7

A First Succinct Argument for Circuit Satisfiability, from Interactive Proofs

Arguments for Circuit-SAT. Recall from Section 6.5.1 that in the arithmetic circuit *satisfiability* problem, a designated circuit $\mathcal{C}$ takes two inputs, x and w. The first input x is public and fixed, i.e., known to both the prover and verifier. The second input w is often called the *witness*, or sometimes the *non-deterministic input* or *auxiliary input*. Given the first input x and output(s) y, the prover in an argument system for circuit-satisfiability wishes to establish that *there exists* a witness w such that $\mathcal{C}(x, w) = y$.

In Section 6.5 we saw an efficient way to turn any computer program into an equivalent instance of the arithmetic circuit satisfiability problem. Roughly, we showed that the problem of checking whether a Random Access Machine M taking at most T steps on an input of size x produces output y can be reduced to a circuit satisfiability instance $(\mathcal{C}, x, y)$, where $\mathcal{C}$ has size close to T and depth close to $O(\log T)$. That is, M outputs y on x if and only if there exists a w such that $\mathcal{C}(x, w) = y$.

This transformation is only useful in the context of interactive proofs and arguments if we can design efficient proof systems for solving instances of circuit satisfiability. In this section, we will see our first

example of such an argument system, by combining the GKR protocol with a cryptographic primitive called a *polynomial commitment scheme*, in a manner that was already outlined in detail in Section 6.5.2. The polynomial commitment scheme we describe in this section is conceptually appealing but highly impractical; more practical polynomial commitment schemes will be covered later in this monograph.

Specifically, the polynomial commitment scheme in this section combines a technique called Merkle-hashing with a protocol called a *low-degree test*. Low-degree tests themselves tend to be very simple protocols, though the analysis showing that they work is very complicated and omitted from this survey. More practical commitments that we will see in Section 10 replace the low-degree test with more efficient interactive variants—the interaction can then be removed via the Fiat-Shamir transformation. Other polynomial commitments based on very different techniques are covered in Sections 14–16.

Arguments of Knowledge and SNARKs. Arguments for circuit satisfiability are particularly useful when they satisfy an enhanced soundness property called *knowledge-soundness*. Informally, this means that the prover establishes not only that there *exists* a witness w such that $\mathcal{C}(x, w) = y$, but in fact that the prover *knows* such a w.

Knowledge-soundness can be a meaningful notion even when standard soundness is not. For example, suppose the prover and verifier agree on a cryptographic hash function h and hash value y, and the prover claims to know a w such that $h(w) = y$. The prover can establish this by applying a *knowledge-sound* argument for circuit-satisfiability to a circuit $\mathcal{C}$ that takes w as input and computes $h(w)$.

An argument satisfying standard soundness, which merely guarantees the *existence* of a witness w such that $\mathcal{C}(w) = y$, would be useless in this context. This is because cryptographic hash functions are typically surjective, meaning for *any* y there will *exist* many pre-images w. Accordingly, the trivial proof system where the verifier always accepts satisfies standard soundness in this context, but not knowledge-soundness.

Knowledge-sound arguments can be particularly useful when they are non-interactive, meaning the proof is just a static string that is accepted or rejected by the verifier, and *succinct*, meaning that the

proofs are very short. Such arguments are called SNARKs. In Section 7.4, we explain that the succinct arguments we give in this section are in fact SNARKs.

7.1 A Naive Approach: An IP for Circuit Satisfiability

A naive way to use the GKR protocol to solve circuit satisfiability is to have the prover explicitly send to the verifier the witness w satisfying $\mathcal{C}(x, w) = y$, and then running the GKR protocol to check that indeed $\mathcal{C}(x, w) = y$. The problem with this simple approach is that in many settings w can be very large. For example, in the transformation from Section 6.5, the witness w is supposed to be a transcript of M's entire execution on input x, and hence w has size at least T. This means that in the time that the verifier would take to read the whole witness, the verifier could have run M on x without any help from the prover.

7.2 Succinct Arguments for Circuit Satisfiability

If an argument system for circuit satisfiability avoids the above bottleneck of sending the entire witness to the verifier, then it is called *succinct*. Formally, we say that an argument system for circuit satisfiability is succinct if the total communication is sublinear in the size of the witness $|w|$.[1] Succinctness is important for a variety of reasons:

- Shorter proofs are always better. For example, in some applications to blockchains, proofs may be stored on the blockchain permanently. If proofs are long, it drastically increases the global storage requirements of the blockchain. For many (but not all) argument systems, shorter proofs also result in faster verification.
- In some applications, witnesses are naturally large. For example, consider a hospital that publishes cryptographic hash $h(w)$ of a

[1] Here, sublinear in $|w|$ means $o(|w|)$, i.e., any expression that asymptotically is much smaller than the witness length. This use of the term "succinct" is slightly nonstandard, as many works reserve the term *succinct* for any proof or argument systems in which the total communication is polylogarithmic (or even logarithmic) in the witness length (or even in the circuit size). Some others use succinctness more informally to refer broadly to argument systems with short proofs.

massive database w of patient records, and later wants to prove that it ran a specific analysis on w. In this case, the witness is the database w, the public input x is the hash value $h(w)$, and the circuit $\mathcal{C}$ should both implement the analysis of w and "check" that $h(w) = x$.

- Efficient transformations from computer programs to circuit satisfiability often produce circuits with very large witnesses (see Section 6.5).

The coming sections will describe a variety of approaches to obtaining succinct arguments.[2] This section will cover one specific approach.

7.3 A First Succinct Argument for Circuit Satisfiability

7.3.1 The Approach

The approach of this section is to "simulate" the trivial application of the GKR protocol to circuit satisfiability described in Section 7.1, but *without* requiring the prover to explicitly send w to the verifier. We will accomplish this by using a cryptographic primitive called a *polynomial commitment scheme*. The idea of combining the GKR protocol with polynomial commitment schemes to obtain succinct arguments was first put forth by Zhang *et al.* [258]. We cover polynomial commitment schemes with state-of-the-art concrete efficiency in Sections 10.4.4 and 10.5, and in Section 14. In this section, we informally introduce the notion of polynomial commitment schemes and sketch a conceptually simple (but impractical) polynomial commitment scheme based on low-degree tests and Merkle trees.

[2] There is strong evidence that succinct interactive *proofs* (as opposed to arguments) for circuit satisfiability do not exist [72], [132], [205], [246]. For example, it is known [132] that interactive proofs for circuit satisfiability cannot have communication cost that is *logarithmic* in the witness length unless **coNP** $\subseteq$ **AM**, and this is widely believed to be false (i.e., it is not believed that there are efficient constant-round interactive proofs to establish that a circuit is *un*satisfiable). Similar, though quantitatively weaker, surprising consequences would follow from the existence of interactive proofs for circuit satisfiability with sublinear (rather than logarithmic) communication cost.

Cryptographic Commitment Schemes. Conceptually, cryptographic commitment schemes can be described via the following metaphor. They allow the committer to take some object b (b could be a field element, vector, polynomial, etc.) place b in a box and lock it, and then send the locked box to a "verifier". The committer holds on to the key to the lock. Later, the verifier can ask the committer to open the box, which the committer can do by sending the verifier the key to the lock. Most commitment schemes satisfy two properties: hiding and binding. In the metaphor, hiding means that the verifier can not "see inside" the locked box to learn anything about the object within it. Binding means that once the box is locked and transmitted to the verifier, the committer cannot change the object within the box. We provide a far more detailed and formal treatment of cryptographic commitment schemes much later in this survey, in Section 12.3, and of polynomial commitment schemes in particular at the start of Section 14.

Polynomial Commitment Schemes. Roughly speaking, a polynomial commitment scheme is simply a commitment scheme in which the object being committed to is (all evaluations of) a low-degree polynomial. That is, a polynomial commitment scheme allows a prover to commit to a polynomial $\tilde{w}$ satisfying a specified degree bound, and later reveal $\tilde{w}(r)$ for a point r of the verifier's choosing. Even though in the commitment phase the prover does *not* send all evaluations of $\tilde{w}$ to the verifier, the commitment still effectively binds the prover to a specific $\tilde{w}$. That is, at a later time, the verifier can ask the prover to reveal $\tilde{w}(r)$ for any desired r of the verifier's choosing, and the prover is effectively forced to reveal $\tilde{w}(r)$ for a fixed polynomial $\tilde{w}$ determined at the time of the original commitment. In particular, the prover is unable to choose the polynomial $\tilde{w}$ to depend on the query point r, at least not without breaking the computational assumption on which security of the commitment scheme is based.

Combining Polynomial Commitment Schemes and the GKR Protocol. When applying the GKR protocol to check that $\mathcal{C}(x, w) = y$, the verifier does not need to know any information whatsoever about w until the very end of the protocol, when (as explained in Section 7.3.2.1

below) the verifier only needs to know $\tilde{w}(r)$ for a randomly chosen input r.

So rather than having the prover send w in full to the verifier as in Section 7.1, we can have the prover merely send a *cryptographic commitment* to $\tilde{w}$ at the start of the protocol. The prover and verifier can then happily apply the GKR protocol to the claim that $\mathcal{C}(x, w) = y$, ignoring the commitment entirely until the very end of the protocol. At this point, the verifier needs to know $\tilde{w}(r)$. The verifier can force the prover to reveal this quantity using the commitment protocol.

Because the polynomial commitment scheme *bound* the prover to a fixed multilinear polynomial $\tilde{w}$, the soundness analysis of the argument system is essentially the same as if the prover had sent all of w explicitly to the verifier at the start of the protocol as in Section 7.1 (see Section 7.4 for additional details of how one formally analyzes the soundness of this argument system).

7.3.2 Details

7.3.2.1 What The GKR Verifier Needs to Know About The Witness

In this subsection, we justify the assertion from Section 7.3.1 that the only information the verifier needs about w in order to apply the GKR protocol to check that $\mathcal{C}(x, w) = y$ is $\tilde{w}(r_1, \ldots, r_{\log n})$.

Let z denote the concatenation of x and w. Let us assume for simplicity throughout this section that x and w are both of length n, so that each entry of z can be assigned a unique label in $\{0,1\}^{1+\log n}$, with the ith entry of x assigned label $(0, i)$, and the ith entry of w assigned label $(1, i)$.

A key observation is that when applying the GKR protocol to check that $\mathcal{C}(z) = y$, the verifier doesn't need to know the exact value of z. Rather, the verifier only needs to know $\tilde{z}(r_0, \ldots, r_{\log n})$ at a single, randomly chosen input $(r_0, \ldots, r_{\log n})$. Moreover, the verifier doesn't even need to know $\tilde{z}(r)$ until *the very end of the protocol*, after the interaction with the prover has finished. We now explain that in order to calculate $\tilde{z}(r)$, it suffices for the verifier to know $\tilde{w}(r_1, \ldots, r_{\log n})$.

It is straightforward to check that

$$\tilde{z}(r_0, r_1, \ldots, r_{\log n}) = (1 - r_0) \cdot \tilde{x}(r_1, \ldots, r_{\log n}) + r_0 \cdot \tilde{w}(r_1, \ldots, r_{\log n}). \tag{7.1}$$

Indeed, the right hand side is a multilinear polynomial in $(r_0, r_1, \ldots, r_{\log n})$ that evaluates to $z(r_0, \ldots, r_{\log n})$ whenever $(r_0, \ldots, r_{\log n}) \in \{0,1\}^{1+\log n}$.[3] By Fact 3.5, the right hand side of Equation (7.1) must equal the unique multilinear extension of z.

Equation (7.1) implies that, given $\tilde{w}(r_1, \ldots, r_{\log n})$, the verifier can evaluate $\tilde{z}(r_0, \ldots, r_{\log n})$ in $O(n)$ time, since the verifier can evaluate $\tilde{x}(r_1, \ldots, r_{\log n})$ in $O(n)$ time (see Lemma 3.8).

In summary, the GKR protocol has the (a priori) amazing property that in order for the verifier to apply it to a known circuit $\mathcal{C}$ on input $z = (x, w) \in \mathbb{F}^n \times \mathbb{F}^n$, the verifier does not need to know anything at all about w other than a *single* field element, namely a single evaluation of $\tilde{w}$. Moreover, the verifier doesn't even need to know this single field element until the very end of the protocol, after the entire interaction with the prover has terminated.

7.3.2.2 A First (Relaxed) Polynomial Commitment Scheme

There are a number of ways to design polynomial commitment schemes. In this section, we describe a simple, folklore commitment scheme (it was also explicitly proposed by Yael Kalai [160]). This scheme is impractical owing to a large prover runtime (see the paragraph on "Costs of this succinct argument system" later in this section), but it provides a clean and simple introduction to cryptographic commitment schemes. We will see (much) more efficient examples of polynomial commitment schemes later in the survey.[4]

[3] To see that the latter statement holds, observe that the right hand side of Equation (7.1) evaluates to $\tilde{x}(r_1, \ldots, r_{\log n})$ when $r_0 = 0$ and to $\tilde{w}(r_1, \ldots, r_{\log n})$ and when $r_0 = 1$. Since $\tilde{x}$ and $\tilde{w}$ extend x and w respectively, this means that the right hand side extends the concatenated input (x, w).

[4] In particular, practical argument systems have replaced low-degree tests with interactive variants that have far superior concrete efficiency (see Sections 10.4.4 and 10.5); the interaction can then be removed from the argument via the Fiat-Shamir transformation. For this reason, we do not cover low-degree tests or their analysis in detail in this survey.

To be more precise, the scheme we give here is not a genuine polynomial commitment scheme, because it only binds the prover to a function that is *close* to a polynomial. We call this a *relaxed* polynomial commitment scheme. As we will see, even this relaxed guarantee is enough to transform the GKR protocol into a succinct argument for circuit-satisfiability.

The scheme makes essential use of two important concepts: Merkle Trees and low-degree tests.

Merkle Trees. A Merkle tree [191] (sometimes also called a hash tree) can be used to design a *string-commitment scheme*, which allows a sender to send a short commitment to a *string* $s \in \Sigma^n$ for any finite alphabet Σ.[5] Later, the sender can efficiently reveal the value of any entries of s that are requested by the receiver.

Specifically, a Merkle tree makes use of a collision-resistant hash function h mapping inputs to $\{0,1\}^\kappa$ where κ is a security parameter that in practice is typically on the order of several hundred.[6,7]

[5] Many treatments of Merkle trees use the phrase *vector* commitment rather than string commitment. We use the phrase string commitment in this section to clarify that the alphabet Σ need not be numerical in nature, but rather can be any finite set.

[6] For example, SHA-3 allows for several output sizes, from as small as 224 bits to as large 512 bits.

[7] As discussed in Section 5.1, cryptographic hash functions such as SHA-3 or BLAKE3 are designed with the goal of ensuring that they "behave like" truly random functions. In particular, for such cryptographic hash functions it is typically assumed that the fastest way to find a collision is via exhaustive search, i.e., randomly choosing inputs at which to evaluate the hash function until a collision is found. If the hash function were a truly random function mapping to range $\{0,1\}^\kappa$, then by the *birthday paradox*, with high probability roughly $\sqrt{2^\kappa} = 2^{\kappa/2}$ evaluations must be performed before exhaustive search finds a collision. This means that for security against attackers running in time, say, 2^{128}, the output size of the hash function should consist of at least $\kappa = 256$ bits. Obtaining security against attackers running in time 2^λ is often referred to by saying the primitive "achieves λ bits of security", and λ is called the *security parameter* (see Section 5.3.1). Quantum algorithms are in principle capable of finding collisions in random functions with codomain $\{0,1\}^\kappa$ in time $2^{\kappa/3}$ via a combination of Grover's algorithm and random sampling [78], meaning that κ should be set larger by a factor of $\frac{3}{2}$ to achieve security against the same number of quantum rather than classical operations.

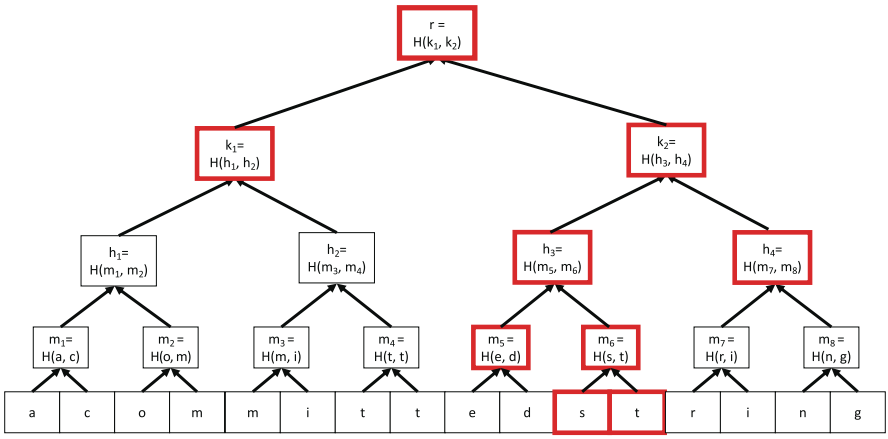

Figure 7.1: A Merkle-tree committing to the string "acommittedstring" using hash function H. Boxes with a red, bold outline represent the authentication path to reveal the twelfth entry of the committed string, namely the letter t. This consists of every node along the path from the root to the twelfth leaf, as well as each such node's sibling.

The leaves of the tree are the symbols of a string s, and every internal node of the tree is assigned the hash of its two children. Figure 7.1 provides a visual depiction of a hash tree.

One obtains a string-commitment protocol from a Merkle tree as follows. In the commitment step, the sender commits to the string s by sending the root of the hash-tree.

If the sender is later asked to reveal the ith symbol in s, the sender sends the value of the ith leaf in the tree (i.e., s_i), as well as the value of every node v along the root-to-leaf path for s_i, and the sibling of each such node v. We call all of this information the *authentication information* for s_i. The receiver checks that the hash of every two siblings sent equals the claimed value of their parent.[8]

Since the tree has depth $O(\log n)$, this translates to the sender sending $O(\log n)$ hash values per symbol of s that is revealed.

[8]In fact, it is unnecessary to include in the authentication information the hash values for the nodes along the root-to-leaf path, as for each such node the receiver can infer its claimed values by hashing its two children.

The scheme is binding in the following sense. For each index i, there is at most one value s_i that the sender can successfully reveal without finding a collision under the hash function h. This is because, if the sender is able to send valid authentication information for two different values s_i and s_i', then there must be at least one collision under h along the root-to-leaf path connecting the root to the ith leaf, since the authentication information for both s_i and s_i' result in the same root hash value, but differ in at least one leaf hash value.

A (Relaxed) Polynomial Commitment Scheme From a Merkle Tree? One could attempt to obtain a polynomial commitment scheme directly from a Merkle tree as follows. To have the prover $\mathcal{P}$ commit to a polynomial p, $\mathcal{P}$ could Merkle-commit to the string consisting of all evaluations of the polynomial, i.e., $p(\ell_1), \ldots, p(\ell_N)$ where $\ell_1, \ldots, \ell_N$ is an enumeration of all possible inputs to the polynomial. This would enable the prover to reveal any requested evaluation of the polynomial: if the verifier asks for $p(\ell_i)$, the prover can reply with $p(\ell_i)$ along authentication information for this value (the authentication information consists of $O(\log N)$ hash values).

Unfortunately, this approach does *not* directly yield a polynomial commitment scheme. The reason is that while the Merkle tree does bind the prover to a fixed string, there is no guarantee that the string is equal to all evaluations of some multilinear polynomial. That is, when the verifier $\mathcal{V}$ asks $\mathcal{P}$ to reveal $p(r)$ for some input r, the binding nature of Merkle trees does force $\mathcal{P}$ to respond with the r'th entry of the committed string and the associated authentication information. But $\mathcal{V}$ has no idea whether the committed string consists of all evaluations of a multilinear polynomial—the committed string could in general consists of all evaluations of some totally arbitrary function.

To address this issue, we combine Merkle trees with a low-degree test. The low-degree test ensures that not only is the prover bound to some (possibly completely unstructured) string, but actually that the string contains all evaluations of a low-degree polynomial. More precisely, it ensures that the string is "close" to the evaluation-table of a low-degree polynomial, so its use here yields a somewhat weaker object than an actual polynomial commitment scheme. The low-degree test

guarantees this despite only inspecting a small number of entries of the string—often logarithmic in the length of the string—thereby keeping the amount of authentication information transmitted by the prover low (at least, lower than the communication that would be required to explicitly send a complete description of the polynomial to the verifier). Details follow.

Low-Degree Tests. Suppose a receiver is given oracle access to a giant string s, which is claimed to contain all evaluations of an m-variate function over a finite field $\mathbb{F}$. Note that there are $|\mathbb{F}|^m$ such inputs, so s consists of a list of $|\mathbb{F}|^m$ elements of $\mathbb{F}$. A low-degree test allows one determine to whether or not the string is consistent with a low-degree polynomial, by looking at only a tiny fraction of symbols within the string.

Unfortunately, because the low-degree test only looks at a tiny fraction of s, it cannot determine whether s is *exactly* consistent with a low-degree polynomial. Imagine if s were obtained from a low-degree polynomial p by changing its value on only one input. Then unless the test gets lucky and chooses the input on which s and p disagree, the test has no hope of distinguishing between s and p itself.[9]

What the low-degree test can guarantee, however, is that s is *close* in Hamming distance to (the string of all evaluations of) a low-degree polynomial. That is, if the test passes with probability γ, then there is a low-degree polynomial that agrees with s on close to a γ fraction of points.

Typically, low-degree tests are extremely simple procedures, but they are often very complicated to analyze and existing analyses often involve very large constants that result in weak guarantees unless the field size is very large. An example of such a low-degree test is the point-versus-line test of Rubinfeld and Sudan, with a tighter analysis

[9]The word "test" in the phrase low-degree test has precise technical connotations. Specifically, it refers to the fact that if a function passes the test, then the function is only guaranteed to be "close" to a low-degree polynomial, i.e., it may not be exactly equal to a low-degree polynomial. This is the same sense that the word test is used in the field of property testing (see https://en.wikipedia.org/wiki/Property_testing). We reserve the word "test" throughout this monograph to have this technical connotation.

subsequently given by Arora and Sudan [12]. In this test, one evaluates s along a randomly chosen line in $\mathbb{F}^m$, and confirms that s restricted to this line is consistent with a univariate polynomial of degree at most m (see Section 4.5.2). Clearly, if the string s agrees perfectly with a multilinear polynomial then this test will always pass. The works [12], [219] roughly show that if the test passes with probability γ, then there is a low-degree polynomial that agrees with s at close to a γ fraction of points.[10] In this survey, we will not discuss how these results are proved.

A (Relaxed) Polynomial Commitment Scheme by Combining Merkle Trees and Low-Degree Tests. Let $\tilde{w}\colon \mathbb{F}^{\log n} \to \mathbb{F}$ be a $(\log n)$-variate multilinear polynomial over $\mathbb{F}$. Let s be the string consisting of all $|\mathbb{F}|^{\log n}$ evaluations of $\tilde{w}$. One obtains a polynomial commitment scheme by applying the Merkle-tree based string commitment scheme of Section 7.3.2.2, and then applying a low-degree test to s. For example, if the point-versus-line low-degree test is used, then the receiver picks a random line in $\mathbb{F}^{\log n}$, asks the sender to provide authentication information for all points along the line, and checks that the revealed values are consistent with a univariate polynomial of degree at most $\log n$.

The guarantee of this commitment scheme is the same as in the string-commitment scheme of Section 7.3.2.2, except that the use of the low-degree test ensures that if the sender passes all of the receivers checks with probability γ, then not only is the sender bound to a fixed string s, but also that there is some low-degree polynomial that agrees with s at close to a γ fraction of points.

This guarantee is enough to use the commitment scheme in conjunction with the GKR protocol applied to the claim $\mathcal{C}(x, w) = y$, as outlined in Section 7.3.1. Specifically, if the verifier's checks in the polynomial commitment scheme pass with probability at least (say) $1/2$, then the prover is bound to a string s such that there is a multilinear polynomial p that agrees with s on close to a $1/2$ fraction of points. As long as the point $(r_1, \ldots, r_{\log n})$ at which the verifier in the GKR protocol evaluates

[10] More precisely, these works show that there is a polynomial of total degree at most d that agrees with s on at least a $\gamma - m^{O(1)}/|\mathbb{F}|^{\Omega(1)}$ fraction of points. This fraction is $\gamma - o(1)$ as long as $|\mathbb{F}|$ is super-polynomially large in m (or even a large enough polynomial in m).

s is not one of the "bad" points on which s and p disagree, then the soundness analysis of the GKR protocol applies exactly as if the prover were bound to the multilinear polynomial p itself.

This is enough to argue that if the prover passes all of the verifier's checks with probability significantly larger than $1/2$, then indeed there exists a w (namely, the restriction of p to the domain $\{0,1\}^{\log n}$) such that $\mathcal{C}(x,w) = y$. The soundness error can be reduced from roughly $1/2$ to arbitrarily close to 0 by repeating the protocol many times and rejecting if any of the executions ever results in a rejection.[11]

Costs of this succinct argument system. In addition to the communication involved in applying the GKR protocol to check that $\mathcal{C}(x,w) = y$, the argument system above requires additional communication for the prover to commit to $\tilde{w}$ and execute the point-versus-line low-degree test. The total communication cost due to the (relaxed) polynomial commit scheme is $O(|\mathbb{F}| \cdot \log n)$ hash values (the cost is dominated by the cost of the prover revealing the value of $\tilde{w}$ on all $|\mathbb{F}|$ points along a line chosen by the verifier). This is $O(n)$ hash values as long as $|\mathbb{F}| \leq n/\log n$. Note that, while in practice we prefer to work over large fields, the soundness error of the GKR protocol is $O\left(\frac{d \log |\mathcal{C}|}{|\mathbb{F}|}\right)$ where d is the circuit depth, so working over a field of size $O(n/\log n)$ is enough to ensure non-trivial soundness error in the GKR protocol as long as $d \log |\mathcal{C}| \ll n/\log n$.

The verifier's runtime is the same as in the GKR protocol, plus the time required to play its part of the polynomial commit scheme. Assuming the collision-resistant hash function h can be evaluated in constant time, and the field size is $O(n/\log^2 n)$, the verifier spends $O(n)$ time to execute its part of the polynomial commitment scheme.

The prover's runtime in the above argument system is dominated by the time required to commit to $\tilde{w}$. This requires building a Merkle tree over all possible evaluations of $\tilde{w}$, of which there are $|\mathbb{F}|^{\log n}$. If we work over a field of size (say) $O(n)$, then this runtime is $n^{O(\log n)}$, which is

[11] It is possible to use a so-called *list-decoding guarantee* of the low-degree test to argue that the soundness error is much lower than $1/2$ (if the field size is large enough), without the need for repetition of the protocol. See Section 8.2.1.4 for details.

superpolynomial. So, as described, this polynomial commitment scheme is asymptotically efficient for the verifier, but not the prover.

Remark 7.1. It is possible to reduce the prover's runtime to $O(n^c)$ for some constant c in the above argument system. The way to do this is to tweak the parameters within the GKR protocol to enable working over a much smaller field, of size $O(\text{polylog}(n))$. This will be explained in more detail in Section 9.3 when we talk about designing succinct arguments from PCPs and multi-prover interactive proofs. However, the resulting prover runtime will still be impractical (practicality requires a prover runtime close to *linear*, rather than polynomial, in the size of the circuit-satisfiability instance). As indicated above, working over such a small field would also lead to soundness error of $1/\text{polylog}(n)$, and the protocol would have to be repeated many times to drive the soundness error low enough for cryptographic use.

Remark 7.2. An alternative polynomial commitment scheme would be to use Merkle trees to have the prover commit to the string consisting of the n coefficients of the multilinear polynomial $\tilde{w}\colon \mathbb{F}^{\log n}$, rather than to the $|\mathbb{F}|^{\log n}$ evaluations of $\tilde{w}$. This approach would have the benefit of allowing the commitment to be computed with $O(n)$ cryptographic hash evaluations, and the commitment would remain small (consisting simply of the root hash evaluation). However, in order to reveal the evaluation $\tilde{w}(r)$ for a point $r \in \mathbb{F}^n$, the prover would have to reveal all n of the coefficients of $\tilde{w}$, resulting in linear communication complexity and verifier runtime. This is no more efficient than the naive interactive proof from Section 7.1 in which $\mathcal{P}$ simply sends w to $\mathcal{V}$ at the start of the protocol. And the naive approach has the benefit of being statistically rather than computationally sound.

7.4 Knowledge-Soundness

Proofs and Arguments of Knowledge. The notion of a proof or argument of knowledge is meant to capture situations in which a prover establishes not only that a statement is true, but also that the prover *knows* a "witness" w to the validity of the statement. For example, in the authentication application of Section 1, Alice chooses a password

x at random, publishes the hash value $y = h(x)$ of x, and later wants to prove to a verifier that she *knows* a preimage of y under h, i.e., a w such that $h(w) = y$.

A natural attempt to achieve this is for Alice to play the role of the prover in a succinct argument for circuit-satisfiability, applied to a circuit $\mathcal{C}$ that takes an input w and outputs $y = h(w)$. However, if the succinct argument only satisfies standard soundness, then Alice's ability to produce a convincing proof will merely guarantee the *existence* of a witness w such that $y = h(w)$. If h is surjective, such a witness w will *always* exist for *any* y. Hence, it is totally uninteresting for Alice to establish the mere existence of such a pre-image w—the trivial proof system in which the verifier always accepts would satisfy the same property.

If the argument for circuit satisfiability satisfies a strengthened security notion called *knowledge-soundness*, then it will in fact guarantee that Alice *knows* the witness w. What does it mean for Alice to prove that she knows a preimage of y under h? The notion of knowledge-soundness posits the following answer. If Alice convinces a verifier to accept her proof with non-negligible probability, then there should be a polynomial time algorithm $\mathcal{E}$ that, if given the ability to repeatedly interact with Alice, is able to output a preimage w of y under h with non-negligible probability. $\mathcal{E}$ is called an *extractor* algorithm. The idea of this definition is that, since $\mathcal{E}$ is efficient, it can't know anything more than Alice does (i.e., anything $\mathcal{E}$ can compute efficiently by interacting with Alice, Alice could compute efficiently on her own, by simulating $\mathcal{E}$'s interaction with herself). Hence, since $\mathcal{E}$ can efficiently find w by interacting with Alice, then Alice must know w. One may think of $\mathcal{E}$ as "efficiently pulling w out of Alice's head".[12, 13]

[12]The interested reader is directed to [130, Section 4.7] for a detailed discussion of how to formalize knowledge-soundness.

[13]The reader may initially suspect that any proof of knowledge cannot be zero-knowledge: if it is possible to "pull a witness w out of the prover's head", doesn't this mean that the proof system reveals the witness to the verifier, grossly violating zero-knowledge? The answer is no. This is because it is not the proof system *verifier* that can extract w from the *proof*, but rather an extractor algorithm $\mathcal{E}$ that can extract w from the *prover*. This means that $\mathcal{E}$ can do things that the verifier cannot. For example, if the proof system is interactive, then $\mathcal{E}$ can run the proof system once

As explained below, the argument system for arithmetic circuit satisfiability in this section (obtained by combining the GKR interactive proof with a commitment c to the multilinear polynomial $\widetilde{w}$) is in fact an argument of knowledge.

Extractable Polynomial Commitments. *Extractability* of a polynomial commitment scheme is a stronger property than mere binding. Roughly, extractability is to binding as knowledge-soundness is to standard soundness. It guarantees that for any efficient prover capable of passing all of the checks performed in the commit and reveal phase of the scheme with non-negligible probability, the prover must actually "know" a polynomial p of the claimed degree that explains its answers to all evaluation queries. That is, if the prover can successfully answer evaluation query z with value v, then $p(z) = v$.

That is, binding of a polynomial commitment scheme guarantees that there is some polynomial p of the appropriate degree that "explains" all of the evaluations that the prover can open the commitment to, but a priori it is possible that the prover itself doesn't actually know what that the polynomial is. Extractability guarantees that in fact the prover does know p. (Section 14.1 later in this monograph contains an example of a polynomial commitment scheme that is binding, but is not extractable).

In more detail, extractability of a polynomial commitment scheme guarantees that for every "efficient committer adversary $\mathcal{A}$" that takes as input the public parameters of the commitment scheme and a degree bound D and outputs a polynomial commitment c, there is an efficient

to completion to see what messages the prover $\mathcal{P}$ sends over the course of the protocol, and then "rewind" $\mathcal{P}$ until just before $\mathcal{P}$ receives the verifier's final challenge, and "restart" $\mathcal{P}$ with a fresh random challenge from the verifier to see how $\mathcal{P}$'s response changes. In contrast, the proof system verifier $\mathcal{V}$ only gets to run the protocol once. In particular, $\mathcal{V}$ does not have the ability to rewind $\mathcal{P}$ and restart it with a new verifier challenge. As another example, if the proof system is non-interactive and operates in the random oracle model, then the extractor algorithm can "watch" all of the queries that $\mathcal{P}$ makes to the random oracle while computing the proof, and try to use those queries to identify a witness. In contrast, the proof system verifier just sees the resulting proof, not the random oracle queries $\mathcal{P}$ made en route to computing the proof. See Remark 12.1 in Section 12.2.1 and Section 12.2.3 for additional discussion and examples.

algorithm $\mathcal{E}'$ (which depends on $\mathcal{A}$) that produces a degree-D polynomial p explaining all of $\mathcal{A}$'s answers to evaluation queries in the sense above. Since $\mathcal{E}'$ is efficient, it cannot know anything more than $\mathcal{A}$ does (since $\mathcal{A}$ can afford to run $\mathcal{E}'$), and $\mathcal{E}'$ clearly knows p by virtue of outputting it.[14] This captures the intuition that $\mathcal{A}$ knows a polynomial p that it is using to answer evaluation queries.

The (relaxed) polynomial commitment scheme described in Section 7.3.2.2 is extractable (we justify this assertion in Section 9.2.1).

The extractability guarantee of the polynomial commitment scheme enables one to take any efficient prover $\mathcal{P}^*$ for the argument system that convinces the argument system verifier to accept with non-negligible probability ε, and extract from $\mathcal{P}^*$ a witness w and prover strategy $\mathcal{P}$ that convinces the verifier within the GKR protocol that $\mathcal{C}(x, w) = y$. Details follow.

Notation. In the remainder of the analysis, we use the following notation to identify various parties.

- $\mathcal{V}$ denotes the prescribed GKR verifier.
- $\mathcal{P}$ represents "a successful GKR prover" (either the prescribed one, or another successful proving procedure, i.e., that convinces $\mathcal{V}$ to accept with non-negligible probability).
- $\mathcal{V}'$ and $\mathcal{P}'$ represent the prescribed succinct-argument verifier and prover.
- $\mathcal{P}^*$ represents a generic (potentially malicious) succinct-argument prover.

Recap of the succinct argument system. Recall that the argument-system prover first sends a commitment c to a multilinear polynomial p

[14] Calling $\mathcal{E}'$ an extractor might initially be puzzling because we have not stated that $\mathcal{E}'$ operates by repeatedly interacting with $\mathcal{A}$ to "pull the polynomial p out of its head". But the extractors for polynomial commitment schemes that we give in this survey actually do work this way. That is, an efficient procedure is given to pull p out of $\mathcal{A}$'s head, and since $\mathcal{A}$ is efficient, the entire procedure, including any computation done "inside" calls to $\mathcal{A}$, is itself efficient, yielding the desired algorithm $\mathcal{E}'$.

claimed to extend a witness w such that $\mathcal{C}(x, w) = y$. After receiving c, the argument system verifier $\mathcal{V}'$ acts identically to the GKR verifier $\mathcal{V}$, i.e., $\mathcal{V}'$ simulates $\mathcal{V}$ and copies its behavior ($\mathcal{V}'$ can do this despite not knowing w, because the GKR verifier $\mathcal{V}$ does not need to know anything about w until the very end of the GKR protocol). Similarly, the honest argument-system prover $\mathcal{P}'$ acts identically to the honest GKR prover for the claim that $\mathcal{C}(x, w) = y$.

At the very end of the GKR protocol, the GKR verifier $\mathcal{V}$ being simulated by $\mathcal{V}'$ does need to evaluate the multilinear extension $\tilde{w}$ of w at a random point r in order to make its final accept/reject decision. $\mathcal{V}'$ obtains this evaluation using the evaluation procedure of the polynomial commitment scheme applied to the commitment c, and outputs whatever accept/reject decision $\mathcal{V}$ would output given the evaluation $p(r)$ obtained from the commitment scheme.

Knowledge-soundness of the argument. Now suppose that $\mathcal{P}^*$ is a polynomial time, but possibly malicious argument-system prover strategy that convinces the argument-system verifier $\mathcal{V}'$ to accept with some non-negligible probability ε. To establish knowledge-soundness of the argument system, we need to explain that there is an efficient extraction procedure $\mathcal{E}$ that can pull out of $\mathcal{P}^*$ a witness w^* such that $\mathcal{C}(x, w^*) = y$.

The extractability of the polynomial commitment scheme implies that we can efficiently extract a pre-image p of the commitment c sent by $\mathcal{P}^*$ at the start of the argument, i.e., p is a multilinear polynomial that opens to all the same values as c. $\mathcal{E}$ sets w^* to be the witness that p extends, i.e., w^* is the set of all evaluations of p at inputs in the Boolean hypercube, $\{0, 1\}^{\log |w|}$.

We still need to explain that w^* satisfies $\mathcal{C}(x, w^*) = y$. To do this, we construct a GKR prover strategy $\mathcal{P}$ that convinces the GKR verifier $\mathcal{V}$ to accept the claim that $\mathcal{C}(x, w^*) = y$ with probability ε. The soundness of the GKR protocol then implies that indeed $\mathcal{C}(x, w^*) = y$.

$\mathcal{P}$ simply simulates $\mathcal{P}^*$ starting from right after $\mathcal{P}^*$ sent the commitment c. That is, in every round i of the GKR protocol, $\mathcal{P}$ sends to $\mathcal{V}$ the message m_i that $\mathcal{P}^*$ would send in that round of the argument system. The GKR verifier $\mathcal{V}$ will reply to m_i with a response r_i, and

$\mathcal{P}$ then continues simulating $\mathcal{P}^*$ into the next round, using r_i as the response of the argument-system verifier $\mathcal{V}^*$ to m_i.

By construction, $\mathcal{P}$ convinces the GKR verifier $\mathcal{V}$ to accept the claim that $\mathcal{C}(x, w^*) = y$ with exactly the same probability that $\mathcal{P}^*$ convinces the argument-system verifier $\mathcal{V}^*$ to accept, namely ϵ. This concludes the proof.

Because the succinct argument of this section is in fact a public coin argument of knowledge, combining it with the Fiat-Shamir transformation yields our first succinct non-interactive argument of knowledge, or SNARK. This SNARK is publicly verifiable, and unconditionally secure in the random oracle model (see Section 9.2.1 for details).