A Survey of 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 Preliminaries

1.1 What is a SNARK?

Loosely speaking, a succinct non-interactive argument of knowledge is a short, one-shot proof that one *knows* some "witness" or value w satisfying a claim C related to public input(s) x. In practice, C is a claim that $x \in L$, where $L \in$ NP. We give a more formal definition below.

A more formal definition Let the relation $\mathcal{R} = \{(x, w)\} \subseteq \{0,1\}^* \times \{0,1\}^*$ be decidable in deterministic polynomial time, where $|w| = \mathsf{poly}(|x|) \ \forall (x, w) \in \mathcal{R}$. Let $\lambda \in \mathbb{N}$ denote the security parameter. Define the language $L = \{x \mid \exists w \text{ s.t. } (x, w) \in \mathcal{R}\}$. A succinct non-interactive argument of knowledge is the tuple of PPT algorithms (Setup, Prove, Verify) where:

- $\mathsf{Setup}(1^\lambda, \mathcal{R}) \to \sigma$ receives an input security parameter and relation, and outputs a common reference string (CRS) σ containing the "setup" terms the prover and verifier will use to construct/verify proofs
- $\mathsf{Prove}(\sigma, \mathsf{x}, \mathsf{w}) \to \pi$ receives the CRS, string x, and witness w and outputs a proof π
- $\mathsf{Verify}(\sigma, \mathsf{x}, \pi) \to \{0, 1\}$ receives the CRS, string x, and proof π; it outputs 1 (accepts) or 0 (rejects)

and Setup, Prove, Verify have the following properties:

- **completeness:** If $\pi \leftarrow \mathsf{Prove}(\sigma, \mathsf{x}, \mathsf{w})$ and $(x, w) \in \mathcal{R}$ then Verify outputs 1 except with negligible probability.
- **knowledge soundness:** For all possible PPT algorithms Prove$'$, if $\pi' \leftarrow \mathsf{Prove}'(\sigma, \mathsf{x})$ and $\mathsf{Verify}(\sigma, \mathsf{x}, \pi')$ outputs 1 with non-negligible probability, then there exists a PPT extractor $\mathcal{E}$ which, given access to the same inputs as Prove and $\mathcal{P}$'s random tape, outputs w s.t. $(x, w) \in \mathcal{R}$.
- **succinctness:** Let $(x, w) \in \mathcal{R}$ and $\pi \leftarrow \mathsf{Prove}(\sigma, \mathsf{x}, \mathsf{w})$. Setup complexity and prover complexity are quasilinear in $|x|$, $|\pi|$ is sublinear in $|x|$, and verifier complexity is sublinear in $|x|$.
- **non-interactivity:** π is generated without any interaction between $\mathcal{P}$ and $\mathcal{V}$ after setup.

As an example, L could be the set of string encodings of satisfiable boolean circuits. The claim to be proven is that $x \in L$, where x is the encoding of the circuit in question; since L is fixed here, the claim can just be represented by x. Our witness w is our potentially secret "certificate" proving the validity of the claim.

1.2   Early work

Succinct non-interactive arguments of knowledge 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 for which verifiers needing logarithmic randomness bits and *constant queries*, where n is the statement size.

With these results in mind, simply sending the full "certificate" of the given problem's solution is useless as it is likely not space-efficient or zero-knowledge. This begs the following question: given results showing all NP languages have efficient verifiers, do "short enough" or *succinct* arguments of knowledge exist for NP? Kilian's construction [Kil92] found an affirmative answer to this inquiry (for verification at least) using merkle tree commitments to probabilistically checkable proofs (PCPs), and Micali later showed how to make this argument non-interactive [Mic94] using the Fiat-Shamir heuristic [FS87]. Despite these breakthroughs, SNARKs constructed from PCPs were undesirable due to massive overhead incurred in proof construction. Many approaches instead use algebraic characterizations of NP which admit means to efficient polynomial identity/property testing. Additionally integrating "compiled" interaction and suitable cryptographic commitment schemes can reduce prover complexity, proof length, and verifier complexity.

As seen in Kilian's work, along this path to succinctness lies an interesting and practically necessary avenue: considering only computationally bounded adversaries. In doing so, one can now use cryptographic schemes to encrypt proof elements and verify their integrity. This can enable small proof sizes and fast verification while still making it infeasible to break soundness in polynomial time. Pairing-based cryptography, which allows one to check algebraic relationships between quantities "in the exponent" using bilinear functions (behaving like multiplication) on elliptic curve group elements, spurred a sequence of works leading to methods practical enough for real-world blockchain applications. Though faster, all of these methods require stronger security assumptions. In particular, pairing-based methods derive their security from the elliptic curve discrete log assumption, which is stronger than the collision resistance assumptions used in other hashing-based methods like STARKs [BSBHR19]. Within and across denominations of SNARKs, such tradeoffs between performance, security, and flexibility are recurring points of comparison; and where they appear becomes clearer with a picture of which components are stitched together to form a SNARK.

1.3   A general "workflow" for SNARKs

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 methods use Fiat-Shamir to have the prover simulate such interaction, which the verifier can check. The result is an interactive protocol compiled into a succinct non-interactive argument of knowledge (the "compilation" here is different from the compilation mentioned in the graph above). 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.

1.4   Representing the complexity class NP

Crucial to modern SNARKs is their representation of NP. How the computation to verify is "framed" directly influences how synergistic each method is with algebraic objects like polynomials and the cryptographic methods applied to them. Clearly, one must use an NP-complete language compatible with other aspects of the method in question. The most immediate choice of representation is
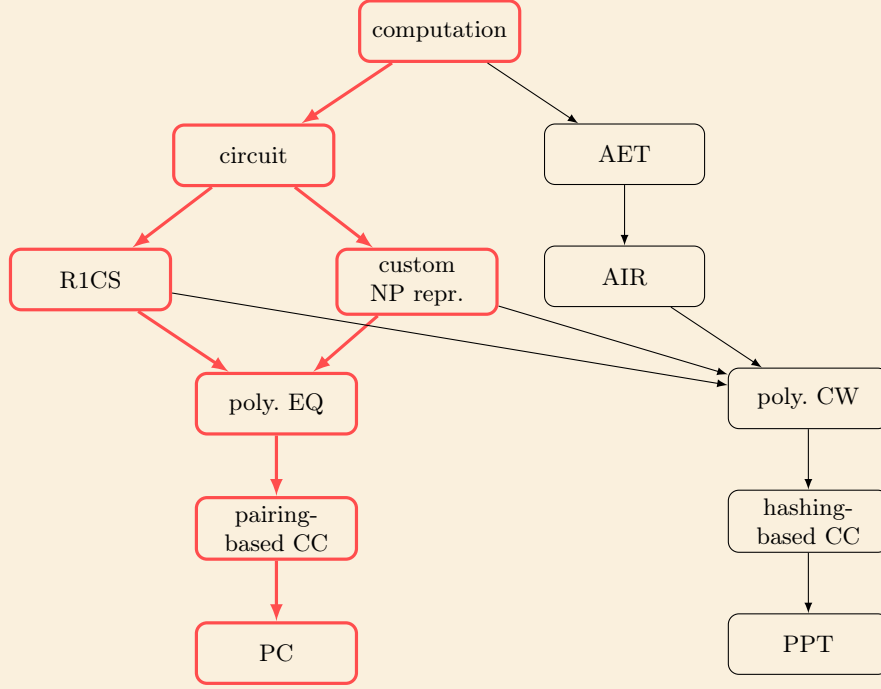
Fig. 1. A general workflow describing how SNARKs convert computations into verifiable statements. The highlighted path shows the course pairing-based approaches take. Acronyms: AET = algebraic execution trace, AIR = algebraic intermediate representation, CC = cryptographic compilation, CW = codewordx, EQ = polynomial equality, PC = pairing check, PPT = polynomial proximity testing, R1CS = rank-1 constraint system.

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. A natural question arises: are there representations of NP that seamlessly connect arithmetic circuit satisfiability with polynomial equality testing? The answer is yes, and we discuss notable examples of this below.

Rank 1 Constraint System (R1CS) Let $m, n \in \mathbb{N}$, and consider an arithmetic circuit with m gates and n input variables. with A rank-1 constraint system [5] (R1CS) consists of a vector $w \in \mathbb{F}_p^n$, the matrices $L, R, O \in \mathbb{F}_p^{m \times n}$ and the relationship

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

A rank-1 constraint system (R1CS) *arithmetizes* the relationships between the left inputs, right inputs, and outputs of each gate, respectively. It assumes that the circuit has been preprocessed so that only multiplication gates remain, with the addition gates instead expressed as sums input to the multiplication gates. Intuitively, row i of L can be seen as a "selector" for the values in w that, when linearly combined via row L_i, form the left input to gate i. The same logic applies for R and O. Applying this logic in aggregate yields the above relationship between a Hadamard product of vectors and the desired output vector - in other words, three rank-1 matrices (hence "rank-1"). This problem is NP-complete with a relatively simple reduction from arithmetic circuit-SAT (follows easily from the loose definition above). One can also view it as a consolidation of the equations in the constraint system used by Bootle et al.[5], which uses two separate equations for multiplication gate constraints and linear constraints.

Quadratic Arithmetic Program (QAP) Connecting the R1CS representation of NP with polynomials involves converting either the rows or the columns of L, R, O into polynomials and creating

some relationship that holds if and only if $Ow = (Lw) \circ (Rw)$. Given that for the i-th gate we have

$$(Oz)_i = \left(\sum_{j=1}^{n} L_{ij} w_j \right) \left(\sum_{j=1}^{n} R_{ij} w_j \right) \tag{2}$$

It could make sense to create a polynomial for each of the n witness variables which evaluate to each L_{ij} given the gate i. This would involve interpolating column j of the matrix L into a polynomial $u_j(x)$. The same intuition applies to the matrices R, O (using $v_j(x), w_j(x)$ respectively) and is the idea behind the quadratic arithmetic program (QAP) introduced by Gennaro et al. [GGPR13]. We focus on the definition of a regular QAP.

Definition 1 (regular QAP). *A regular Quadratic Arithmetic Program Q over a field $\mathbb{F}_p$ comprises:*

- *The target polynomial $t(x) = \prod_{i=1}^{m}(x - i)$, where m is the number of gates*
- *Three sets of polynomials $\{u_i(x)\}_{i=0}^{n}$, $\{v_i(x)\}_{i=0}^{n}$, and $\{w_i(x)\}_{i=0}^{n}$, where $u_i(j) = L_{ji}$, $v_i(j) = R_{ji}$, and $w_i(j) = O_{ji}$*

   Q is satisfied by a witness $c \in \mathbb{F}_p^n$ if and only if:

$$p(x) = \left(\sum_{i=0}^{n} c_i u_i(x) \right) \cdot \left(\sum_{i=0}^{n} c_i v_i(x) \right) - \left(\sum_{i=0}^{n} c_i w_i(x) \right) = h(x)t(x) \equiv 0 \mod t(x) \tag{3}$$

Here $t(x)|p(x)$ is synonymous with $p(x)$ vanishing at all points which are gate identifiers, which will be true if the gate relationship is indeed satisfied by the given inputs, and will not be true otherwise. QAP satisfiability is also NP-complete with intuitive reductions from arithmetic circuit-SAT and R1CS. Gennaro et al. originally coined QAPs as a way to compute arithmetic circuits, but the relationship to R1CS is more noteworthy given the methods to be discussed, hence the slightly altered presentation.

Though not exhaustive by any means, these two representations of NP yield a connection between the computation being verified, arithmetic circuit satisfiability, and an instance of polynomial divisibility testing. From this point, the polynomial divisibility 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.

1.5   Polynomial Properties

As mentioned, modern SNARKs reduce proving/verifying statements about computation to polynomial identity/property testing via suitable characterizations of NP. A natural question about this arises: why polynomials over other mathematical objects? The answer in large part lies in how little information one needs to distinguish between two polynomials, which is a consequence of the following lemma.

Schwarz-Zippel lemma Let $f(x_1, x_2, \ldots x_n) \in R[x_1 \ldots x_n]$ be a nonzero polynomial in n variables defined over an integral domain $\mathbb{F}^n$. Suppose the element $(a_1, a_2, \ldots a_n)$ is selected uniformly at random from a finite subset $S \subset \mathbb{F}^n$. Then $\Pr(f(a_1, a_2, \ldots a_n) = 0) \leq \frac{\deg(f)}{|S|}$ where $\deg(f)$ is the maximum sum of the degrees of any term's variables. It immediately follows that if $f = g - h$, as g and h are distinct since f is not zero; then for a randomly sampled point $(a_1, \ldots a_n)$, the probability that $(g - h)(a_1, \ldots a_n) = 0 \Leftrightarrow g(a_1, \ldots a_n) = h(a_1, \ldots a_n)$ is identically bounded. In other words, g and h will output different values at $(a_1 \ldots a_n)$ with high probability if they are not equal, assuming the d and $\mathbb{F}$ are chosen appropriately $d \ll |F|$. 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.

Schemes relying on this property would yield shorter proofs, since we need a single evaluation point; faster prover complexity, since polynomial evaluation has many known efficient algorithms over both finite fields and the real numbers; and faster verification, since checking a polynomial identity can be done quickly via point evaluation without significant soundness loss. Succinct argument systems can make use of this to check polynomial equalities at a single point with little probability of spurious equality.

1.6   Pairing-based Cryptography

Pairing-based cryptography, which allows one to check algebraic relationships between quantities "in the exponent" using bilinear functions on arbitrary source group elements (usually from elliptic curve groups), is a heavily used cryptographic primitive in modern SNARKs. Elliptic curve groups are preferred here due to small key sizes – determined by the size of the base field the point coordinates are from. There is also a grooup operation over elliptic curve points behaving like addition, in which one computes a sum of points $A + B$ by reflecting over the x-axis the point on both the curve and the line AB (if $A = B$ we use the tangent line through A). We can then define multiplication of a point A by some $s \in \mathbb{F}_s$ in the curve's *scalar field* as summing s copies of A. Here, s can be considered an *elliptic curve discrete logarithm* and is believed to be hard to compute given A and sA; this is the elliptic curve variant of the discrete log assumption. The hardness of this problem yields a scheme for additively homomorphic encryption: given a cyclic multiplicative group $\mathbb{G}$ over some wisely-chosen elliptic curve and its generator G, encrypt x as $xG \in \mathbb{G}$. Pairings can render this scheme *doubly* homomorphic due to the properties of bilinear pairings, which behave like multiplication (linear in each input). This paper abstracts elliptic curve groups and pairings for brevity; we instead focus on the properties they exhibit which make pairing-based cryptography useful in this setting.

More formally, 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 some scalar field. 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 $a, b \in \mathbb{F}$, $e(aP_1, bP_2)e(P_1, P_2)^{ab} = e(P_1, P_2)^{ba} = e(bP_1, aP_2)$
- 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. From bilinearity of e It follows that

$$e(G_1, G_2)^{A(\alpha)B(\alpha)} = e(G_1, G_2)^{C(\alpha)} \tag{4}$$

$$\Leftrightarrow e([A(\alpha)]_1, [B(\alpha)]_2) = e([C(\alpha)]_1, G_2) \tag{5}$$

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 relationship, we can send elliptic curve points as proof elements with which the verifier would perform such a pairing check. The complexity of a naive pairing computation is quasi-quadratic in the target group size, but there are pairing-friendly choices of elliptic curve groups which make this operation faster. Most importantly, the number of pairings to be performed does not depend on the size of the computation being verified.

1.7   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

$$h(X) = \frac{f(X) - z}{X - \beta} \tag{6}$$

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

$$\{[1]_1, [\alpha]_1, [\alpha^2]_1, \dots [\alpha^d]_1, [1]_2, [\alpha]_2\} \tag{7}$$

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

$$e([f(\gamma) - z]_1, g_2) = e([h(\gamma)]_1, [\gamma]_2 - [\alpha]_2) \tag{8}$$

$$\Leftrightarrow e(g_1, g_2)^{f(\gamma)-z} = e(g_1, g_2)^{h(\gamma)(\gamma-\alpha)} \tag{9}$$

$$\Leftrightarrow f(\gamma) - z = h(\gamma)(\gamma - \alpha) \tag{10}$$

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.

There are other general and protocol-specific commitment schemes which check slightly different polynomial relationships than KZG or avoid pairing-based cryptography entirely. But 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.

1.8   Turning interactive protocols non-interactive

Interactive pro, without requiring the original prover to remain online. The Fiat-Shamir heuristic provides an elegant solution to transform interactive protocols into non-interactive arguments.

2   Circuit-specific pairing-based SNARKs

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 view a constant number of proof bits *regardless of proof 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. Several other works [**WHICH ONES**] came close to meeting this end but could not attain sublinear proof size.

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:

$$\underbrace{\left(\sum_{j=1}^{n} a_i v_i(x)\right)}_{\text{left inputs}} \cdot \underbrace{\left(\sum_{j=1}^{n} b_i w_i(x)\right)}_{\text{right inputs}} - \underbrace{\left(\sum_{j=1}^{n} c_i y_i(x)\right)}_{\text{outputs}} = h(x)t(x) \equiv 0 \mod t(x) \tag{11}$$

This idea leverages 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

$$\pi = \Big([v_{mid}(s)], [w(s)], [y(s)], [h(s)], \tag{12}$$

$$[\alpha v_{mid}(s)], [\alpha w(s)], [\alpha y(s)], [\alpha h(s)], \tag{13}$$

$$[\beta_v v_{mid}(s) + \beta_w w(s) + \beta_y y(s)]\Big) \tag{14}$$

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 definedn similarly, but over all $k \in \{0 \ldots m\}$. 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:

$$\pi' = (\pi_{v_{mid}}, \pi_w, \pi_y, \pi_h, \pi_{v'_{mid}}, \pi_{w'}, \pi_{y'}, \pi_{h'}, \pi_{z'}) \tag{15}$$

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

$$e([v_0(s) + v_{in}(s) + v_{mid}(s)]_1, [w_0(s) + w_{in}(s) + w_{mid}(s)]_2) = \tag{16}$$
$$e([t(s)]_1, [h(s)]_2) \cdot e([y_0(s) + y_{in}(s) + y_{mid}(s)]_1, G_2) \tag{17}$$
$$e([v'_{mid}(s)]_1, G_2) = e([v_{mid}(s)]_1, [\alpha]_2) \tag{18}$$
$$e([w'(s)]_1, G_2) = e([w(s)]_1, [\alpha]_2) \tag{19}$$
$$e([y'(s)]_1, G_2) = e([y(s)]_1, [\alpha]_2) \tag{20}$$
$$e([h'(s)]_1, G_2) = e([h(s)]_1, [\alpha]_2) \tag{21}$$
$$e([z'(s)]_1, [\gamma]_2) = e([\beta_v v_{mid}(s) + \beta_w w(s) + \beta_y y(s)]_1, [\gamma]_2) \tag{22}$$

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 q-PKE where $q = 1$; 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.

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. **WHAT ARE THE OPTIMIZATIONS OVER GGPR?**

With these two methods as a springboard, Groth [Gro16] used stronger security assumptions to make massive improvements in 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 group element (ex. just $w(s)$ instead of $w(s), \alpha w(s)$). This eliminates multiple setup terms, reduced proof size to 3 group elements, and reduced verifier operations to a single pairing check using three pairings. This imaginably resulted in a massive performance increase across multiple fronts over previous methods. However, this single-element compilation prevents use of knowledge-of-exponent assumptions, so security is argued in the generic group model where, loosely speaking, adversaries cannot exploit any group specific properties. While seemingly strong, such assumptions are not necessarily uncalled for, as Gentry et al. have already shown that SNARKs for NP cannot exist without non-falsifiable assumptions to begin with [GW11]. Reduction in proof size and verifier complexity could also be regarded as "worth the security assumptions" on this front. Consequently, Groth's work (colloquially known as groth16) saw large adoption in blockchain systems verifying 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.

3 Circuit-independent (universal) pairing-based SNARKs

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. Consider the groth16 trusted setup for instance:

$$\sigma = \Big([\alpha]_1, [\beta]_1, [\gamma]_1, [\delta]_1, \{[x^i]_1\}_{i\in[0,n-1]}, \{[\gamma^{-1}(\beta u_i(x) + \alpha v_i(x) + w_i(x))]_1\}_{i\in[0,\ell]}, \tag{23}$$

$$\{[\delta^{-1}(\beta u_i(x) + \alpha v_i(x) + w_i(x))]_1\}_{i\in[\ell+1,m]}, \tag{24}$$

$$\{[\delta^{-1}x^i t(x)]_1\}_{i\in[0,\deg(h)]}, [\beta]_2, [\gamma]_2, [\delta]_2, \{[x^i]_2\}_{i\in[0,n-1]}\Big) \tag{25}$$

As is the case with other discussed methods, the contributions of the i-th witness component to gate g are "stuck" in the encodings of $u_i(x), v_i(x), w_i(x)$ (not the component's value, but its index). Thus if the circuit structure changes, the setup terms encoding the hidden combinations of the $v_i(x), w_i(x), y_i(x),$ etc. must be regenerated. A natural question arises: why not just update the setup somehow if the computation structure changes? This is not possible for two reasons, either of which is sufficient to deter updatability. The first is that, as mentioned the hidden $u_i(x), v_i(x), w_i(x)$ are interpolated from relationships between witness values (secret) 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 updating anything. The other 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.

To address these shortcomings, Groth [GKM$^+$18] produced a multivariate scheme that encoded QAP elements differently and achieved constant proof size and verification complexity. Although this scheme involved 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 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 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). Though they take up more space due to the presence of X^{-i} term for every X^i term (loosely speaking), the setup contains only univariate monomials and therefore requires linear space. Being composed of monomials, the setup can be updated by any party, for which they supply a proof of correctness. In a realistic deployment of this scheme, the setup would not be circuit dependent, and a single update could eliminate the risk that the deployers still hold valid toxic waste that can be used to subvert the setup. Maller et al. improved upon this with Marlin [?] a more verifier-succinct approach which cryptographically compiles a "sparsity-friendly" polynomial representation of an R1CS instance into familiar pairing checks. Within, both methods make use of bi-variate and univariate KZG commitments, respectively; more importantly, they also make use of *permutation arguments*, which prove the correct copying of values in consecutive circuit wires. **maybe say why these details are important, instead of randomly mentioning them**

Though significant in their own right, Sonic and Marlin suffer large constants in proof construction complexity despite being asymptotically quasilinear, with Sonic more negatively affected in this regard. A likely cause for Sonic's shortcoming here 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 et al. instead used a simpler arithmetization combined with structural assumptions about the number of inputs each circuit gate receives (2 inputs per gate). Here, each gate constraint can express any gate, and their permutation argument only requires a constant number of permutation check per gate due to the 2-fan-in assumption. Combining these traits with randomized interaction from the verifier yields a polynomial divisibility relation involving a random linear combination of the polynomials encoding the gate relations and copy constraints. This can be checked using univariate KZG commitment scheme, marking the use of pairing-based cryptography. The resulting scheme is known as "PlonK", where the "L" refers to "Lagrange bases" consisting of sparsely represented Lagrange polynomials defined over the roots of unity. These polynomials are used in the interpolation and aggregation of the various polynomials involved in proof construction.

The combination of flexibility, intuitive arithmetization, and better efficiency has made PlonK a catalyst for advancement.
Here is a table:

Table 1. Comparison of Pairing-based SNARK Systems

method	SRS size	prover-time	proof-length	verifier-time	universal	updatable	assumptions
GGPR13	$O(n)$	$O(n \log n)$	$7\mathbb{G}_1 + 1\mathbb{G}_2 + 1\mathbb{F}$	$O(n)$	No	No	q-PKE, q-PDH
PGHR13	$O(n)$	$O(n \log n)$	$7\mathbb{G}_1 + 1\mathbb{G}_2$	$O(n)$	No	No	q-PDH
Groth16	$O(n)$	$O(n \log n)$	$2\mathbb{G}_1 + 1\mathbb{G}_2$	$O(1)$	No	No	q-type
GMKL18	$O(n^2)$	$O(n \log n)$	$O(1)\mathbb{G}_1 + O(1)\mathbb{G}_2 + O(1)\mathbb{F}$	$O(1)$	Yes	Yes	SXDH
MBKM19	$O(n)$	$O(n \log n)$	$7\mathbb{G}_1 + 1\mathbb{G}_2 + 5\mathbb{F}$	$O(1)$	Yes	Yes	SXDH
GWC19	$O(n)$	$O(n \log n)$	$7\mathbb{G}_1 + 1\mathbb{G}_2$	$O(1)$	Yes	Yes	SXDH
CHM+19	$O(n)$	$O(n \log n)$	$7\mathbb{G}_1 + 1\mathbb{G}_2 + 6\mathbb{F}$	$O(1)$	Yes	Yes	AGM

4 Applications

4.1 Privacy-preserving pools

Barring some exceptions, most blockchains allow public access to transaction details. These could include the source wallet, destination wallet, the tokens transferred, and other potentially sensitive information. Thus, traditional blockchain network innerworkings make privacy preservation nontrivial at the application layer. Fortunately, modern SNARKS can help address this issue when endowed with zero-knowledge properties. A prime example of this is Tornado Cash [PSS19], an privacy-preserving pool running as a set of smart contracts on Ethereum network with some off-chain components. Tornado Cash allows one to deposit funds from wallet A, then withdraw the funds to a different wallet B. To do so, the user has to prove that they are the owner of the depositing wallet, and that they initiated it. They do so by constructing a commitment to the deposit using a secret value and the deposit details, and they submit this commitment to the smart contracts as part of the deposit process. The commitment is stored in a Merkle tree of all deposit commitments, and a nullifier is computed and stored to ensure the same deposit commitment is claimed only once. To withdraw, the user can do so via a different receiving wallet; they just need to submit a groth16 proof that their original commitment is present in the tree, which requires the source wallet info and secret they used to construct the commitment in the first place. The proof is just three group elements, as groth16 proofs usually are; but they encode way more information, like the merkle authentication path for the deposit commitment the tree, and the correct hashing procedure to produce the commitment and nullifier.

For instance, if we make a deposit (commitment C_6 in figure 2) and we want to withdraw later, we would submit a proof π encoding the secret, the nullifier hash, other commitment hash inputs,
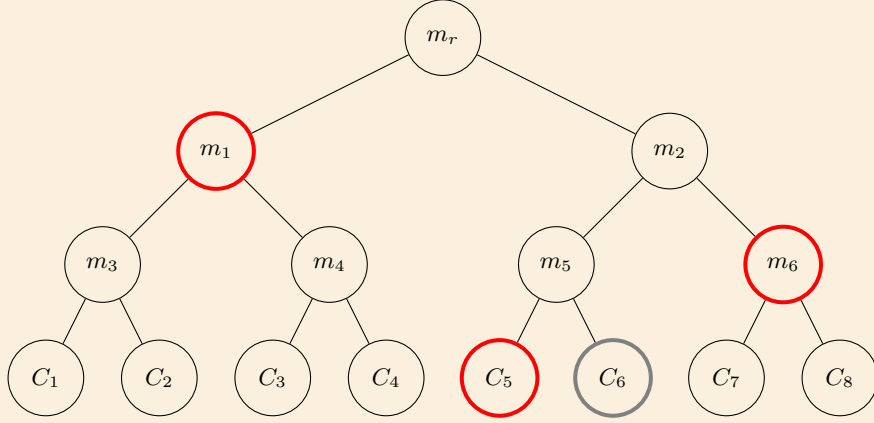
Fig. 2. Merkle tree with authentication path (in red) for commitment C6 (in gray)

and the list of nodes and their relative positions in the hashing sequence that would lead to the correct root if C_6 were in fact within the tree. The tornado cash infrastructure includes off-chain components with this functionality. In particular, it contains circuits written in Circom which can be used to generate and verify groth16 proofs of deposit commitment validity, and front-end/command-line tools to generate proofs.

4.2   Privacy-preserving blockchains

Beneath the application layer there are blockchains utilizing zero-knowledge properties of SNARKS for infrastructural privacy preservation. For each transaction submitted, such systems must obscure the source, relevant currencies, amounts used, and payment recipients. Zerocash [BSCG+14] is a notable example of a network that does such a thing. Intended to be a privacy-preserving version of Bitcoin [Nak08], Somewhat similar to (and preceding) Tornado Cash, Zerocash the unspent-transaction-output (UTXO) model, where transactions create so-called "unspent funds" that the party who "owns" them is entitled to spend. When they do spend those funds, another record of "unspent funds" for the recipient of the spending is created. These UTXO records are stored in a merkle tree, and spending funds requires proving knowledge of a leaf in the tree corresponding to the funds. 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 commitment scheme requiring no trusted setup (and adding flexibility for more complex future privacy-preserving computation).

4.3   Verifiable Virtual Machines (VVM)

So-called "Zero-knowledge virtual machines" (zkVMs) have emerged as an elegant solution to settling transaction batches from layer 2 (L2) blockchains on the layer 1 (L1) chain they derive crypto-economic security from. We use the more accurate phrase "verifiable virtual machine" (VVM) since succinctness is a higher priority than zero-knowledge in most L2 architectures. To align with real-world use cases like polygon zkEVM, Scroll, and others, we consider a centralized L2 blockchain with a single node that receives transactions, orders them, and creates blocks from them. The node then constructs a SNARK of block validity and sends this proof to a verifier contract on the L1 chain for verification. The SNARK is constructed from a large circuit which must constrain the witness to describe only valid computation paths through the VVM. Since the computation structure can change every time, this calls for universal SNARKs, of which PlonK and its variants are popular choices. More concretely, many VVM implementations use circuit domain-specific languages (DSLs) like halo2 to implement the circuits constraining the computation details.

A crucial point of consideration here is that not all computations performed in VVMs are "SNARK-friendly"; a prime example of this is hash functions like SHA256 which use non-linear or non-algebraic

operations like bit mixing. This can cause the circuits constraining this computation to be overly complex and inefficient. To meet this end, lookup-based methods are of great interest here since they can avoid constraining "SNARK-unfriendly" operations directly without losing soundness. An example of this is lies in the proof for validity of transaction that called a contract function using `sha256()`. Instead of constraining the exact sha-256 computation, the prover and verifier would agree on a publicly known table of acceptable inputs and outputs for SHA-256, which the prover commits to as part of the proof. The verifier does not need to know the whole table - they just need to know a commitment to the one considered correct. This can then be compared with what the prover committed to.

5   Future Research

5.1   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] use hashing-based cryptography relying on hash function collision-resistance, an assumption believed to hold against quantum adversaries. The common denominator is to reduce testing computational integrity to hashing-based polynomial proximity testing via methods like FRI [BSBHR18], which involve merkle tree commitments to Reed-Solomon codewords of witness polynomials. While such approaches are believed to be post-quantum secure, their use cryptographic primitive affording no algebraic structure hinders the attainment of smaller proof elements / faster verification. The most interesting direction addressing these issues involves SNARK constructions using lattice-based cryptography, which use algebraic objects but derive security from different post-quantum assumptions. Albrecht et al. [ACL+22] made some progress in this direction via a lattice-based SNARK with logarithmic-time verification. As it is operating over a completely different algebraic object, this method uses a completely different set of security assumptions than those discussed prior – namely, generalizations of the short integer solution (SIS) problem. Future work that quantifies the security assumptions in this area and decreases prover/verifier complexity would bring SNARKs with efficiency imposed by algebraic structure and post-quantum security to reality, which the blockchain ecosystem would eventually accept given sufficient tooling and support.

5.2   Automated verification of SNARK methods/applications

Currently, several layer 2 blockchains collectively hold tens of billions of dollars **[CITE]** protected by the integrity of SNARK methods used to verify block validity; if either the methods or their implementations have soundness bugs, it could allow a knowledgeable adversary to prove an invalid state transition for financial gain, likely causing massive financial loss to users in the process.

One approach to mitigate these risks is through formally verifying the behavior of SNARK methods and their implementations, especially in verifiable virtual machines. This way, we can have formal guarantees that such systems obey certain invariants mitigating various attack vectors and otherwise catastrophic impacts. There are two noteworthy and related directions of ongoing work in this domain. The first is related to formally verifying the SNARK methods themselves in various cryptographic models. An example of this is ArkLib by Dao et al. [ark24], a modular attempt to formally verify argument systems using the Lean proving assistant [dMU]. 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 in various mechanized theoretical models will make them increasingly fit for use in large-scale DeFi infrastructure/applications needing to verify computation.

The other direction concerns automated verification of VVM correctness. A notable example of this is Picus by Pailoor et al. [PCW+23] which uses symbolic execution and SMT solving to detect nondeterministic behavior in the circuit used by verifiable virtual machines to constrain execution traces. As we have seen in prior methods, the circuit structure directly influences the terms the verifier uses to check proofs, whether through the trusted setup in the non-univeral case or the committed

proof elements in the universal case. Checking that a given circuit configuration can produce at most one circuit output reduces the likelihood that some circuit value is underconstrained, thus making it more difficult to produce a proof from an tampered witness which passes verification. Picus has been used heavily in verifying Risc Zero VM [RIS23], and similar efforts exist for JOLT zkVM [AST24, KDT24] lookup-based arguments. 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 for the security of VVMs. For instance, it would be useful to build upon Picus and other tools of this nature so that they can detect nondeterminism in more complex circuits.

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.

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.

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.

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.

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.

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.

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).

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.

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.

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.

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.

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.

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.

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.

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].

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