A Survey of Pairing-Based SNARKs in Decentralized Finance Systems

Isaac Thomas[1]

Computer Science & Engineering, UC San Diego
isthomas@ucsd.edu

Abstract. Succinct non-interactive arguments of knowledge (SNARKs) have seen increasing adoption in decentralized finance (DeFi) systems, where scalability and privacy preservation are of high concern in verifying computation performed on blockchain networks. In particular, SNARKs using pairing-based cryptography are widely used in blockchain infrastructure/applications due to their small proof sizes, fast verification, and easily endowed zero-knowledge properties. In this survey, we review advancements in pairing-based SNARKs relevant to the performance, flexibility and confidentiality of blockchain networks and their hosted applications. Along with fundamental SNARK properties and early work, we discuss alternate characterizations of the complexity class NP, cryptographic primitives, and optimizations modern pairing-based SNARKs collectively employ to verify payments and execution paths through on-chain programs; and we weigh concrete security/performance/privacy tradeoffs emerging from such design choices. Finally, we discuss applications and future SNARK research avenues relevant to verifying on-chain computation.

1 Introduction

1.1 What is a SNARK?

Loosely speaking, a succinct non-interactive argument of knowledge (SNARK) is a short, one-shot proof that one *knows* some "witness" or value w satisfying a statement of the form "x is in the language $L \in \text{NP}$." A *prover* $\mathcal{P}$ would send such a proof to a *verifier* $\mathcal{V}$, which checks the proof against the statement to be proven and either accepts the proof or rejects it. Unlike traditional arguments, arguments *of knowledge* require the prover to show that they know the right "evidence" for the statement they are trying to prove; for instance, they should not be able to mix and match valid statements and valid witnesses which are not actually related to each other. Ultimately, the proof should convince the verifier that the prover knows the witness, not just that the statement being proven is true. To accommodate this stronger requirement, SNARKs are often defined in terms of an efficiently decidable binary relation $\mathcal{R}$ connecting statements of membership in an NP language (so-called "NP statements") with the correct "supporting evidence" (witness); this relation is then used to define the language L, thereby capturing all valid statements for which some corresponding evidence exists.

Prior to giving a more formal SNARK definition, we introduce some notation/conventions used throughout this paper. We let F_p be finite field of prime modulus p. We let $\mathbb{G}_i$ denote the multiplicative subgroup over some elliptic curve, where the multiplicative operation in question is an elliptic curve scalar multiplication. We denote by λ the security parameter controlling field/group element size and other aspects related to cryptographic hardness. Where it is not explicitly specified, assume λ is implicitly passed to algorithms mentioned. We now formally define a SNARK.

Definition 1 (Succinct Non-Interactive Argument of Knowledge). For $m, n \in \mathbb{N}$ and prime p, let $\mathcal{R} \subseteq \mathbb{F}_p^m \times \mathbb{F}_p^n$ be an NP relation (i.e., decidable in deterministic polynomial time and $|w| = \mathsf{poly}(|x|)\ \forall (x, w) \in \mathcal{R}$). Let $\lambda \in \mathbb{N}$ denote the security parameter. Define the language $L_{\mathcal{R}} = \{x \mid \exists w \text{ s.t. } (x, w) \in \mathcal{R}\}$. A succinct non-interactive argument of knowledge (SNARK) for $\mathcal{R}$ is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ where:

- $\mathsf{Setup}(1^\lambda, \mathcal{R}) \rightarrow \sigma$ outputs a common reference string σ
- $\mathsf{Prove}(\sigma, \mathsf{x}, \mathsf{w}) \rightarrow \pi$ outputs a proof π
- $\mathsf{Verify}(\sigma, \mathsf{x}, \pi) \rightarrow \{0, 1\}$ outputs accept (1) or reject (0)

and $\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}$ satisfy the following properties:

- **Completeness:** For all $(x, w) \in \mathcal{R}$, if $\sigma \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$ and $\pi \leftarrow \mathsf{Prove}(\sigma, \mathsf{x}, \mathsf{w})$, then $\Pr[\mathsf{Verify}(\sigma, \mathsf{x}, \pi) = 1] \geq 1 - \mathsf{negl}(\lambda)$.

- **Computational Knowledge Soundness:** For any PPT adversary $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}$ such that:

$$\Pr\begin{bmatrix}\sigma \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}); \\ (x, \pi) \leftarrow \mathcal{A}(\sigma); \\ w \leftarrow \mathcal{E}^{\mathcal{A}}(\sigma) \\ : \mathsf{Verify}(\sigma, x, \pi) = 1 \wedge (x, w) \notin \mathcal{R}\end{bmatrix} \leq \mathsf{negl}(\lambda)$$

  where $\mathcal{E}^{\mathcal{A}}$ indicates that $\mathcal{E}$ has black-box access to $\mathcal{A}$ (cannot see $\mathcal{A}$'s internal state).
- **Succinctness:** For any $(x, w) \in \mathcal{R}$ where $|x| = n$:
  - Prover complexity is $O(n \cdot \mathsf{poly}(\log n))$ (quasilinear in statement size)
  - The proof size $|\pi| = O(\mathsf{poly}(\log n))$ (sublinear in statement size)
  - Verifier complexity is $O(\mathsf{poly}(\log n))$ (sublinear in statement size)
- **Non-interactivity:** After setup, the prover sends exactly one message to the verifier.
- **Computational Zero-Knowledge (optional):** There exists a PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, any $(x, w) \in \mathcal{R}$, the following distributions are computationally indistinguishable (denoted by $\approx_c$):

$$\left\{\begin{matrix}(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}); \\ \pi \leftarrow \mathsf{Prove}(\sigma, \mathsf{x}, \mathsf{w}) : (\sigma, \mathsf{x}, \pi)\end{matrix}\right\} \approx_c \left\{\begin{matrix}(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}); \\ \pi \leftarrow \mathcal{S}(\sigma, \tau, x) : (\sigma, x, \pi)\end{matrix}\right\}$$

For example, L could be the set of satisfiable boolean circuits. Here, the statement is "C is in L" where C is the circuit in question; the witness w would be the satisfying assignment to C's inputs, potentially accompanied by intermediate wire values from "executing" C. In a blockchain setting, L could be the set of tuples (P, z) where P is an on-chain program (smart contract) and z is the claimed output of running P on some inputs. The witness could contain inputs x, y and potentially other information needed to verify the output of the execution path through P.

The notions of non-interactivity, computational knowledge soundness, and computational zero-knowledge warrant explanation. First, an interactive proof can be turned non-interactive via the Fiat-Shamir transform. In particular, the prover foregoes interaction with the verifier and instead computes the sequence of challenges according to public values and the elements generated during proof generation. In the previous blockchain-related example, suppose the prover sends a commitment c_1 to some proof elements. The verifier's first challenge would be computed by the prover as $\alpha = h(P||z||c_1)$ for instance. Since α would be considered a message traditionally sent by the verifier, the next challenge would be computed as $h(P||z||c_1||\alpha||r_1)$, where r_1 is the prover's response to the challenge α. Since the verifier knows the public values used to generate the challenge sequence from the proof details, it can verify that the prover generated the challenges correctly. One can think of this as follows: instead of the verifier sending a "question“ to ask the prover, receiving an answer, and repeating this cycle, the prover instead picks the sequence questions the verifier should ask, answers them, then puts the questions and their answers in a single "report" and sends it to the verifier. The verifier then verifies that the right questions were picked, and that the answers to those questions are correct. This scheme has been proven to preserve completeness and soundness properties in the Random Oracle model [BR93], where all parties involved have access to a truly random function they can query as an oracle (for hashing, among other things).

In this non-interactive setting and others, "computational" implies the adversary cannot do anything that takes asymptotically more than polynomial time. Constrast this with "statistical“ knowledge soundness, which would imply that even a computationally unbounded adversary would have at most a small probability of breaking the aforementioned knowledge soundness requirements. The same logic applies to the computational zero-knowledge property, which says it is intractable to distinguish between the distributions of the real proof and the fake, simulated one (would be an entire transcript of messages in the interactive setting). In this vein, it is important to note that the simulator $\mathcal{S}$, who has the "trapdoor" value τ used to generate the preprocessing, *does not know a valid witness.* If one cannot tell these proofs apart in polynomial time, then this corresponds to the proof not leaking any information that would allow doing so using witness information. "Statistical" zero-knowledge implies that even a computationally unbounded entity cannot distinguish the real and fake proofs except with small probability. Though the statistical properties are stronger, the computational variants enable integration of cryptographic primitives to form SNARKs, which in turn enable fast verification in real blockchain systems. Nonetheless, the statistical properties are a noteworthy part of a bigger theoretical cornerstone discussed in following sections.

1.2 Early work

SNARKs are preceded by foundational work in complexity theory, probabilistic proofs, and cryptography. First was the formalization of interactive proofs by Goldwasser et al. [GMR19], which established a theoretical framework by which a prover $\mathcal{P}$ exchanges a finite number of messages during *interaction* with a verifier

$\mathcal{V}$, which issues *random challenges* to $\mathcal{P}$ in order to verify their claim. They also introduced the notion of zero-knowledge via which a proof system leaks no other information beyond the validity of the claim being verified. Arora et al. [AS98, ALM$^+$98] later drew the seminal equivalence between NP and PCP$[O(\log n), O(1)]$, yielding a powerful characterization of NP as the set of languages with PCP verifiers needing logarithmic randomness bits and *constant queries*, where n is the statement size. In a similar vein, Kilian's construction [Kil92] realized a method with succinct verification using merkle tree commitments to probabilistically checkable proofs (PCPs), and Micali later showed how to make his argument non-interactive [Mic94] using the aforementioned Fiat-Shamir heuristic [FS87]. Modern approaches avoid the undesirable proof construction overhead associated with PCPs and opt for algebraic characterizations of NP which admit means to efficient polynomial equality/divisibility testing. Additionally integrating "compiled" interaction and suitable cryptographic commitment schemes brings us closer to the practical SNARK schemes used in real-world blockchain systems.

As seen in Kilian's work, along this path to practically succinct argument systems lies an interesting avenue: considering only computationally bounded adversaries. Along with polynomial checking techniques, one can now use cryptographic schemes to produce small encrypted proof elements and quickly verify relationships between them while ensuring the intractability of breaking soundness. Though this requires stronger security assumptions (i.e. discrete log) than in other hashing-based methods [BSBHR19, AHIV17, WTS$^+$18, BSCR$^+$19], it enables fast verification when compounded with distinguishability properties of polynomials being checked. Within and across denominations of SNARKs, such tradeoffs between performance, security, and flexibility are recurring points of comparison; and they become more visible given the components that are stitched together to form a SNARK.

2 SNARK Components

2.1 A general "workflow" for SNARKs

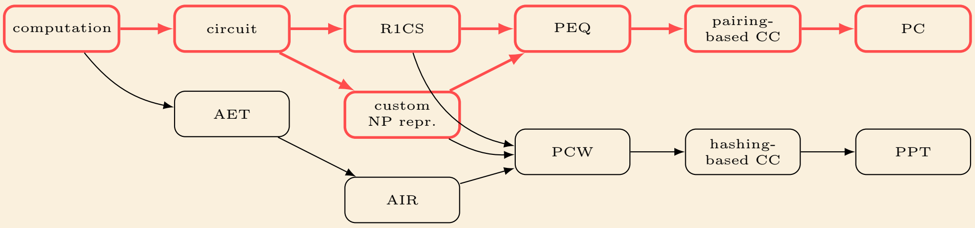

Fig. 1. A (non-exhaustive) general workflow describing how SNARKs reduce checking computation to checking polynomials. The highlighted path shows the course pairing-based approaches largely take take. Acronyms: AET = algebraic execution trace, AIR = algebraic intermediate representation, CC = cryptographic compilation, PCW = polynomial codeword, PEQ = polynomial equality, PC = pairing check, PPT = polynomial proximity testing, R1CS = rank-1 constraint system.

Modern SNARKs largely use the "workflow" depicted in figure 1 to prove/verify computation. Here there are two courses; one reduces to checking polynomial relationships using pairing-based cryptography (left), and the other reduces to checking proximity to a Reed-Solomon code via combination of coding theoretic results and merkle trees. In an interactive setting, the "checks" or "testing" steps might involve randomized interaction with the verifier; but many interactive-proof-based methods use Fiat-Shamir to have the prover simulate such interaction, which the verifier can check. To balance coverage, depth, and accessibility, this paper focuses on approaches involving pairing-based cryptography (highlighted paths). We leave the surveying of methods using hashing-based cryptography as future work, but we note that like pairing-based methods, their effectiveness stems from how the NP computation being verified is represented.

2.2 Representing the complexity class NP

How SNARKs ‘'frame" the computation to verify directly influences how synergistic each method is with algebraic objects like polynomials and the cryptographic methods applied to them. This in turn affects the proof size and prover/verifier complexity, all of which are highly relevant to blockchain infrastructure/applications using SNARKs. Clearly, one must use an NP-complete language compatible with other aspects of the method in question. The most immediate choice of representation is boolean circuit satisfiability; in light of the use of polynomials, a reasonable sequel is *arithmetic* circuit satisfiability, in which the circuit of importance uses addition gates in place of OR gates, and multiplication gates in place of AND gates. Wires can also take on values in some prime field F_p rather than just 0 or 1. Another reasonable step forward is via more direct polynomial representations of NP discussed below.

Definition 2 (Rank-1 Constraint System (R1CS)). Let $m, n \in \mathbb{N}$. A rank-1 constraint system (R1CS) consists of the matrices $L, R, O \in \mathbb{F}_p^{m \times (n+1)}$. A vector $w \in \mathbb{F}_p^{n+1}$ with $w_0 = 1$ satisfies the R1CS if:

$$Ow = (Lw) \circ (Rw) \tag{1}$$

where $\circ$ denotes the Hadamard (element-wise) product.

Here, the second dimension is increased to account for constant values in the constraint system. We now loosely describe how to reduce an instance of arithmetic circuit-SAT to R1CS-SAT. First, preprocess the arithmetic circuit so that addition gates become sums of values along multiplication gate input wires. Then, one can view row i of L as a "selector row" for the values in w that, when linearly combined via row L_i, form the left input to multiplication gate i. The same logic applies for R and O. Populating L, R, O accordingly yields the above relationship between three matrix-vector products – in other words, three rank-1 matrices (hence "rank-1"). In some sense, this consolidates the equations in the constraint system used by Bootle et al. [BCC+16], which uses two separate equations for multiplication gate constraints and linear constraints. It is also reducible to yet another important representation of NP based on polynomial equality checking, defined below.

Definition 3 (Regular Quadratic Arithmetic Program (QAP)). Let $m, n \in \mathbb{N}$. A regular Quadratic Arithmetic Program Q over a field $\mathbb{F}_p$ consists of the following:

- The target polynomial $t(x) = \prod_{i=1}^{m}(x - i)$
- The polynomials $\{u_i(x), v_i(x), w_i(x)\}_{i=0}^{n}$

A vector $c = (c_0 \dots c_n) \in \mathbb{F}_p^{n+1}$ with $c_0 = 1$ satisfies the QAP Q if and only if there exists a polynomial $h(x)$ such that:

$$p(x) = \left(\sum_{i=0}^{n} c_i u_i(x)\right) \cdot \left(\sum_{i=0}^{n} c_i v_i(x)\right) - \left(\sum_{i=0}^{n} c_i w_i(x)\right) = h(x)t(x) = 0 \mod t(x) \tag{2}$$

Again, we loosely describe one reduction from arithmetic circuit-SAT to regular QAP-SAT. First preprocess the gate as described before. Then label each of the multiplication gates 1 through m. Then reduce to R1CS-SAT as described before. Now, define $t(x)$ as mentioned, which forms the polynomial that vanishes whenever $x \in \{1 \dots m\}$ (is a gate identifier). Then for $j \in \{0 \dots n\}$ define each $u_j(x), v_j(x), w_j(x)$ such that $u_j(i) = L_{ij}, v_j(i) = R_{ij}, w_j(i) = O_{ij}$ for all $i \in \{1 \dots m\}$. Evaluating $p(i)$ then effectively checks whether the constraint for gate i is satisfied using the defined $u_j(x), v_j(x), w_j(x)$. In other words, it evaluates $(L_i \cdot c)(R_i \cdot c) - (O_i \cdot c)$, which should be zero for a satisfying assignment and nonzero otherwise. The polynomials include a constant term ($j = 0$), and we typically set $c_0 = 1$ to enable constants in the polynomials analogous to the R1CS-SAT instance. The constants involved in the gates represented by the R1CS-SAT instance would be in the "0th column" of each matrix.

Though not exhaustive by any means, these two representations of NP already yield a a connection between the computation being verified, arithmetic circuit satisfiability, and an instance of polynomial equality/divisibility checking. From this point, the polynomial check can be verified in a hidden fashion using pairing-based cryptography. This idea is used by virtually all pairing-based methods, with modifications to the constraint system and the polynomial relationship being checked.

2.3 Polynomial Distinguishability

As mentioned, modern SNARKs reduce proving/verifying statements about computation to polynomial identity/property testing via suitable characterizations of NP. The use of polynomials over other mathematical objects is justified by how little information one needs to distinguish between two polynomials, which is a consequence of the following lemma.

Lemma 4 (Schwarz-Zippel lemma). Let $f(x_1, x_2, \dots x_n) \in R[x_1 \dots x_n]$ be a nonzero polynomial in n variables defined over an integral domain $\mathbb{F}^n$. Suppose the element $(a_1, a_2, \dots a_n)$ is selected uniformly at random from a finite subset $S \subset \mathbb{F}^n$. Then

$$\Pr(f(a_1, a_2, \dots a_n) = 0) \leq \frac{\deg(f)}{|S|}$$

where $\deg(f)$ is the maximum sum of the degrees of any term's variables. It immediately follows that if $f = g - h$, then for a randomly sampled point $(a_1, \dots a_n) \in \mathbb{F}$, we have that $\Pr(g(a_1, \dots a_n) = h(a_1, \dots a_n))$ is bounded from above by $\frac{d}{|\mathbb{F}|}$. In other words, g and h will output different values at $(a_1 \dots a_n)$ with high probability if they are not equal, assuming the d and $\mathbb{F}$ are chosen so that $d \ll |\mathbb{F}|$. Thus, given the right choice of degree and field size, a verifier will still be able to distinguish between two unequal polynomials with high probability using a single evaluation point. Succinct argument systems can make use of this to check polynomial equalities at a single point, enjoying small proof size and little probability of spurious equality.

2.4 Pairing-based Cryptography

Pairing-based cryptography [BF01] allows one to check algebraic relationships between quantities "in the exponent" using bilinear functions on arbitrary source group elements (usually from elliptic curve groups). This technique works well with checking polynomial equalities and is frequently used in modern SNARKs as a result. This paper omits details on elliptic curve groups and pairings for brevity; we instead focus on the properties they exhibit which make pairing-based cryptography useful in this setting.

Suppose we have two cyclic elliptic curve groups $\mathbb{G}_1$ and $\mathbb{G}_2$ generated by elements G_1 and G_2, respectively. We denote the scalar multiple of a point $P \in \mathbb{G}_i$ by kP, which is just the same as k additions of P and $k \in \mathbb{F}_s$ where $\mathbb{F}_s$ is the scalar field determining the scalars one can multiply points in $\mathbb{G}_1, \mathbb{G}_2$ by. For a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ where $P_1 \in \mathbb{G}_1$ and $P_2 \in \mathbb{G}_2$, we have the following properties:

- for $a, b \in \mathbb{F}$, $e(aP_1, bP_2) = e(P_1, P_2)^{ab}$
- for $Q_1 \in \mathbb{G}_1$, $e(P_1 + Q_1, P_2) = e(P_1, P_2)e(Q_1, P_2)$
- for $Q_2 \in \mathbb{G}_2$, $e(P_1, P_2 + Q_2) = e(P_1, P_2)e(P_1, Q_2)$
- $e(G_1, G_2)$ generates $\mathbb{G}_t$ (non-degeneracy)

These properties make it easy to check multiplicative relationships. For instance, suppose we have three polynomials $A(x), B(x), C(x)$ and we want to check that $A(\alpha)B(\alpha) = C(\alpha)$ for some input α. Then it follows that

$$\begin{aligned} e(G_1, G_2)^{A(\alpha)B(\alpha)} &= e(G_1, G_2)^{C(\alpha)} \\ \Leftrightarrow e([A(\alpha)]_1, [B(\alpha)]_2) &= e([C(\alpha)]_1, G_2) \end{aligned}$$

where $[A(\alpha)]_1 = A(\alpha)G_1$ (same idea for the other values). So if we can reduce checking circuit satisfiability to checking a polynomial equality at a point, we can send elliptic curve points as proof elements with which the verifier would perform such a pairing check.

2.5 Polynomial commitment schemes (PCS)

A polynomial commitment scheme is a protocol by which a prover claims they know a polynomial $f(x)$ satisfying some relationship, and a verifier checks this claim. A noteworthy example of such a relationship is that $f(y) = z$ for some fixed y, z. We detail a noteworthy polynomial commitment scheme for this exact task known as the Kate-Zaverucha-Goldberg commitment scheme (KZG) [KZG10]. Suppose P wants to prove they know f of degree d such that $f(\beta) = z$. It follows that $f(X) - z$ is divisible by $(X - \beta)$, so we should be able to construct $h(X) = \frac{f(X)-z}{X-\beta}$ since we know f. P commits to f and h by their hidden evaluations on some α agreed upon in advance by the P and the verifier V. In practice, the evaluations are hidden via scalar multiplication by an elliptic curve group generator $g_1 \in \mathbb{G}_1$. This scheme uses two elliptic curve groups for this purpose, so assume the point g_2 generates the EC group $\mathbb{G}_2$. Let $[a]_i = ag_i \in \mathbb{G}_i$. Suppose $\mathcal{P}$ and $\mathcal{V}$ agree on a *trusted setup* containing the hidden terms $\{[1]_1, [\alpha]_1, [\alpha^2]_1, \ldots [\alpha^d]_1, [1]_2, [\alpha]_2\}$. P sends $[f(\alpha)]_1$ and $[h(\alpha)]_2$. V then sends a challenge point γ to which $\mathcal{P}$ responds with $[f(\gamma)]_1$ and $[h(\gamma)]_1$. V then checks that $f(\gamma) - z = h(\gamma)(\gamma - \alpha)$ where h was constructed from the evaluation requirement we wanted to prove that f satisfies. For a bilinear pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$, $\mathcal{V}$ checks that

$$e([f(\gamma) - z]_1, g_2) = e([h(\gamma)]_1, [\gamma]_2 - [\alpha]_2) \tag{3}$$

$$\Leftrightarrow e(g_1, g_2)^{f(\gamma)-z} = e(g_1, g_2)^{h(\gamma)(\gamma-\alpha)} \tag{4}$$

$$\Leftrightarrow f(\gamma) - z = h(\gamma)(\gamma - \alpha) \tag{5}$$

KZG commitment schemes require constant size communication since the polynomials involved can be collapsed to a point without losing virtually any distinguishability. Given their use of elliptic curve groups to meet this end, their security depends on the hardness of computing elliptic curve discrete logarithms (find α given the points g and αg), for which no classical polynomial time algorithm is known [Die11]. The drawbacks of using this scheme include the need to generate a trusted setup and the lack of post-quantum security. Being quite customizable, KZG works with batching, multivariate polynomials, and various optimizations for faster polynomial operations. That said, many methods use different polynomial commitment schemes, some of which avoid pairing-based cryptography entirely. Nonetheless we opt for this example to give an idea of commitment schemes making use of pairings, as this is relevant to many modern pairing-based SNARKs.

3 Circuit-specific pairing-based SNARKs

3.1 The QAP recipe

The seminal PCP theorem results [AS98, ALM$^+$98] yielded a powerful characterization of NP as a set of languages with polynomial-time PCP verifiers only needing to check a constant number of proof bits *regardless of statement size.* However, early work on pairing-based SNARKs by Gennaro et al. [GGPR13] was motivated by the possibility of succinct arguments of knowledge using a representation *more suitable* than PCPs for integration with cryptographic primitives. To this end, Gennaro et al. [GGPR13] coined quadratic span programs (QSP) and quadratic arithmetic programs (QAP). These representations of NP allow one to reduce checking a computation to checking a multiplicative relationship between two polynomials in a way that "computes" the circuit representing the computation. Although the QSP construction is noteworthy, we limit discussion to their QAP-related construction due to the clearer connection to arithmetic circuits and downstream polynomial checking. We recall the strong QAP form here:

$$\underbrace{\left(\sum_{j=1}^{n} a_i v_i(x)\right)}_{\text{left inputs}} \cdot \underbrace{\left(\sum_{j=1}^{n} b_i w_i(x)\right)}_{\text{right inputs}} - \underbrace{\left(\sum_{j=1}^{n} c_i y_i(x)\right)}_{\text{outputs}} = h(x)t(x) \equiv 0 \mod t(x) \tag{6}$$

This representation is compiled into cryptographic proof elements via an additively homomorphic encoding E – namely, a one-to-one function for which addition operations are preserved in the output space, and inversion is difficult. Given an instance of arithmetic circuit satisfiability one is trying to prove/verify, E serves to compile the corresponding QSP/QAP instance checking polynomial divisibility (by $t(x)$) into a proof containing just 9 group elements (non-ZK). We detail the non-zero-knowledge version here for clarity, denoting $E(x)$ by $[x]$. The prover $\mathcal{P}$ computes the quotient polynomial $h(x)$ and sends

$$\begin{aligned}\pi = \Big(&[v_{mid}(s)], [w(s)], [y(s)], [h(s)],\\ &[\alpha v_{mid}(s)], [\alpha w(s)], [\alpha y(s)], [\alpha h(s)],\\ &[\beta_v v_{mid}(s) + \beta_w w(s) + \beta_y y(s)]\Big)\end{aligned}$$

where $\alpha, s, \beta_v, \beta_w, \beta_y \in \mathbb{F}_p$, are secret random preprocessing elements; $v_{mid}(s), w(s), y(s)$ are the witness-weighted combinations of the wiring polynomials in the QAP; and $h(s)$ is the evaluation of the quotient polynomial an honest prover would know. In particular, $v_{mid}(x) = \sum_{k \in \mathcal{I}_{mid}} a_k v_k(x)$ where $\mathcal{I}_{mid}$ are the indices corresponding to private circuit inputs. $w(x), y(x)$ are defined similarly, but over all $k \in \{0 \dots m\}$. The verifier in turn checks 5 equations using the additive properties of E. In particular, the verifier receives the following proof π' *which it does not yet know is valid*, hence the new notation of elements reminiscent of those in π:

$$\pi' = (\pi_{v_{mid}}, \pi_w, \pi_y, \pi_h, \pi_{v'_{mid}}, \pi_{w'}, \pi_{y'}, \pi_{h'}, \pi_{z'})$$

$\mathcal{V}$ then checks 6 pairing equations involving the components of π' (recall the pairing notation from earlier section):

$$\begin{aligned}&e([v_0(s) + v_{in}(s) + v_{mid}(s)]_1, [w_0(s) + w_{in}(s) + w_{mid}(s)]_2) = && (7)\\ &e([t(s)]_1, [h(s)]_2) \cdot e([y_0(s) + y_{in}(s) + y_{mid}(s)]_1, G_2) && (8)\\ &e([v'_{mid}(s)]_1, G_2) = e([v_{mid}(s)]_1, [\alpha]_2) && (9)\\ &e([w'(s)]_1, G_2) = e([w(s)]_1, [\alpha]_2) && (10)\\ &e([y'(s)]_1, G_2) = e([y(s)]_1, [\alpha]_2) && (11)\\ &e([h'(s)]_1, G_2) = e([h(s)]_1, [\alpha]_2) && (12)\\ &e([z'(s)]_1, [\gamma]_2) = e([\beta_v v_{mid}(s) + \beta_w w(s) + \beta_y y(s)]_1, [\gamma]_2) && (13)\end{aligned}$$

where $v'_{mid}(s)$ is encoded in $\pi_{v'_{mid}}$, $w'(s)$ is encoded in $\pi_{w'}$, and so on. The first one checks the QAP divisibility relation. The next four equations are based on knowledge-of-exponent assumptions; they check if $\mathcal{P}$ knows the wiring polynomials used in the proof. The last equation checks that $\mathcal{P}$'s proof used wiring polynomials consistent with those agreed upon in the trusted setup.

3.2 More optimized QAP-based methods

GGPR uses a *strong* QAP for their scheme, which uses a different set of coefficients for each sum in the QAP equation. This mandates a *strengthening step* in which extra constraints between the coefficient sets are materialized, tripling prover work and preprocessing size. Parno et al. [PHGR16] made the simple optimization of using a *regular* QAP, which uses the same set of coefficients a_i for each sum in the QAP expression, contrary to the a_i, b_i, c_i seen in the strong QAP form. This change eliminated the need for the strengthening step without any noteworthy compromises, although it required modifications to proof elements and verification checks that ensure consistent use of coefficients in QAP terms. Another improvement was the use of an asymmetric pairing operation $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ such that $\mathbb{G}_1 \neq \mathbb{G}_2$, which in practice is 3-4x as efficient as its symmetric counterpart ($\mathbb{G}_1 = \mathbb{G}_2$). Imaginably, both of these optimizations appear in the following

Table 1. Comparison of work done by pairing-based SNARKs. n represents the number of circuit gates; $\mathbb{G}_1$ and $\mathbb{G}_2$ represent group elements; $\mathbb{F}$ represents field elements; $\mathbf{P}$ represents pairing operations. In prover/verifier work columns, $\mathbb{G}_i$ and $\mathbb{F}$ refer to elliptic curve group scalar multiplications in $\mathbb{G}_i$ and field element multiplications in $\mathbb{F}$, respectively. An asterisk implies the method is not fully succinct. Where more fine-grained source group information is easily discerned, $\mathbb{G}$ implies the elements / operations could be in either source group. Where more fine-grained information is not available or easily comparable in a standardized manner, we resort to asymptotic terms.

method	CRS size (asymp.)	$\mathcal{P}$ work (asymp.)	proof size	$\mathcal{V}$ work	universal	updatable	assumptions
GGPR13	$O(n)\mathbb{G}$	$O(n)\mathbb{G}$	$9\mathbb{G}$	$14\mathbf{P}$	No	No	q-PKE, q-PDH
PGHR13	$O(n)\mathbb{G}$	$O(n)\mathbb{G}$	$8\mathbb{G}$	$11\mathbf{P}$	No	No	q-PKE, q-PDH
Groth16	$9n\mathbb{G}_1, 3n\mathbb{G}_2$	$n\mathbb{G}_1$	$2\mathbb{G}_1, 1\mathbb{G}_2$	$3\mathbf{P}$	No	No	q-type, GGM
GMKL18	$O(n^2)\mathbb{G}$	$O(n)\mathbb{G}_1$	$2\mathbb{G}_1, 1\mathbb{G}_2$	$5\mathbf{P}$	Yes	Yes	q-type, KOE
MBKM19	$36n\mathbb{G}_1$	$273n\mathbb{G}_1$	$20\mathbb{G}_1, 16\mathbb{F}$	$13\mathbf{P}$	Yes	Yes	AGM
Gab19*	$2n\mathbb{G}_1$	$8n\mathbb{G}_1$	$6\mathbb{G}_1, 4\mathbb{F}$	$5\mathbf{P}$	Yes	Yes	AGM
GWC19	$3n\mathbb{G}_1, 2\mathbb{G}_2$	$11n\mathbb{G}_1$	$7\mathbb{G}_1, 6\mathbb{F}$	$2\mathbf{P}, 16\mathbb{G}_1$	Yes	Yes	AGM
GW21	$9n\mathbb{G}_1, 2\mathbb{G}_2$	$35n\mathbb{G}_1$	$4\mathbb{G}_1, 15\mathbb{F}$	$5\mathbb{G}_1, 2\mathbf{P}$	Yes	Yes	AGM
CHM+19	$(4n+2)\mathbb{G}_1$	$22n\mathbb{G}_1$	$13\mathbb{G}_1, 8\mathbb{F}$	$2\mathbf{P}$	Yes	Yes	AGM

QAP-based approaches.

With these two methods (among others) as a springboard, Groth [Gro16] used stronger security assumptions to optimize proof size, setup size, and verifier complexity. Groth still used the regular QAP representation; but unlike previous methods that compiled each polynomial evaluation into two group elements, he compiled the wiring polynomials to just one (ex. just $w(s)$ instead of $w(s), \alpha w(s)$). This eliminated multiple setup terms, reduced proof size to 3 group elements, and reduced verifier operations to a single pairing check using three pairings. However, this single-element compilation prevents use of knowledge-of-exponent assumptions, so security is argued in the generic group model (GGM) [Sho97] where, loosely speaking, adversaries cannot exploit group specific properties or encodings of group elements. While seemingly strong, such assumptions are not necessarily uncalled for, as Gentry et al. showed that SNARKs for NP languages cannot exist without strong – namely, non-falsifiable – assumptions to begin with [GW11]. Reduction in proof size and verifier complexity could also be regarded as "worth it" on this front. Possessing zero-knowledge properties in addition to these other traits, Groth's work (colloquially known as groth16) unsurprisingly saw large adoption in blockchain systems verifying privacy-preserving homogeneous computation. Applications like Tornado Cash [PSS19] use this method to verify withdrawals from privacy-preserving pools without exposing source-destination address relationships; and UTXO-based privacy-preserving blockchains like Zerocash network [BSCG$^+$14] used groth16 to prove/verify presence of unspent funds in the Merkle tree inherent to the UTXO model.

4 Circuit-independent (universal) pairing-based SNARKs

4.1 Shortcomings of the QAP paradigm

Homogeneous computation aside, circuit-specific pairing-based methods have a couple of noteworthy shortcomings. The first of these as noted by Groth [GKM$^+$18] is that there is no way of ensuring the deployers of the SNARK in question have disposed of the secret randomness used to generate the trusted setup. The second and more important issue is a lack of flexibility due to the use of computation-dependent (non-universal) preprocessing. As an example, we depict the groth16 trusted setup here. As in the QAP definition, the $u_i(x), v_i(x), w_i(x)$ encode the contribution of the i-th witness component to left input, right input, and output of gate x respectively.

$$\begin{aligned}\sigma = \Big(&[\alpha]_1, [\beta]_1, [\gamma]_1, [\delta]_1, \{[x^i]_1\}_{i\in[0,n-1]}, \{[\gamma^{-1}(\beta u_i(x) + \alpha v_i(x) + w_i(x))]_1\}_{i\in[0,\ell]},\\ &\{[\delta^{-1}(\beta u_i(x) + \alpha v_i(x) + w_i(x))]_1\}_{i\in[\ell+1,m]},\\ &\{[\delta^{-1}x^i t(x)]_1\}_{i\in[0,\deg(h)]}, [\beta]_2, [\gamma]_2, [\delta]_2, \{[x^i]_2\}_{i\in[0,n-1]}\Big)\end{aligned}$$

As is the case with other discussed methods, the contributions of the i-th witness component to gate x are "stuck" in the encrypted $u_i(x), v_i(x), w_i(x)$ (not the component's value, but its index). Thus if the circuit wiring changes, these polynomials must change to reflect this. Updating the setup incrementally, however, is not possible for two reasons, either of which is sufficiently limiting. The first is the hidden $u_i(x), v_i(x), w_i(x)$ are interpolated from relationships between witness components and the gates they feed into. Adding a new gate and its associated wire connections would require adding a new data point to each of these polynomials.

Thus, one would have to re-interpolate all of them which is not possible without violating cryptographic assumptions (since they are encrypted) or regenerating the entire setup, which defeats the purpose of updates. Another related reason concerns the setup's combination of these hidden polynomial terms, such as the $[\beta u_i(x) + \alpha v_i(x) + w_i(x)]_1$. Groth [GKM$^+$18] showed these setup terms could be used to extract the the constituent monomials and ultimately break soundness if the setup allowed updates. These observations suggest that the conception of a QAP-based updatable SNARK is unlikely, and that different approaches are necessary. Such approaches should be *universal*, in the sense that they allow proving computation of any structure up to a certain size; and *updatable*, meaning that the trusted setup can be efficiently updated by any party and remains sound if at least one setup contributor is honest. This idea becomes highly relevant when deploying blockchain systems using SNARKs with trusted setups. After one update to the setup (which can be performed by anyone with a corresponding proof), users do not have to worry about the network being subverted by its deployers or other adversaries.

As they stray from the QAP-based paradigm which seems to be "maxed out", the following methods use a more diverse set of arithmetizations and polynomial checking to fulfill these universality/updatability related goals. They also largely build upon the simple idea of shifting circuit wiring information into the proof itself; this avoids aforementioned consequences of keeping it in the preprocessing, but at the expense of proof size. Consequently, such design choices yield the most notable points of comparison. For this reason, our discussion of security is significantly more sparse in the following sections; that said, the most noteworthy universal methods prove security in the algebraic group model (AGM) [FKL18], in which (loosely speaking) adversarial algorithms can exploit group-specific structure (weaker security assumptions than GGM).

4.2 Road to a practical universal pairing-Based SNARK

To meet universality/updatability-related ends, Groth [GKM$^+$18] produced a universal and updatable scheme using a multivariate polynomial encoding of QAP elements. Although this scheme had constant proof size, constant verification complexity, and a linear-time procedure by which a linear-size circuit-specific setup could be produced as needed, true universality required quadratically many terms w.r.t circuit size. Furthermore, SRS updates would take a quadratic number of group exponentiations and update verification a linear number of pairing operations. For circuits with millions of gates this is impractical. However, this attempt hints that a SNARK with linear-size universal & efficiently updatable setup could be attainable if the setup terms are univariate (and monomial).

Maller et al. achieved this with Sonic, which draws inspiration from techniques of Bootle et al. [BCC$^+$16] reducing circuit satisfiability to checking Laurent polynomials encoding a Hadamard matrix product (models mul. gate operations) paired with a linear constraint system (models wire connections & addition gates). They also use a permutation argument inspired by Bootle et al. [BG12] to enforce correct copying of values across linked wires. Though significant in its own right, Sonic suffers from large constants in the proof construction complexity. A likely cause is the attempt to accommodate for n-fan-in circuits whose gates can accept arbitrarily many inputs. While a reasonable generalization, this allows any given linear constraint the ability to use arbitrarily many witness inputs, requiring the use of entire witness and selector vectors for each constraint. It also causes a bloated permutation argument since a given gate may require arbitrarily many copy constraints between its own inputs and outputs from preceding gates that feed into it.

Gabizon [Gab19] attempted to optimize Sonic via a sum-checking argument with nonnegative-power Laurent polynomials, reducing setup size but ultimately not reaching full succinctness. Straying from Sonic-style methods, Gabizon et al. [GWC19] addressed prior issues with a simplifying assumption that may not catch one's eye, but turns out to have important performance implications: simply let each circuit gate take two inputs. The impact of this is twofold. Firstly, it enables a much simpler constraint representation of arithmetic circuit-SAT, where for an n gate circuit we have the following constraint for gate i. Here a_i, b_i, c_i are the indices in $\mathbf{x}$ corresponding to the left input, right input, and output of the i-th gate, respectively; and $\mathbf{q_L}, \mathbf{q_R}, \mathbf{q_O}, \mathbf{q_M}, \mathbf{q_C}$, are "selector vectors" determining which gate-related values partake in the constraint. This allows one to express addition or multiplication gates in the same constraint by setting $(\mathbf{q_L})_i, (\mathbf{q_R})_i, (\mathbf{q_M})_i$ accordingly.

$$(\mathbf{q_L})_i \cdot \mathbf{x}_{(\mathbf{a})_i} + (\mathbf{q_R})_i \cdot \mathbf{x}_{(\mathbf{b})_i} + (\mathbf{q_O})_i \cdot \mathbf{x}_{(\mathbf{c})_i} + (\mathbf{q_M})_i \cdot (\mathbf{x}_{(\mathbf{a})_i} \cdot \mathbf{x}_{(\mathbf{b})_i}) + (\mathbf{q_C})_i = 0 \tag{14}$$

During proof construction, these terms are collected into polynomials $\mathbf{a}(X), \mathbf{b}(X), \mathbf{c}(X) \ldots \mathbf{q_M}(X), \mathbf{q_C}(X)$ and combined into a large equation checking the gate constraints and their consistency with other polynomials. We do not show it here due to its length, but the equation is viewable in section 8 of the PlonK paper. Secondly, the 2-fan-in assumption enables optimizations to the permutation argument inspired by Groth et al. and Maller et al. [BG12, MBKM19]. While Sonic used a product check accounting for the arbitrarily many copy constraints per gate, each product check per gate involves three terms in the numerator and denominator

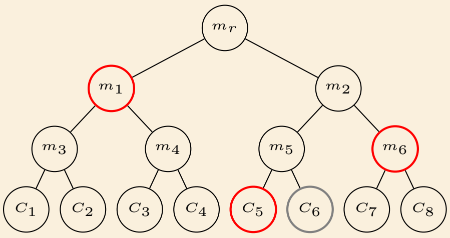

Fig. 2. Merkle tree with authentication path (in red) for deposit commitment C_6 (in gray) used by applications like Tornado cash. Here, C_6 is recursively hashed with its children all the way to the root. If the computed root matches m_r at deposit time, then the user is allowed to withdraw the amount of funds specified in C_6.

each, resulting in a "grand product" used in proof construction. We show the non-zero-knowledge version here, but adding ZK properties requires shifting this equation by adding a "masking" polynomial that still vanishes on the desired domain in order to not corrupt the expression.

$$z(X) = L_1(X)+ \tag{15}$$

$$\sum_{i=1}^{n-1} L_{i+1}(X) \prod_{j=1}^{i} \frac{(w_j + \beta\omega^j + \gamma)(w_{n+j} + \beta k_1 \omega^j + \gamma)(w_{2n+j} + \beta k_2 \omega^j + \gamma)}{(w_j + \beta\sigma^*(j) + \gamma)(w_{n+j} + \beta\sigma^*(n+j) + \gamma)(w_{2n+j} + \beta\sigma^*(2n+j) + \gamma)} \tag{16}$$

This equation is essentially asking the following question over all gates, with random shifts to prevent erroneous cancellations or malicious swapping of wire values: "If I place the wire values for this gate on the top, and the values of the wires that this gate's wire values copy into on the bottom, will the result cancel out to 1?" Here ω is a primitive n-th root of unity, w_j are the wire values, and β, γ are random challenges. The function $\sigma^*(j)$ defines the permutation mapping between wire positions. The constants k_1 and k_2 separate the "identifiers" for these wire values based on the permutation polynomials, which essentially map one coset of roots of unity to another. The function $L_i(X) = \frac{X^n-1}{X-\omega^i}$ is the i-th Lagrange basis polynomial defined over the n-th roots of unity (gives 1 when $X = \omega^i$ and 0 otherwise). Despite its appearance, computing this expression does not need a quadratic number of operations because the products can be accumulated to avoid redundant work. In this same vein, the polynomial check works well with a common optimization to PlonK's invocation of the batched KZG scheme; by representing the hidden monomials in the Lagrange basis, the point evaluations of polynomials can be computed in $O(d)$ operations where d is the degree of the polynomial in question, which is faster than the $O(d \log d)$ complexity for interpolation via inverse FFT & subsequent evaluation. The pairing check performed by the verifier is also "fixed argument" in the second source group, enabling the 2 pairings the verifier performs to be ≈ 30% faster than the traditional pairing [CS10].

The combination of flexibility, intuitive arithmetization, and better efficiency has made PlonK a gateway to other methods increasing expressiveness and performance, with important implications for so-called "zero-knowledge virtual machines" (zkVMs). Gabizon et al. produced FFT-based commitment scheme optimizations to the original PlonK method via fflonk [GW21], improving verifier performance at the expense of increased prover work. Ambrona et al. [GW20b] proposed methods to optimize the constraint system used, as well as optimized circuits for PlonK-based verifiable implementations of the "SNARK-friendly" Poseidon hash function [GKR$^+$21]. For zkVM operations less compatible with SNARK systems, Gabizon et al. proposed Plookup [GW20a], which adapts the PlonK permutation argument to verify that a set of values is present in some predetermined table; this is particularly useful for constraining the correctness of zkVM operations involving nonlinear operations that would make the resulting circuits / constraint system inefficient to verify. A notable example is a "SNARK-unfriendly" hash function like SHA-3 [D$^+$15], which has many nonlinear bit-mixing operations. Variations like halo2 [Dev24] and plonky2 [POL22] combine the PlonK arithmetization with commitment schemes inspired by Bulletproofs [BBB$^+$18] and FRI [BSBHR18] (respectively) to shed the need for a trusted setup at the expense of slower verification.

PlonK-style methods aside, Chiesa et al. [CHM$^+$20] developed a sumcheck-inspired argument over polynomials encoding an indexed sparse R1CS representation. This method is used by the Aleo layer-1 blockchain [Ale24], but has clearly seen markedly less adoption than prior methods. In this same vein, the following sections explore the real-world usage of the methods discussed.

5 Applications

5.1 Application/Infrastructure-Level Privacy Preservation

Traditional blockchain network innerworkings expose transaction details, making privacy preservation non-trivial at the application/infrastructure layers. Applications like Tornado Cash [PSS19] partially solve this problem at the application layer, using groth16 proofs to enable privacy-preserving deposits and withdrawals to different wallets without revealing the link between the two. Withdrawing funds involves proving membership of one's deposit record in a Merkle tree, as shown in figure 2; combined with tight EVM smart contract gas limits, the homogeneous nature of this computation makes groth16 a suitable choice. Blockchains like Zerocash [BSCG$^+$14] (privacy-preserving version of Bitcoin [Nak08]) somewhwat generalize tornado cash's abilities using the unspent transaction output (UTXO) model, where spending funds requires proving knowledge of an "unspent funds" record in a Merkle tree. Early versions of Zerocash used groth16 for this, but the NU5 upgrade of 2022 switched to the universal halo2 proof system [Dev24], which enables a combination of a PlonK-style arithmetization and Bulletproofs-style commitment scheme [BBB$^+$18] requiring no trusted setup at the expense of verifier performance. Beyond such homogeneous computation, non-universal (zk)SNARKs could be used for application-specific privacy-preserving blockchains as well (ex. a blockchain for private trading/lending).

5.2 zkVMs and "Layer-2" Scaling Solutions for Ethereum Network

Layer-2 (L2) blockchains [GGT23] have emerged as a dominant scaling solution to layer-1 (L1) chains like Ethereum [But13]. These alternate blockchains are essentially Ethereum forks with their own application layer and transaction processing; however, they settle batches of transactions directly on Ethereum network. These alternate outlets afford users a similar experience while easing computational load on the Ethereum chain. "Zero-knowledge" virtual machines (zkVMs) [Scr24, Pol24, Suc23, RIS23, Ope25] are a crucial component of this architecture, as they produce succinct proofs of L2 block validity which are later settled by verifier contracts on the L1 chain. zkVMs using PlonK variants [Scr24] have already played a huge role in the functionality of zkVMs powering widely used L2 chains [Scr22]. Further advancements in lookup arguments based on either PlonK or other methods will have huge impacts on transaction processing speed, layer 2 network performance, and ultimately the scalability of Ethereum network itself.

6 Future Research

6.1 Post-quantum algebraically-friendly SNARKs

Though multiple of the SNARKs discussed are practical from a performance standpoint, quantum computers may render them useless from a security standpoint in the near future. As a result, there is a strong effort to develop efficient enough post-quantum SNARKs. To this end, many works [BSBHR19, AHIV17, COS20, Set20, AST24] use hashing-based Reed-Solomon proximity testing methods [BSBHR18], which relies on the hardness of computing hash function collisions. While such approaches are believed to be post-quantum secure, the cryptographic primitives used offer no additively homomorphic structure enabling the recursive composition of smaller proof elements & faster verification we see in pairing-based methods. Albrecht et al. [ACL$^+$22] made some progress in this direction via a lattice-based SNARK with logarithmic-time verification; it derives security from the hardness of the short integer solution (SIS) problem in lattices. Future work that decreases prover/verifier complexity could realize practical SNARKs using additively homomorphic post-quantum encryption, which virtually every blockchain would need in a post-quantum era.

6.2 Automated verification of SNARK methods/applications

Layer 2 blockchains collectively hold tens of billions of dollars [l2b] protected by the integrity of SNARK methods used to verify block validity; if the methods/implementations have soundness bugs, an adversary could prove an invalid state transition for financial gain and steal funds. There are two promising avenues to mitigate this risk. The first is formal verification of SNARKs in mechanized cryptographic models using Lean 4 and other methods [ark24, dMU, KDT24]. Though very promising and easily updatable, there are still many gaps to fill. Furthermore, the periodic introduction of new methods will require constant updates to the relevant types and theorems. Nonetheless, seeking formal guarantees on SNARK behavior could eliminate entire classes of soundness bugs in L2-related implementations, lowering risks of fund theft. The second and more targeted direction concerns automated detection of nondeterministic circuit behavior using symbolic execution and SMT solvers. An example of this is Picus [PCW$^+$23], which has been used to verify parts of the Risc Zero VM [RIS23]. Preliminary work by Chaliasos et al. [CET$^+$24] has found that circuit-related soundness bugs are the most frequent kind of bug by a huge margin, suggesting that building upon this existing work will be important especially as zkVM implementations become more complex. LogUp-based arguments [Hab22] for zkVMs like SP1 [Suc23] based on interacting sub-witnesses would be a necessary target of future work in this domain.

Acknowledgements

This survey was compiled independently, but not without helpful insights from presentation-style content provided by the authors of works discussed/mentioned. In particular, we thank Mary Maller, Dan Boneh, and Jens Groth for their informative auxiliary media content on Sonic, PlonK, and Groth16 respectively.

References

ACL$^+$22. M. R. Albrecht, V. Cini, R. W. Lai, G. Malavolta, and S. A. Thyagarajan. Lattice-based snarks: publicly verifiable, preprocessing, and recursively composable. In *Annual International Cryptology Conference*, pages 102–132. Springer, 2022.

AHIV17. S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 2087–2104, 2017.

Ale24. Aleo. Aleo developer documentation. `https://developer.aleo.org/`, 2024. Accessed: 2025-05-14.

ALM$^+$98. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

ark24. GitHub - Verified-zkEVM/ArkLib: Formally Verified Arguments of Knowledge in Lean — github.com. `https://github.com/Verified-zkEVM/ArkLib`, 2024. [Accessed 08-05-2025].

AS98. S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.

AST24. A. Arun, S. Setty, and J. Thaler. Jolt: Snarks for virtual machines via lookups. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2024.

BBB$^+$18. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

BCC$^+$16. J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 327–357. Springer, 2016.

BF01. D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001.

BG12. S. Bayer and J. Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*, pages 263–280. Springer, 2012.

BR93. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

BSBHR18. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *45th international colloquium on automata, languages, and programming (icalp 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

BSBHR19. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable zero knowledge with no trusted setup. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pages 701–732. Springer, 2019.

BSCG$^+$14. E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014.

BSCR$^+$19. E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. Aurora: Transparent succinct arguments for r1cs. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 103–128. Springer, 2019.

But13. V. Buterin. Ethereum: A next-generation smart contract and decentralized application platform. `https://ethereum.org/en/whitepaper/`, 2013. Accessed: May 2025.

CET$^+$24. S. Chaliasos, J. Ernstberger, D. Theodore, D. Wong, M. Jahanara, and B. Livshits. Sok: What don't we know? understanding security vulnerabilities in snarks, Feb 2024. arXiv:2402.15293v1.

CHM$^+$20. A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 738–768. Springer, 2020.

COS20. A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 769–793. Springer, 2020.

CS10. C. Costello and D. Stebila. Fixed argument pairings. In *Progress in Cryptology–LATINCRYPT 2010: First International Conference on Cryptology and Information Security in Latin America, Puebla, Mexico, August 8-11, 2010, proceedings 1*, pages 92–108. Springer, 2010.

D$^+$15. M. J. Dworkin et al. Sha-3 standard: Permutation-based hash and extendable-output functions. 2015.

Dev24. Z. Developers. Halo2 proofs documentation. `https://zcash.github.io/halo2/`, 2024. Accessed: 2024-04-07.

Die11. C. Diem. On the discrete logarithm problem in elliptic curves. *Compositio Mathematica*, 147(1):75–104, 2011.

dMU. L. de Moura and S. Ullrich. The lean 4 theorem prover and programming language (system description).

FKL18. G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part II 38*, pages 33–62. Springer, 2018.

FS87. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

Gab19. A. Gabizon. Auroralight: Improved prover efficiency and srs size in a sonic-like system. *Cryptology ePrint Archive*, 2019.

GGPR13. R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pages 626–645. Springer, 2013.

GGT23. A. Gangwal, H. R. Gangavalli, and A. Thirupathi. A survey of layer-two blockchain protocols. *Journal of Network and Computer Applications*, 209:103539, 2023.

GKM$^+$18. J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. Updatable and universal common reference strings with applications to zk-snarks. In *Annual International Cryptology Conference*, pages 698–728. Springer, 2018.

GKR$^+$21. L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. Poseidon: A new hash function for {Zero-Knowledge} proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535, 2021.

GMR19. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*, pages 203–225. 2019.

Gro16. J. Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016.

GW11. C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 99–108, 2011.

GW20a. A. Gabizon and Z. J. Williamson. plookup: A simplified polynomial protocol for lookup tables. *Cryptology ePrint Archive*, 2020.

GW20b. A. Gabizon and Z. J. Williamson. Proposal: The turbo-plonk program syntax for specifying snark programs, 2020.

GW21. A. Gabizon and Z. J. Williamson. fflonk: a fast-fourier inspired verifier efficient version of plonk. *Cryptology ePrint Archive*, 2021.

GWC19. A. Gabizon, Z. J. Williamson, and O. Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

Hab22. U. Haböck. Multivariate lookups based on logarithmic derivatives. *Cryptology ePrint Archive*, 2022.

KDT24. C. Kwan, Q. Dao, and J. Thaler. Verifying jolt zkvm lookup semantics. *Cryptology ePrint Archive*, 2024.

Kil92. J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732, 1992.

KZG10. A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.

l2b. L2BEAT - The state of the layer two ecosystem. [Accessed 08-05-2025].

MBKM19. M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2111–2128, 2019.

Mic94. S. Micali. Cs proofs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 436–453. IEEE, 1994.

Nak08. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

Ope25. OpenVM-Org. openvm-org/openvm: A performant and modular zkVM framework built for customization and extensibility., Mar 2025.

PCW+23. S. Pailoor, Y. Chen, F. Wang, C. Rodríguez, J. Van Geffen, J. Morton, M. Chu, B. Gu, Y. Feng, and I. Dillig. Automated detection of Under-Constrained circuits in Zero-Knowledge proofs. *Proc. ACM Program. Lang.*, 7(PLDI), June 2023.

PHGR16. B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016.

POL22. POLYGON. Plonky2: Fast recursive arguments with plonk and fri. `https://github.com/0xPolygonZero/plonky2/blob/136cdd053f2175134cddc61abc587f1862e76921/plonky2/plonky2.pdf`, 2022. Accessed: 2025-05-10.

Pol24. Polygon Labs. zkevm and execution traces- polygon knowledge layer, 2024.

PSS19. A. Pertsev, R. Semenov, and R. Storm. Tornado cash privacy solution. Technical report, Tornado Cash, 2019.

RIS23. RISC Zero. Proof system in detail. Technical report, RISC Zero, 2023. [Accessed 08-05-2025].

Scr22. Scroll. An overview of scroll's architecture, 8 2022.

Scr24. Scroll. zkevm overview, 2024.

Set20. S. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

Sho97. V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16*, pages 256–266. Springer, 1997.

Suc23. Succinct. Introducing SP1, 2023.

WTS+18. R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943. IEEE, 2018.