ell$, the verifier can obtain $g(r)$ with one query to the oracle. Recall (Section 4.1) that the sum-check protocol consists of ℓ rounds, where the honest prover's message in each round i is a univariate polynomial of degree $\deg_i(g)$ derived from g, namely

$$\sum_{b_{i+1},\ldots,b_\ell \in \{0,1\}} g(r_1, \ldots, r_{i-1}, X_i, b_{i+1}, \ldots, b_\ell).$$

Here, $\deg_i(g)$ is the degree of g in variable i and is assumed known to the verifier, and $r_1, \ldots, r_{i-1}$ are random field elements chosen by the verifier in rounds $1, 2, \ldots, i-1$.

There are three ways in which the verifier "learns" information about g in the sum-check protocol. First, the verifier learns that $\sum_{x \in \{0,1\}^\ell} g(x) = G$, but this information is not meant to be "protected" as the entire point of the sum-check protocol is to ensure that the verifier learns this value. Second, the prover's messages leak information about g to the verifier that the verifier may not be able to compute on her own. Third, the verifier learns the value $g(r)$ via the oracle query at the end of the protocol.

In the preceding section, we addressed the second source of information leakage by having the prover send hiding commitments to the messages rather than the messages themselves. Here is a different technique for ensuring that the prover's messages do not leak any information about g to the verifier; this approach originated in [39], [97].

To ensure that the prover's messages in the sum-check protocol reveal no information about g, the prover can at the very start of the protocol choose a random polynomial p with the same degree as g in each variable, commit to p, and send to the verifier a value P claimed to equal $\sum_{x \in \{0,1\}^\ell} p(x)$. The verifier then picks a random $\rho \in \mathbb{F} \setminus \{0\}$ and sends it to the prover, and the prover and verifier apply the sum-check protocol not to g itself but rather to $g + \rho \cdot p$, to check that $\sum_{x \in \{0,1\}^\ell} (g + \rho \cdot p)(x) = G + \rho \cdot P$.

At the end of the sum-check protocol, the verifier needs to evaluate $g + \rho \cdot p$ at a random input $r \in \mathbb{F}^\ell$. The value $p(r)$ can be obtained via the evaluation phase of the commitment scheme that was applied to p, while $g(r)$ is obtained by the verifier with a single oracle query.

Completeness and soundness. The protocol clearly satisfies completeness. To see that it is sound, consider any prover strategy $\mathcal{P}$ capable of convincing the verifier to accept with non-negligible probability. By extractability of the polynomial commitment scheme, it is possible to efficiently extract from $\mathcal{P}$ a polynomial p such that the prover is bound to p, in the sense that any value revealed by the prover in the evaluation phase of the commitment scheme is consistent with p. Letting P be the claimed value of $\sum_{x\in\{0,1\}^\ell} p(x)$ sent by $\mathcal{P}$, consider the two functions $\pi_1(\rho) = G + \rho P$ and $\pi_2(\rho) = \sum_{x\in\{0,1\}^\ell} (g + \rho \cdot p)(x)$. Both are linear functions in ρ. If either $G \neq \sum_{x\in\{0,1\}^\ell} g(x)$ or $P \neq \sum_{x\in\{0,1\}^\ell} \rho \cdot p(x)$ then $\pi_1 \neq \pi_2$, and hence the two linear functions can agree on at most one value of ρ. This means that with probability at least $1 - \frac{1}{|\mathbb{F}|-1}$ over the random choice of ρ, $G + \rho P \neq \sum_{x\in\{0,1\}^\ell} (g + \rho \cdot p)(x)$. In this event, the sum-check protocol is applied to a false claim, and we conclude that the verifier will reject with high probability because the sum-check protocol is sound.

Honest-Verifier Zero-Knowledge. We claim that the honest verifier in this protocol learns nothing about g other than G and $g(r)$. This is formalized by giving an efficient simulator that, given G and the ability to query g at a single input r, produces a distribution identical to that of the prover's messages in the above protocol.

The intuition is that, since p is random, adding $\rho \cdot p$ to g yields a random polynomial satisfying the same degree bounds as g, and hence the prover's messages in the sum-check protocol applied to $g + \rho \cdot p$ are indistinguishable from those obtained by applying the sum-check protocol to a randomly chosen polynomial. Formally, the simulator selects a random polynomial p subject to the appropriate degree bounds (i.e., $\deg_i(p) = \deg_i(g)$ for all i), commits to p exactly as does the honest prover in the protocol above (here, we are using the fact that p is chosen totally independent of g and hence the simulator can commit to p even though the simulator has no knowledge of g), and sets

$P \leftarrow \sum_{x \in \{0,1\}^\ell} p(x)$. The simulator then chooses ρ at random from $\mathbb{F} \setminus \{0\}$, and chooses a random value $r = (r_1, \ldots, r_\ell) \in \mathbb{F}^\ell$. The simulator queries the oracle for g at r to obtain $g(r)$ and then chooses a random polynomial f subject to the constraint that f sums to G over inputs in $\{0,1\}^\ell$ and $f(r) = g(r)$ (this can be done in time $O(2^\ell)$, which is polynomial in n if $\ell = O(\log n)$). The simulator then computes the honest prover's messages in the sum-check protocol applied to $f + \rho p$ when the sum-check verifier's randomness is r. At the end of the sum-check protocol, when the verifier needs to learn $p(r)$, the simulator simulates the honest prover and verifier in the evaluation phase of the polynomial commitment scheme applied to reveal $p(r)$ to the verifier. This completes the description of how the simulator produces a simulated transcript of the verifier's interaction with the prover in the zero-knowledge sum-check protocol. We now explain why the simulated prover messages are distributed identically to those sent by the honest prover in response to the honest verifier.

By the zero-knowledge property of the polynomial commitment scheme, the evaluation proof of the commitment scheme can itself be simulated given $p(r)$ alone, and in particular does not depend on p's evaluations at any points other than r. This ensures that, conditioned on the values of $g(r)$, ρ, and the prover's messages during the evaluation phase of the polynomial commitment scheme, $q := g + \rho p$ is a random polynomial with the same variable degrees as g, subject to the constraints that $q(r) = g(r) + \rho \cdot p(r)$ and $\sum_{x \in \{0,1\}^\ell} q(x) = G + \rho \cdot P$. Since $f + \rho p$ is a random polynomial subject to the same constraints, the prover messages generated by the simulator are distributed identically to the honest prover's messages in the actual protocol (we omit details of how this assertion is achieved, as it does require modest modifications to the univariate and multilinear polynomial commitment schemes, because p is neither a univariate polynomial nor multilinear).

Costs. When the sum-check protocol is applied in an IP or MIP for circuit-satisfiability, the polynomial g to which it is applied has $\ell \approx 2 \log S$ or $\ell \approx 3 \log S$, where S is either the size of the circuit $\mathcal{C}$ or a the number of gates at a single layer of $\mathcal{C}$ (see Sections 4.6 and 8.2). This means that a random polynomial p with the same variable degree

as g has at least S^2 coefficients, so even writing p down takes time at least quadratic in S, which is totally impractical. Fortunately, Xie *et al.* [251] show that p does not actually need to be a random polynomial of the appropriate variable degrees. Rather, it suffices for p to be a sum of ℓ randomly chosen univariate polynomials $s_1, \ldots, s_\ell$, one for each of the ℓ variables of g, where the degree of s_i equals $\deg_i(g)$. This ensures that p can be committed to in time $\tilde{O}(\ell)$ using (zero-knowledge variants of) any of the polynomial commitment schemes discussed in this monograph.

Masking $g(r)$. When the sum-check protocol is applied to a polynomial g in an IP or MIP for circuit- or R1CS-satisfiability, allowing the verifier to learn even a single evaluation $g(r)$ of g will violate zero-knowledge, because g itself depends on the witness.

For example, recall that in the MIP for circuit satisfiability of Section 8.2, to check the claim that $\mathcal{C}(x, w) = y$, the prover applies the sum-check protocol exactly once, to the polynomial

$$h_{x,y,Z}(Y) \coloneqq \widetilde{\beta}_{3k}(r, Y) \cdot g_{x,y,Z}(Y).$$

(See Equation (8.2)). Here, Z denotes some polynomial mapping $\{0,1\}^{\log S}$ to $\mathbb{F}$, and

$$\begin{aligned} g_{x,y,Z}(a,b,c) \coloneqq &\widetilde{\text{io}}(a,b,c) \cdot (\widetilde{I}_{x,y}(a) - Z(a)) + \widetilde{\text{add}}(a,b,c) \cdot (Z(a) \\ &- Z(b) + Z(c)) + \widetilde{\text{mult}}(a,b,c) \cdot (Z(a) - Z(b) \cdot Z(c)). \end{aligned}$$

The honest prover in the MIP sets Z to the multilinear extension $\widetilde{W}$ of a correct transcript W for the claim that $\mathcal{C}(x, w) = y$ (where W is viewed as a function mapping $\{0,1\}^{\log S} \to \mathbb{F}$).

The key point above is that, since the correct transcript W fully determines the multilinear extension $\widetilde{W}$, and W depends on the witness w, even a single evaluation of $\widetilde{W}$ leaks information about w to the verifier. Hence, any zero-knowledge argument system cannot reveal $\widetilde{W}(r)$ to the verifier for even a single point r.

Here is a technique for addressing this issue. In a sentence, the idea is to replace $\widetilde{W}(r)$ with a slightly higher-degree (randomly chosen) extension polynomial Z of W. This ensures that if the verifier learns

a couple of evaluations $Z(r_1)$, $Z(r_2)$ of Z, so long as $r_1, r_2 \notin \{0,1\}^\ell$, these evaluations are simply independent random field elements and in particular are totally independent of the transcript W.

In more detail, recall further that in argument systems derived from the MIP, the prover uses a polynomial commitment scheme to commit to an extension Z of a correct transcript W. The verifier ignores the commitment until the very end of the sum-check protocol applied to $h_{x,y,Z}$, at which time the verifier needs to evaluate $h_{x,y,Z}$ at a random input $r = (r_1, r_2) \in \mathbb{F}^{\log S} \times \mathbb{F}^{\log S}$. Assuming the verifier can efficiently evaluate $\widetilde{\text{io}}$, $\widetilde{I}$, $\widetilde{\text{add}}$, and $\widetilde{\text{mult}}$ on its own, $h_{x,y,Z}(r)$ can be easily computed given $Z(r_1)$ and $Z(r_2)$. The verifier obtains these two values using the evaluation phase of the polynomial commitment scheme.

As discussed above, in the MIP of Section 8.2, the prover sets Z to $\widetilde{W}$, but this does not yield a zero-knowledge argument. Instead, let us modify the protocol as follows to achieve perfect zero-knowledge. First, we insist that the verifier choose the coordinates of r from $\mathbb{F} \setminus \{0,1\}$ rather than from $\mathbb{F}$ (this has a negligible effect on soundness). Second, we prescribe that the honest prover chooses Z to be a random extension polynomial of the correct transcript W where Z has at least two more coefficients than a multilinear polynomial. For example, we can prescribe that the prover set

$$Z(X_1, \ldots, X_{\log S}) := \widetilde{W}(X_1, \ldots, X_{\log S}) + c_1 X_1(1 - X_1) + c_2 X_2(1 - X_2),$$

where the prover chooses c_1 and c_2 at random. Since $X_1(1 - X_1)$ and $X_2(1 - X_2)$ vanish on inputs in $\{0,1\}^2$, it is clear that Z extends W. Basic linear algebra implies that for any two points $r_1, r_2 \in \mathbb{F}^{\log S} \setminus \{0,1\}^{\log S}$, $Z(r_1)$ and $Z(r_2)$ are uniform random field elements, independent of each other and of W. Third, as in the zero-knowledge sum-check protocol described earlier in this section, we insist that the polynomial commitment scheme used to commit to Z is zero-knowledge, meaning that the verifier learns nothing from the commitment or the prover's messages in the evaluation phase of the protocol other than the requested evaluations $Z(r_1)$ and $Z(r_2)$. Fourth, rather than directly applying the sum-check protocol to $h_{x,y,Z}$, we apply the zero-knowledge variant of the sum-check protocol described earlier in this section (with g set to $h_{x,y,Z}$).

The modified argument system clearly remains complete, and it is sound, as the (zero-knowledge) sum-check protocol applied to $h_{x,y,Z}$ confirms with high probability that Z is an extension polynomial of a valid transcript W. It is also perfect zero-knowledge. The simulator is essentially the same as that for the zero-knowledge sum-check protocol. The primary modification of the simulator is that, for $r = (r_1, r_2)$, the simulator's oracle query to obtain $g(r)$ is replaced with the following procedure. First, the simulator chooses values $Z(r_1)$ and $Z(r_2)$ to be random field elements, and then derives $g(r)$ based on these values, according to the definition of $g = h_{x,y,Z}$ in Equation (8.2). Second, the simulator simulates the evaluation proof from the polynomial commitment scheme for the evaluations $Z(r_1)$, $Z(r_2)$ using the zero-knowledge property of the commitment scheme.

Costs. The argument system above is essentially the same as the nonzero-knowledge argument system derived from the MIP of Section 8.2, since all we did was replace the multilinear extension $\widetilde{W}$ with a slightly higher-degree extension Z. Committing to the extension Z as above does require minor modification of the polynomial commitment schemes covered in this survey, as Z is not multilinear (we omit these details for brevity). However, the modifications add very little cost to the commitment protocol, since Z has only two more coefficients than $\widetilde{W}$.

13.4 Discussion and Comparison

This section provided two quite general techniques for transforming a non-zero-knowledge protocol $\mathcal{Q}$ into zero-knowledge one $\mathcal{Z}$. This allows protocol designers to first design an efficient protocol $\mathcal{Q}$ without having to worry about zero-knowledge, and then apply one of the two transformations to "add" zero-knowledge (hopefully, with minimal concrete overhead or additional cognitive load).

Here is a recap of the first transformation. Suppose in the non-zero-knowledge protocol $\mathcal{Q}$, all messages from prover to verifier consist of elements of some field $\mathbb{F}$. Section 13.2 would render the protocol zero-knowledge via the "commit-and-prove" approach as follows: for every field element sent by the prover in $\mathcal{Q}$, the prover in $\mathcal{Z}$ would

instead send a *hiding commitment* to that field element, and then at the end of the protocol the prover in $\mathcal{Z}$ would prove in zero-knowledge (via the proof system of Section 13.1) that the committed values would have caused the $\mathcal{Q}$ verifier to accept.

The downsides of this first transformation are two-fold: first, if (as with Pedersen commitments) the commitment scheme is not binding against quantum adversaries, then $\mathcal{Z}$ will not be post-quantum-sound even if $\mathcal{Q}$ is. Second, as discussed already in Section 13.1.3, verification costs are higher in $\mathcal{Z}$ than in $\mathcal{Q}$, first because commitments to field elements may be larger than the field elements themselves (thereby increasing proof length) and second because the $\mathcal{Z}$ verifier must effectively run the $\mathcal{Q}$ verifier "on the committed values", without opening the commitments, and this will further increase proof size and verifier time. For example, each field multiplication that the $\mathcal{Q}$ verifier does may turn into an invocation of Protocol 9 from Section 12.3.2, which requires the prover to send at least 9 extra group elements and the verifier to perform at least nine group exponentiations and several group multiplications, rather than one field multiplication.

While this may appear to be a massive overhead in verifier time and proof length, many non-zero-knowledge protocols $\mathcal{Q}$ will make use of a polynomial commitment scheme. If the commitment scheme is hiding, i.e., it reveals no information to the verifier about the committed polynomial, then the messages sent by the prover of $\mathcal{Q}$ within the scheme do not need to be fed through the commit-and-prove transformation (see the paragraph "Reducing the dependence on witness size below linear" in Section 13.2 for further discussion). In these settings, the verification overhead introduced by the commit-and-prove transformation may be a low-order cost relative to that of the polynomial commitment.

Similarly, from the perspective of the prover's runtime, the transformation from $\mathcal{Q}$ to $\mathcal{Z}$ does not add much overhead so long as $\mathcal{Q}$ is succinct (i.e., the proof length in $\mathcal{Q}$ is much smaller than the size of the statement being proven). This is because, from the prover's perspective, all of the cryptographic overhead of the transformation (namely sending commitments to field elements rather than the field elements themselves, and proving in zero-knowledge that the committed values would have

caused the $\mathcal{Q}$ verifier to accept) is applied only to the *verification procedure* in $\mathcal{Q}$. If this verification procedure is much simpler than statement being proven, this computational overhead should be dwarfed by the cost of simply processing the statement itself, which is a lower bound on the prover's runtime in $\mathcal{Q}$.

If $\mathcal{Q}$ is based on the sum-check protocol (Section 4.2), then the second transformation of this section, based on masking-polynomials (Section 13.3) can be applied. This has the dual benefits of plausibly preserving post-quantum soundness, and typically adding less overhead than the commit-and-prove-based transformation. With masking polynomials, the main extra cost in the resulting zero-knowledge protocol $\mathcal{Z}$ compared to the non-zero-knowledge protocol $\mathcal{Q}$ is that the prover has to commit to one masking polynomial for each invocation of the sum-check protocol in $\mathcal{Q}$, and the verifier has to obtain an evaluation of the committed masking polynomial. As described in Section 13.3, these masking polynomials can typically be made very small (of size linear in communication cost of the sum-check protocol, which is typically just logarithmic in the size of the statement being proven).

On the other hand, the masking-polynomial-based transformation is conceptually more complicated and ad hoc than the "commit-and-prove" approach, and accordingly is not as general: it applies only to sum-check-based protocols $\mathcal{Q}$ (though related techniques typically can be used to render other polynomial-based protocols zero-knowledge, such as those in Section 10.3).