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 $L \in \text{NP}$. 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 a 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 $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ 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 $\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}$ 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
- **soundness:** For all possible PPT algorithms Prove', if $\pi' \leftarrow \mathsf{Prove}'(\sigma, \mathsf{x}, \mathsf{w})$ and $x \notin L$, then $\mathsf{Verify}(\sigma, \mathsf{x}, \pi') = 1$ with probability at most δ_s ($\mathcal{V}$ rejects with probability at least $1 - \delta_s$).
- **knowledge soundness:** For all possible PPT algorithms Prove', if $\pi' \leftarrow \mathsf{Prove}'(\sigma, \mathsf{x})$ and $\mathsf{Verify}(\sigma, \mathsf{x}, \pi') = 1$ 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 $\pi \leftarrow \mathsf{Prove}(\sigma, \mathsf{x}, \mathsf{w})$. $|\pi|$ is polylogarithmic in $|x|$, prover complexity is quasilinear in $|x|$, and verifier complexity is polylogarithmic in $|x|$. Setup complexity is quasilinear 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 (there's some input out there making the circuit output 'True'. 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 and foremost 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.

An obvious choice of proof system would involve sending the full "certificate" of the given problem's solution to $\mathcal{V}$; however, this is not succinct or zero-knowledge. In light of this, the following question seems natural: 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 which address this fundamental issue use alternate characterizations of NP – namely rank-1 constraint systems (R1CS), and algebraic intermediate representations (AIRs) – 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 assumptions about what information an adversary must know to have produced a particular group element. Due to the use of elliptic curve cryptography (ECC), they also derive their security from the discrete log assumption, which is stronger than the collision resistance assumptions used in other hashing-based methods like STARKs. With the pairing-based denomination of SNARKs and others, the tradeoff between performance and security is a recurring and noteworthy point of comparison.

1.3 A general "workflow" for SNARKs

Modern SNARKs largely use the "workflow" depicted below 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 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

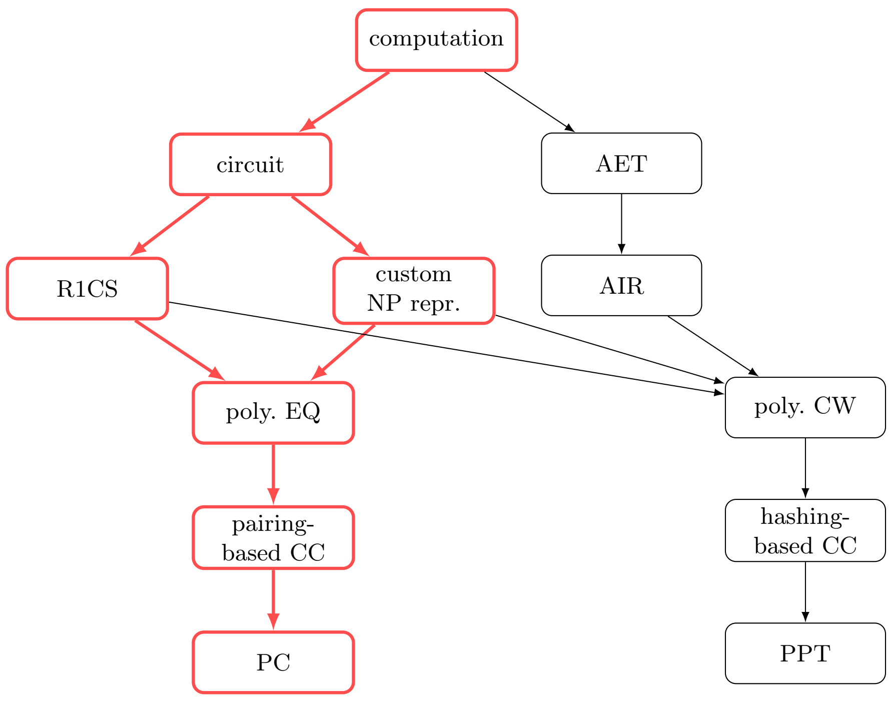

Fig. 1. A general workflow describing how SNARKs convert computations into verifiable statements. The highlighted path shows the course pairing-based approaches take. Acronyms: AET = algebraic execution trace, AIR = algebraic intermediate representation, CC = cryptographic compilation, CW = codewordx, EQ = polynomial equality, PC = pairing check, PPT = polynomial proximity testing, R1CS = rank-1 constraint system.

0 or 1. A natural question arises: are there representations of NP that seamlessly connect arithmetic circuit satisfiability with polynomial 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 lies in how little information one needs to distinguish between two arbitrary 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. Thus, if a succinct argument system can verify identities resemblent to this one, it would warrant the characterization of the claim being proven in a manner easily convertible to a collection/combination of polynomials.

1.6 Pairing-Friendly Elliptic Curve 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 coordinates of the points from. There is also a natural group operation over elliptic curve points behaving like addition; when compounded with the hardness of computing discrete logarithms over elliptic curve groups, this yields a scheme for additively homomorphic encryption. Pairings can render this scheme *doubly* homomorphic due to the properties of bilinear pairings. This paper abstracts the details of elliptic curve groups for brevity; we instead focus on the properties they exhibit which make them a useful component of pairing-based cryptography.

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 $P \in \mathbb{G}_{\mathbb{k}}$ 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 $s \in \mathbb{F}$, $e(P_1, sP_2) = e(sP_1, P_2) = e(P_1, P_2)^s$
- 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 scheme (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 KZG commitment scheme, named after the authors Kate et al. Kate-Zaverucha-Goldberg (KZG) commitments facilitate verification of the commitment made by a prover P to a specific polynomial f meeting certain requirements [7]. 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

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

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 points 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 upstream, and the reliance on the discrete log assumption (not post-quantum).

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 every modern pairing-based SNARK discussed.

1.8 Turning interactive protocols non-interactive

nothing.

2 Circuit-specific pairing-based SNARKs

The seminal PCP theorem yielded a powerful characterization of NP as the set of languages with polynomial-time PCP verifiers. However, early work on pairing-based SNARKs was actually motivated by the possibility of succinct arguments of knowledge using a representation *more suitable than PCPs* for integration with cryptographic primitives. Several other works 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) as a way of representing boolean/arithmetic circuit-SAT with polynomials, along with succinct NIZK for both QSP-SAT and QAP-SAT. Although the QSP construction is noteworthy, we limit discussion to their QAP-related construction due to the easier connection to arithmetic circuits and polynomials. We recall the strong QAP form here:

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

The idea was to leverage the notion of some additively homomorphic encoding E – namely, an injective, additively homomorphic function for which inversion is difficult – to compile a QSP/QAP instance into a proof containing just 9 group elements (non-ZK). We detail the non-zero-knowledge version here for clarity. Denote $E(x)$ by $[x]$. The prover $\mathcal{P}$ solves for $h(x)$ 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. 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:

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

and checks 6 equations; the first one checks the QAP relation, and the rest essentially compare proof elements with themselves, but recomposed in a way that tests if the prover knows the unhidden QAP elements:

$$(v_0(s) + v_{in}(s) + \pi_{v_{mid}})(w_0(s) + \pi_w) - (y_o + \pi_y) - \pi_h t(s) = 0 \tag{16}$$
$$\alpha v_{mid}(s) - v'_{mid}(s) = 0 \tag{17}$$
$$\alpha w(s) - w'(s) = 0 \tag{18}$$
$$\alpha y(s) - y'(s) = 0 \tag{19}$$
$$\alpha h(s) - h'(s) = 0 \tag{20}$$
$$z'(s) - \gamma\beta_v v_{mid}(s) - \gamma\beta_w w(s) - \gamma\beta_y y(s) = 0 \tag{21}$$

where $v'_{mid}(s)$ is encoded in $\pi_{v'_{mid}}$, $w'(s)$ is encoded in $\pi_{w'}$, and so on. This is done "in the exponent" using pairings, but we provide the unhidden equations for clarity.

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 for each sum in the QAP expression. We recall the regular QAP form:

$$\underbrace{\left(\sum_{j=1}^{n} a_i v_i(x)\right)}_{\text{left inputs}} \cdot \underbrace{\left(\sum_{j=1}^{n} a_i w_i(x)\right)}_{\text{right inputs}} - \underbrace{\left(\sum_{j=1}^{n} a_i y_i(x)\right)}_{\text{outputs}} = h(x)t(x) \equiv 0 \mod t(x) \tag{22}$$

The use of the same a_i across all sums 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. The resulting proof sent by the prover looks slightly different. Consider the following preprocessing elements (among others):

$$r_v, r_w, s, \alpha_v, \alpha_w, \alpha_y, \beta, \gamma \xleftarrow{R} \mathbb{F} \tag{23}$$
$$r_y = r_v r_w \tag{24}$$
$$g_v = r_v G \tag{25}$$
$$g_w = r_w G \tag{26}$$
$$g_y = r_y G \tag{27}$$

The prover $\mathcal{P}$ would send

$$\pi_2 = \Big([v_{mid}(s)], [w_{mid}(s)], [y_{mid}(s)], [h(s)], \tag{28}$$
$$[\alpha_v v_{mid}(s)], [\alpha_w w_{mid}(s)], [\alpha_y y_{mid}(s)], \tag{29}$$
$$[\beta_v v_{mid}(s) + \beta_w w_{mid}(s) + \beta_y y_{mid}(s)]\Big) \tag{30}$$

where like before, $v_{mid}(x) = \sum_{k \in \mathcal{I}_{mid}} a_k v_k(x)$ and likewise for $w_{mid}(x)$, $y_{mid}(x)$. Similar to GGPR, the verifier receives

THE POTENTIALLY VALID PROOF THE VERIIFER RECEIVES

and checks the following pairing-based equations:

$$e([v_0(s) + v_{in}(s) + v_{mid}(s)]_1 \cdot [w_0(s) + w_{in}(s) + w_{mid}(s)]_1, G_2) = \tag{31}$$

$$([t(s)]_1, [h(s)]_2) \cdot e([y_0(s) + y_{in}(s) + y_{mid}(s)]_1, G_2) \tag{32}$$

$$e([r_v v'_{mid}(s)], G) = e([v_{mid}(s)], [r_v\alpha]) \tag{33}$$

$$e([r_w w'_{mid}(s)], G) = e([w_{mid}(s)], [r_w\alpha]) \tag{34}$$

$$e([r_y y'_{mid}(s)], G) = e([y_{mid}(s)], [r_y\alpha]) \tag{35}$$

$$e(z'(s), [\gamma]) = e([r_v v_{mid}(s) + r_w w_{mid}(s) + r_y y_{mid}(s)], [\beta\gamma]) \tag{36}$$

Imaginably, this form of QAP became a precedent for later work. In particular, Groth used regular QAPs and stronger knowledge assumptions to produce a SNARK with proofs containing just three elliptic curve group elements while preserving succinct verification. The improvements here were largely due to more aggressive compilation of QAP polynomial elements into single setup terms, which decreased setup size and proof size at the expense of needing stronger assumptions about what witness information the prover knows.

While some would regard them as extreme, these stronger knowledge assumptions are not necessarily uncalled for; SNARKs for NP cannot exist without non-falsifiable assumptions to begin with, and reduction in proof length could also be regarded as "worth it" on this front. Consequently, Groth's work (colloquially known as groth16) saw large adoption in blockchain systems verifying homogeneous computation; applications like Tornado Cash and Railgun use this method to verify withdrawals from privacy-preserving pools, for instance.

3 Circuit-independent 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, 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:

$$\sigma = \Big([\alpha]_1, [\beta]_1, [\gamma]_1, [\delta]_1, \tag{37}$$

$$\{[x^i]_1\}_{i\in[0,n-1]}, \tag{38}$$

$$\{[\gamma^{-1}(\beta u_i(x) + \alpha v_i(x) + w_i(x))]_1\}_{i\in[0,\ell]}, \tag{39}$$

$$\{[\delta^{-1}(\beta u_i(x) + \alpha v_i(x) + w_i(x))]_1\}_{i\in[\ell+1,m]}, \tag{40}$$

$$\{[\delta^{-1}x^i t(x)]_1\}_{i\in[0,\deg(h)]}, \tag{41}$$

$$[\beta]_2, [\gamma]_2, [\delta]_2, \{[x^i]_2\}_{i\in[0,n-1]}\Big) \tag{42}$$

As is the case with other discussed methods, the contributions of the i-th witness component to gate g are “hardcoded” in the $u_i(x), v_i(x), w_i(x)$ (not the component's value, but its index). Thus if the computation 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). 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 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.

To address these shortcomings, Groth 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. Nonetheless, this hints that a SNARK with linear-size universal & updatable setup could be attainable if the setup terms are univariate (and monomial).

Maller et al. achieved this with Sonic, which draws inspiration from to build a SNARK reducing checking computation to checking evaluations of Laurent polynomials. Unlike with previous work, these polynomials can be represented in the (updatable) setup using univariate terms, requiring linear setup space. Though a breakthrough in its own right, Sonic suffers from very large constants in proof construction complexity despite being asymptotically quasilinear. A likely cause is the attempt to accommodate for n-fan-in circuits whose gates can accept arbitrarily many inputs. This means each intermediate gate would have an arbitrary amount of associated copy constraints, which would deny any attempt to condense or collapse the permutation argument based on structural assumptions. 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-runtime	proof-length	verifier-runtime	universal	updatable	security-model
GGPR	$O(n)$	$O(n \log n)$	9 group elements	$O(n)$	No	No	CRS, q-PKE, q-PDH
Pinocchio	$O(n)$	$O(n \log n)$	8 group elements	$O(n)$	No	No	CRS
Groth16	$O(n)$	$O(n \log n)$	3 group elements	$O(1)$	No	No	CRS, q-type
Groth-Updatable	$O(n^2)$	$O(n \log n)$	$O(1)$	$O(1)$	Yes	Yes	CRS
Sonic	$O(n)$	$O(n \log n)$	$O(1)$	$O(1)$	Yes	Yes	CRS
PlonK	$O(n)$	$O(n \log n)$	$O(1)$	$O(1)$	Yes	Yes	CRS
Marlin	$O(n)$	$O(n \log n)$	$O(1)$	$O(1)$	Yes	Yes	CRS, 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. When endowed with zero-knowledge properties, modern SNARKs can help address this issue as needed. A helpful solution could involve someone depositing into some liquidity pool from wallet A, then withdrawing some portion of that deposit to wallet B without revealing any deposit details (beyond the amount), including the link between wallets A, B. Tornado cash smart contracts make this possible by keeping a merkle tree of deposit commitments. When a user deposits into the pool, they create a commitment to their deposit derived partially from secret data only they should know. This commitment is inserted into the tree. When the same user goes to withdraw, they must provide a groth16 proof that their deposit commitment is indeed present in the tree. The homogeneous nature of the computation being verified – namely, the recursive hashing of nodes to the tree root – makes groth16 suitable for this use case.

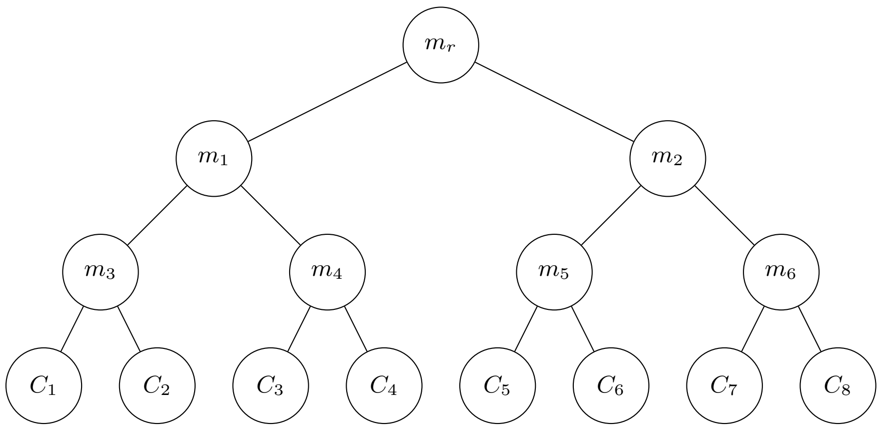

Fig. 2. Merkle tree of deposit commitments in a privacy-preserving pool

If we make a deposit associated with commitment C_6 and we want to withdraw later, we would have to construct a proof π showing that

– we know the inputs (secret included) that produce C_6
– we know the merkle authentication path involving C_5, m_6, m_1 that would yield m_r.
– we know the secret used to produce a *nullifier hash* which helps to mark π as already used so it cannot be replayed

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.

4.2 Privacy-preserving blockchains

discuss UTXO model and proof of membership

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 use of 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 Proof aggregation & Distributed Proving

As one would expect, proof generation remains the performance bottleneck in most SNARKs (pairing-based or not). In a world where computational resources are quite asymmetrically distributed (DeFi being no exception), it would be sensible to be able to outsource more intensive computations within the same proof to another party; naturally they would include a proof that such computation was done correctly. **Is there any work on this right now**. A similar idea emerges when we consider distributed proving of a statement by segmenting the *statement* instead of the *proof components*. Each delegated prover would then have to prove that a given subcircuit has an assignment which produces some desired value, then all of these proofs would get aggregated along with a proof that the aggregation was performed correctly.

5.2 Standardized benchmarks for pairing-based methods (and others)

Current SNARK benchmarks lack standardization by the computation being verified.

5.3 Integration of optimizations to pairing-based cryptography

optimize miller function

5.4 Formal verification of pairing-based systems

maybe

5.5 Post-quantum pairing-based SNARKs (how?)

Could be done using pairings for isogeny-based cryptography?

5.6 Extending pairings to higher-degree operations

nothing

References

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

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.

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.

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.

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

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