Baby SNARK (do do dodo dodo)

Andrew Miller Ye Zhang Sanket Kanjalkar

January 16, 2020*

1 Introduction

SNARKs, or verifiable computation, are some of the most important cryptographic techniques, widely used in blockchains and other secure distributed systems. They're very expressive, and offer a seemingly impossible performance promise: *Verify any computation in constant time!* However, SNARKs have also earned a reputation as "magic moon math." At least, they're somewhat more complicated than, say, public key signatures. There are already many excellent SNARK tutorials out there [1, 2], and this one is meant to be complementary. Our goals are:

- *Give a self-contained definition of the simplest possible SNARK that still shows off the "magic."* The SNARK we've selected, based on Square Span Programs due to Danezis et al [3], is expressive enough to represent arbitrary boolean circuits and includes a fully succinct proof and verifier.
- *Demystify the SNARK's soundness proof.* We've already seen vulnerabilities in real-life SNARK deployments due to inadequate soundness proofs [4, 5]. We've also noticed that existing tutorials, especially ones geared towards engineers, run out of steam by the time they arrive at this step. By simplifying the soundness proof and walking through it in detail, we hope to instill in readers the confidence to understand and validate the soundness proofs of other SNARKs as well.
- *Build a clear connection from the on-paper construction to the implementation in code.* As SNARKs are rapidly transitioning to practice and the focus is shifting to finer grained optimizations, state-of-the-art implementations are increasingly complicated to follow. This tutorial is accompanied by a codebase `https://github.com/initc3/babySNARK` that follows the on-paper definition very closely, so it can serve as a study guide to learn how SNARKs work and as a prototyping tool for SNARK researchers. The codebase includes 1) a naive implementation that has roughly cubic computation overhead for the prover and setup routines, as well as 2) an implementation with quasilinear overhead using FFT and sparse matrix optimizations.

Other differences between this tutorial and others. Since our focus is on simplifying the soundness proof, we ignore extra features like zero-knowledge hiding of the witness. In other words, we focus just on SNARKs, not zk-SNARKs.

To further cut down on notation clutter, we present our SNARK using symmetric (Type 1) bilinear groups. For performance, asymmetric (Type 2 or Type 3) bilinear groups are preferred.

The constraint system we describe, while expressive enough to embed boolean circuits, is not as concretely efficient as its generalization, *Rank-1 Constraint System*; for our purposes, we prefer it simply because it reduces the number of matrices to keep track of from three to one.

Other presentations carry around a constant $a_0 = 1$ term, separately from the statement $a_1, \ldots, a_\ell$; since this can just be included in the statement, we elide this to save a few symbols.

*This document may be updated frequently. Check `https://github.com/initc3/babySNARK` for updates.

Our soundness proof is in the Algebraic Group Model [6]. Proofs in this setting are simpler than in the standard model using knowledge-of-exponent assumptions. We help shed light on the nature of such proofs by formalizing a generic step (reduction to q-DLOG) that is reusable for many other SNARK variants as well.

2 Preliminaries

Polynomials over finite fields. BabySNARK, along with most of the SNARK protocols we know of, makes extensive use of polynomials over prime-order finite fields. Assume we are working in a finite field $\mathbb{F}_p$ of prime order p. For background on finite fields and polynomials, as well as their implementations, we recommend Kun, Ch. 2 [7, 8]. Our BabySNARK implementation uses the code provided with these references as a library. Here we recall a bit of notation and facts about polynomials.

Definition 2.1 (Coefficient Representation)**.** A degree-k bound polynomial $\phi(X) : \mathbb{F}_p \to \mathbb{F}_p$ is represented by $k+1$ coefficients $\{a_i\}_i$, such that $f(X) = a_0 + a_1 X + ... + a_k X^k$.

Definition 2.2 (Evaluation Representation)**.** A degree-k bound polynomial can alternatively be represented by $k+1$ or more values $\{y_i\}_i$, corresponding to evaluations of ϕ at fixed distinct locations $\{x_i\}_i$. These locations are called the *evaluation domain*.

We can convert between these two representations through Lagrange interpolation, or through FFT as we'll explain later. Some operations, including polynomial multiplication and division, can be computed more efficiently in the evaluation representation. Other times, however, it's necessary to work with the coefficient representation. We'll see this later on, for example, when the Prover evaluates a polynomial at an encrypted location.

Fact 2.1. Multiplying a degree-k_1 bound polynomial by a degree-k_2 bound polynomial results in a degree-$(k_1 + k_2)$ bound polynomial.

Fact 2.2 (Factor Theorem)**.** If a degree-k bound polynomial $p(X)$ has distinct roots at $(r_1, r_2 ... r_m)$, then it is divisible by $t(X) = (X - r_1)(X - r_2), ..., (X - r_m)$. In other words, $p(X) = t(X) \cdot h(X)$ for some degree-$(k - m)$ bound polynomial $h(X)$. We also write this as

$$p(X) \equiv 0 \mod t(X).$$

Fact 2.3. Given a degree-k polynomial over a finite field, we can solve for its roots in time polynomial in k.

Bilinear groups Let $\mathcal{G}$ be an additive group, with G a generator. We will use additive notation, just so that more of the math can be written in the same size font. We will use the notation $[a, b, ..., z]_G$ to mean $(aG, bG, ..., cG)$, as a way of visually distinguishing group elements from field elements.

Definition 2.3. A cyclic group $\mathcal{G}$ is bilinear if we have an efficiently computable pairing function, $e : \mathcal{G} \times \mathcal{G} \to \mathcal{G}_T$ for some target group G_T of the same order, such that

$$e([a]_G, [b]_G) = e(g, g)^{a,b}$$

and that $e(G, G)$ is a generator of the target group.

The form we have written above is more precisely called a symmetric, or *Type 1*, bilinear group; in general, we have may have two distinct source groups, $e : \mathcal{G}_1 \times \mathcal{G}_\in \to \mathcal{G}_T$.

3 Square Span Programs

Square Constraint Systems A square constraint system is defined by a matrix $U : \mathbb{F}^{m \times n}$. A vector $\vec{a} : \mathbb{F}^n$ is a solution to this system if

$$(U\vec{a}) \circ (U\vec{a}) = \vec{1}, \tag{1}$$

where $\circ$ represents the Hadamard (element-wise) product. We will also write $(U\vec{a}) \circ (\vec{a})$ as $(U\vec{a})^2$.

When we fix a prefix of this vector, the statement $\vec{a}_{\mathsf{stmt}} : \mathbb{F}^\ell$ for $\ell < n$, then the constraint matrix U defines a predicate, such that a witness $\vec{a}_{\mathsf{wit}} : \mathbb{F}^{n-\ell}$ satisfies the predicate if $\vec{a} = [\vec{a}_{\mathsf{stmt}};\ \vec{a}_{\mathsf{wit}}]^\intercal$ is a solution.

Encoding Boolean Circuits The Square Constraint System defined above is NP-complete, in the sense that it can easily be used to encode any Boolean circuit satisfiability problem. Consider a Boolean circuit of fan-in 2. Each wire in the circuit is represented by an element of the solution vector.

First, we show that we can express constraints such that an element a in the solution vector is either 0 or 1. Notice that

$$(2a - 1)^2 = 1$$

implies $(2a - 1) \in \{-1, 1\}$, which implies $a \in \{0, 1\}$. Next, observe that if a,b,c are all constrained to lie in $\{0, 1\}$, then we can express $c = \neg(a \wedge b) = \mathsf{NAND}(a, b)$ as

$$(2a + 2b - 5c + 4)^2 = 1.$$

Putting these together as an example, a square constraint program including the above wires and gates would take the form

$$\left(\begin{bmatrix} -1 & 2 & & & \\ -1 & & 2 & & \dots \\ -1 & & & 2 & \\ 4 & 2 & 2 & -5 & \\ \vdots & & & & \ddots \end{bmatrix} \cdot \begin{bmatrix} 1 \\ a \\ b \\ c \\ \vdots \end{bmatrix} \right)^2 = \vec{1}.$$

While NAND gates alone suffice to show that any Boolean circuit can be represented, other gates can also be encoded directly [3].

From Square Constraint Systems to Square Span Programs So far we've described a constraint system in terms of matrix multiplication. For the SNARK protocol, we need to translate this problem into one about polynomials. To start, we treat the columns of U as poynomials in the evaluation representation. More specifically, let $u_0(X), u_1(X), ..., u_{n-1}(X)$ be the degree $m-1$ polynomials defined through interpolation using each column of U, such that

$$u_i(r_j) = U_{j,i}.$$

Then we have that (1) is equivalent to the following:

$$\forall r_j.\ \left(\sum_{i=0}^{n-1} a_i u_i(r_j) \right)^2 - 1 = 0. \tag{2}$$

Now, recall from Fact 2.2, that if this polynomial it has roots at all of the $\{r_j\}$, then it is divisible by the vanishing polynomial on $\{r_j\}_j$. Hence this is equivalent to the Square Span Program,

$$\left(\sum_{i=0}^{n-1} a_i u_i(X) \right)^2 - 1 \equiv 0 \mod t(X) \tag{3}$$

This is the form we'll use in our SNARK protocol description. Square Span Programs can also be defined more generally, in terms of an arbitrary $t(X)$, although for us this will always be the vanishing polynomial [3].

Comparison with R1CS The Square Constraint System is a special case of the Rank-1 Constraint System commonly used. While Square Constraint Systems are defined by one constraint matrix U, Rank-1 Constraint Systems are defined three (U, V, W). That is, instead of $(U\vec{a})^2 = 1$, we have $U\vec{a} \circ V\vec{a} = W\vec{a}$. While both suffice for embedding boolean circuits, representing one bit per field element, with R1CS we can also represent arithmetic circuits that can compute some programs much more efficiently.

4 Soundness definition for SNARKs in the Algebraic Group Model

A SNARK for Square Span Programs (SSP) consists of three algorithms,

- $\sigma \leftarrow \mathsf{Setup}(U)$ generates the public parameters σ from a SSP constraint matrix U.
- $\pi \leftarrow \mathsf{Prove}(\sigma, \vec{a}_{\mathsf{stmt}}, \vec{a}_{\mathsf{wit}})$ takes a statement and witness and produces a proof π
- $\{0,1\} \leftarrow \mathsf{Verify}(\sigma, \vec{a}_{\mathsf{stmt}}, \pi)$ checks a proof

The main security goal for a SNARK is soundness, roughly that if $\mathsf{Verify}(\sigma, \vec{a}_{\mathsf{stmt}}, \pi) = 1$, then there exists some satisfying witness $\vec{a}_{\mathsf{wit}}$. A knowledge-soundness definition (the K in SNARK) is actually even stronger, and says that if an adversary can produce a valid proof, then the adversary must *know* the witness. This is formally captured by stating that for any adversary $\mathcal{A}$ producing such a proof, we can construct an extractor $\mathcal{E}$ that produces the witness.

Algebraic Group Model We give our security proof in the Algebraic Group Model (AGM) [6], because it leads to the simplest statements and proofs. In the AGM, we say that an adversary that outputs group elements must also explain where these came from, as a linear combination alg of group elements the adversary has previously seen. One reason this model simplifies our security proof is that we can use a single discrete-log style reduction target, q-DLOG, for proving the soundness of most SNARKs, rather than variations of Knowledge-of-Exponent and Diffie-Hellman assumptions as in earlier proofs.

Knowledge Soundness The following attack game lets the adversary choose both the statement and the proof, and the attack only succeeds if the corresponding extractor fails to find the witness.

$$\mathsf{Adv}^{\textsc{Knwl-Sound}}_{k,\mathsf{Setup},\mathsf{Verify}}(V, \mathcal{A}, \mathcal{E}) = \Pr\left[\begin{array}{r} \sigma \leftarrow \mathsf{Setup}(V) \\ \vec{a}_{\mathsf{stmt}}, \pi, \mathsf{alg} \leftarrow \mathcal{A}(\sigma) \\ \vec{a}_{\mathsf{wit}} \leftarrow \mathcal{E}(\mathsf{alg}) \\ \hline \mathsf{Verify}(\sigma, \vec{a}_{\mathsf{stmt}}, \pi) \wedge \\ (V \cdot (\vec{a}_{\mathsf{stmt}} \shortparallel \vec{a}_{\mathsf{wit}})^{\intercal})^2 \neq \vec{1} \end{array}\right]$$

Definition 4.1 (Knowledge Soundness)**.** Putting these together, A SNARK is knowledge sound if $\forall \mathcal{A}$, there exists an extractor $\mathcal{E}$, such that the advantage $\mathsf{Adv}^{\textsc{Knwl-Sound}}_{k,\mathsf{Setup},\mathsf{Verify}}(V, \mathcal{A}, \mathcal{E})$ game is negligible in the size of the field.

5 Baby SNARK

We now discuss the BabySNARK protocol, defined in Figure 1 and discuss it below.

Note on indexing There are three kinds of indexing that occur. To keep the clutter down, we abbreviate them, but explain more clearly here:

- Just the statement elements $_{i<\ell}$, could be written more verbosely as $_{i=0}^{n-1}$
- Just the witness elements $_{i\geq\ell}$, more verbosely $_{i=\ell}^{n-1}$
- The entire solution vector $_{i\geq 0}$, more verbosely $_{i=0}^{n-1}$

Trusted Setup This is a preprocessing SNARK — the performance of the Verify and Prove routines are improved because they depend on precomputation performed during setup.

The SNARK requires a trusted setup to produce the Common Reference String (CRS). Unlike a uniform CRS, which could be generated in public from a beacon, our CRS is structured. This means the only effective way to sample from it is to first sample a few uniform values called trapdoors, and then to derive the CRS from these. The setup outputs the CRS, but it very important that the trapdoor does not leak in any way (if it does, it's known as "toxic waste"). Our implementation of Setup is appropriate for a trusted third

Setup$(1^\lambda, \{u_i\}_{i\geq 0})$:

$\tau, \beta, \gamma \xleftarrow{\$} \mathbb{F}^3$

CRS $\sigma := \ [1, \tau, ..., \tau^m, \ \gamma, \ \gamma\beta, \ \{\beta u_i(\tau)\}_{i\geq \ell}\ ]_G$

Precompute $[t(\tau), \{u_i(\tau)\}_{i\geq 0}]_G$

Prove$(\sigma, \vec{a})$:

$h(X) := \left(\left(\sum_{i\geq 0} a_i u_i(X) \right)^2 - 1 \right) / \, t(X)$

$H := [h(\tau)]_G$

$V_w := \sum_{i\geq \ell} a_i \, [u_i(\tau)]_G$

$B_w := \sum_{i\geq \ell} a_i \, [\beta u_i(\tau)]_G$

$\pi := (H, V_w, B_w)$

Verify$(\sigma, \vec{a}_{\mathsf{stmt}} = \{a_i\}_{i<\ell}, \pi)$:

$V_s := \sum_{i<\ell} a_i \, [u_i(\tau)]_G$

$V := V_s + V_w$

$e(H, [t(\tau)]_G) \ e(g, g) \stackrel{?}{=} e(V, V)$

$e(B_w, [\gamma]_G) \stackrel{?}{=} e([\gamma\beta]_G \, , V_w)$

Figure 1: BabySNARK protocol definition

party. Multiparty computation or hardware enclaves would be preferable approaches to implementing the trusted setup. Modifying babySNARK to include circuit-independent trusted setups [9], or with entirely transparent setups [10, 11], would be important ways to improve.

Computing in the exponent The $H := [h(\tau)]_G$ term must be "computed in the exponent." That is, first compute the coefficients of polynomial $h(X) = h_0 + h_1 X + ... + h_m X^n$, and then $H = \sum_j h_j \left[\tau^j\right]_G$ using the powers of τ from the CRS. In other words, we evaluate $h(X)$ at τ without ever learning τ.

The other terms, V_w, B_w, and V_s are computed by the Prover with the help of precomputation from the Setup. We distinguish the precomputation from the CRS, because the precomputation does not have to be part of the "trusted" portion of the setup. That is, given just the CRS, even without any other information about the trapdoor, the precomputation can be computed in the exponent by anyone in the public.

An intuitive explanation of the design The CRS is built around three trapdoors τ, β, γ. The first $m+1$ terms, $[1, \tau, ..., \tau^m]_G$ can be used by the prover and verifier to evaluate (in the exponent) any degree-m polynomial on τ. The $\{\beta u_i(\tau)\}$ terms are more restrictive, and allow the prover only to evaluate a polynomial that lies in the linear span of $\{u_i(X)\}_{i\geq \ell}^{n-1}$. This corresponds to $U\vec{a}$ for some vector $\vec{a}_{\mathsf{wit}}$. The γ and $\gamma\beta$ terms are used by the verifier to check that these are consistent with each other.

Extraction (Knowledge Soundness) Proof To validate the intuitive explanation more carefully, we now work through the formal security theorem and proof.

Theorem 1. *The BabySNARK construction above is sound in the Algebraic Group Model as long as the q-DLOG assumption holds for $q \geq n + 1$.*

Proof. The proof term π consists of three group elements V_w, B_w, H. Since we assume an algebraic adversary, these must come along with a representation $\mathsf{alg} = V_w(\cdot), B_w(\cdot), H(\cdot)$ as a linear combination of elements from the CRS. We write these out in terms of variables X, Y, Z to visually distinguish these from the sampled trapdoor elements τ, β, γ. Since the proof involves tracing the presence or absence of these terms across several equations, we also color code them to make it easier to follow.

$$B_w(\cdot) = b_0 + b_1 X + ... + b_n X^n + b_\gamma Z + b_{\gamma\beta} Y Z + \{b_i' Y u_i(X)\}_{i\geq \ell}$$

$$V_w(\cdot) = v_0 + v_1 X + ... + v_n X^n + v_\gamma Z + v_{\gamma\beta} Y Z + \{v_i' Y u_i(X)\}_{i\geq \ell}$$

$$H(\cdot) = h_0 + h_1 X + ... + h_n X^n + h_\gamma Z + h_{\gamma\beta} Y Z + \{h_i' Y u_i(X)\}_{i\geq \ell}$$

So the verifying process means the following equations hold at least when $X = \tau, Y = \beta, Z = \gamma$

$$(I) \qquad B_w(\cdot) = YV_w(\cdot)$$
$$(II) \quad H(\cdot)\,t(\cdot) = V(\cdot)^2 - 1$$

The proof proceeds in two cases. The first case is different for every SNARK protocol, while the second case is generic (and discussed in more detail in Section 6.

Case 1 (Constraints hold everywhere): Both (I) and (II) hold for $\forall X, Y, Z$. If the polynomials produced by the adversary satisfy the equations everywhere, then we can show the coefficients $\{a_i'\}_{i\geq\ell}$ from $V_w(\cdot)$ are a satisfying witness.

(1) Starting from the Equation (I), since RHS has a factor Y for every item, $B(\cdot)$ in the LHS must only contain terms with a Y component. So $b_0, b_1, \ldots, b_n, b_\gamma$ are all 0, so we can write $B(\cdot)$ as

$$B_w(\cdot) = b_{\gamma\beta}YZ + \{b_i'Yu_i(X)\}_{i\geq\ell}$$

Dividing by Y, we also see that $V(\cdot)$ must not have any Y terms at all.

$$V_w(\cdot) = b_{\gamma\beta}Z + \{b_i'u_i(X)\}_{i\geq\ell}$$

(2) Next turning to Equation (II), we use $V_w(\cdot)$ from Step (a) to get $V(\cdot)$ below. $\{a_i\}_{i<\ell}$ is the public statement, and we define $\{b_i'\}_{i<\ell} = \{a_i\}_{i<\ell}$ for simplicity of notion.

$$V(\cdot) = V_w(\cdot) + \{a_iu_i(X)\}_{i<\ell} = b_{\gamma\beta}Z + \{b_i'u_i(X)\}_{i\geq 0}$$

Then we replace $V(\cdot)$ in equation (1) using the form mentioned above. If $b_{\gamma\beta} \neq 0$, then we would have a $b_{\gamma\beta}^2Z^2$ term in the RHS. However, since in the LHS Z only appears with degree-1 in $H(\cdot)$ and does not appear in $t(\cdot)$ at all, we can conclude that $b_{\gamma\beta} = 0$.

Finally we arrive at the conclusion that the coefficients $\{b_i'\}_{i\geq\ell}$ define a satisfying witness such that

$$V(\cdot) = \{b_i'u_i(X)\}_{i\geq 0}$$
$$V(\cdot)^2 - 1 = 0 \mod t(\cdot).$$

Case 2 (Constraints hold at (τ, β, γ) but not everywhere): At least one of (1) and (2) doesn't hold for $\forall X, Y, Z$. Define two polynomials

$$Q_1(\cdot) = B(\cdot) - YV(\cdot)$$
$$Q_2(\cdot) = H(\cdot)\,t(\cdot) - V(\cdot)^2 + 1.$$

In both cases, we get a multivariate polynomial $Q(X, Y, Z)$ which is not zero but when evaluating at (τ, β, γ) will get zero. Using Lemma 1 from Section 6, we know that such prover will win the polynomial checking game which has negligible probability.

6 Generic Reduction from q-DLOG to Polynomial Checking

In this section we state and prove a general lemma about embedding an instance of the DLOG problem into the setup for the snark game.

In the Generic Group Model, the polynomials chosen by the adversary must be distributed independently of $\vec{\tau} = (\tau, \beta, \gamma)$. Hence Case 2 occurs with low probability by applying the Schwarz-Zippel lemma. In the Algebraic Group Model however, the polynomials may be dependent on $\vec{\tau}$, hence a different approach is needed.

The proof from Fuchsbauer et al. [6] is based on embedding an instance of the q-DLOG problem into the CRS given to the adversary. As they noted, this reduction does not depend on the particular details of the SNARK, just that the CRS is based on polynomial functions of uniformly sampled trapdoors. Gabizon also gave a proof sketch for generic statement here [12].

q-DLOG assumption The q-DLOG game is similar to the ordinary discrete log, except that the adversary sees multiple powers.

$$\mathsf{Adv}^{q\text{-DLOG}}(\mathcal{A}) = \Pr\left[\begin{array}{r} z \xleftarrow{\$} \mathbb{F} \\ \hline z' \leftarrow \mathcal{A}(1, g^z, \ldots, g^{z^q}) \\ z = z' \end{array}\right]$$

The q-DLOG assumption is that this advantage is negligible.

$$\forall \mathcal{A}, \mathsf{Adv}^{q\text{-DLOG}} \leq \mathsf{negl}(\|\mathbb{F}\|)$$

Polynomial Checking Game Let $\vec{S} : \mathbb{F}^k \to \mathbb{F}^d$ be a vector of multi-variable polynomials of maximum degree q. The polynomials $\vec{S}$ correspond to the SNARK setup, where $\vec{\tau}$ is the uniformly sampled trapdoor and $\sigma \leftarrow g^{\vec{S}(\vec{\tau})}$ is the resulting common reference string.

$$\mathsf{Adv}^{\textsc{PolyCheck}}_{k,\vec{S}}(\mathcal{A}) = \Pr\left[\begin{array}{r} \vec{\tau} \xleftarrow{\$} \mathbb{F}^k \\ \sigma \leftarrow g^{\vec{S}(\vec{\tau})} \\ Q(\vec{\mathbf{T}}) \leftarrow \mathcal{A}(\sigma) \\ \hline Q(\vec{\tau}) = 0 \ \wedge \ Q(\vec{\mathbf{T}}) \not\equiv 0 \end{array}\right]$$

Lemma 1. *Let $\vec{S} : \mathbb{F}^k \to \mathbb{F}^d$ be a set of polynomials of maximum degree q. For each adversary $\mathcal{A}$ that wins the* PolyCheck *game, we can derive an adversary $\mathcal{B}_{\mathcal{A},\vec{S}}$ that solves the q-DLOG problem with similar probability.*

$$\forall \mathcal{A}, \mathsf{Adv}^{\textsc{PolyCheck}}_{k,\vec{S}}(\mathcal{A}) \leq \mathsf{Adv}^{q\text{-DLOG}}(\mathcal{B}_{\mathcal{A},\vec{S}}) + \frac{k}{\|\mathbb{F}\|}$$

Proof. We define $\mathcal{B}_{\mathcal{A},\vec{S}}$ as follows:

$$\begin{array}{r} \mathcal{B}_{\mathcal{A},\vec{S}}(g, g^z, \ldots, g^{z^q}) : \\ \vec{r} \xleftarrow{\$} \mathbb{F}^k, \ \vec{s} \xleftarrow{\$} \mathbb{F}^k \\ \sigma \leftarrow g^{\vec{S}(\vec{r}z + \vec{s})} \\ Q(\vec{\mathbf{T}}) \leftarrow \mathcal{A}(\sigma) \\ \text{Let } Q'(\mathbf{Z}) = Q(\vec{r}\,\mathbf{Z} + \vec{s}) \\ \text{Solve for roots of } Q'(\mathbf{Z}) \text{ to find } z \text{ (Fact 2.3)} \end{array}$$

Suppose $\mathcal{A}$ has advantage p in the PolyCheck game. $Q(\vec{\mathbf{T}})$ in the PolyCheck game is distributed identically as $Q(\vec{\mathbf{T}})$ in the q-DLOG game, since $\vec{\tau}$ is distributed the same as $\vec{r}z + \vec{s}$. Let d^* be the maximum total degree of $Q(\tilde{\mathbf{T}})$ in PolyCheck. With probability p, we have $d^* \geq 1$, since $Q(\vec{\tau}) = 0$ but $Q(\vec{\mathbf{T}}) \not\equiv 0$.

Now consider the polynomial $Q''(\vec{\mathbf{R}}, \vec{\mathbf{S}}, \mathbf{Z}) = Q(\vec{\mathbf{R}}\mathbf{Z} + \vec{\mathbf{S}})$. Collecting the $\mathbf{Z}$ terms, we can write this as

$$Q''(\vec{\mathbf{R}}, \vec{\mathbf{S}}, \mathbf{Z}) = q^*(\vec{\mathbf{R}})\mathbf{Z}^{d^*} + q_{\mathsf{rest}}(\vec{\mathbf{R}}, \vec{\mathbf{S}}) \cdot \left(1, \mathbf{Z}, ..., \mathbf{Z}^{d^*-1}\right)^{\mathsf{T}}$$

for some polynomial $q^*(\vec{\mathbf{R}})$ and $q_{\mathsf{rest}}(\vec{\mathbf{R}}, \vec{\mathbf{S}}) : \mathbb{F}^{2k} \to \mathbb{F}^{d^*-1}$. That is, the coefficient of the $\vec{\mathbf{Z}}^{d^*}$ term, $q^*(\vec{\mathbf{R}})$, cannot depend on $\tilde{\mathbf{S}}$. By the Schwartz-Lippel lemma, since $\vec{r}$ is sampled uniformly from $\mathbf{F}^k$, independently of $\vec{\tau}$, we have that $\Pr[q^*(\vec{r}) = 0] \leq k/\|\mathbb{F}\|$. Hence we have that the coefficient of the $\mathbf{Z}^{d^*}$ term in $Q'(\mathbf{Z})$ is zero with at most this probability. Note that this holds regardless of the fact that $\vec{s}$, z, and other terms in Q' may be dependent on $\vec{r}$ and $\vec{\tau}$. □

7 FFT-based and Sparsity-based optimizations

In the accompanying code, we include an alternative implementation that shows off a few algorithmic improvements. In particular, the Setup and Prover computation overhead is reduced from quadratic to quasi-linear (with the Verify routine remaining the same).

Sparsity The BabySNARK protocol is defined for any square constraint system. However, when we generate the constraints by compiling a boolean circuit as described, the resulting constraint system will be sparse. More specifically, if the circuit has constant average fan-in (say, an average fan-in of 2), then U is an $m \times n$ matrix, but will have only $O(m)$ non-zero elements with $m > n$. To take advantage of this sparsity, we can implement the evaluation representation of polynomials (e.g., of $u_i(X)$) so as to keep track of just the non-zero elements. For example, when the prover sums the $a_i u_i(X)$ terms while computing $h(X)$, this summation alone incurs an $O(m^2)$ cost. In the sparse representation, this only costs $O(m)$ since we only add in the non-zero terms.

FFT-friendly parameters We can also improve the running time of interpolation and extrapolation from $O(m^2)$ to $O(m \log m)$. For example, when the prover computes H by evaluating $h(\tau)$ in the exponent, we need to interpolate $h(X)$ to its coefficient form.

The SNARK construction and soundness proof work just fine for any choice of distinct roots, $r_0, ..., r_{m-1}$. However, to implement radix-2 FFT, we need that the order of the multiplicative group of $\mathbb{F}_p^*$ is highly 2-adic, meaning that $2^\kappa | p-1$ for some sufficiently large $2^\kappa \geq m$. When this is the case, we can find a primitive 2^κ'th root of unity, such that $\omega_0, \omega_1, ..., \omega_{2^\kappa - 1}$ are all distinct, but $\omega^{2^\kappa} = 1$. We can then use the Fast Fourier Transform (FFT) over finite fields, also known as the Number Theoretic Transform. We choose the roots accordingly,

$$r_0, ..., r_{m-1} = \omega^0, \omega^1, ..., \omega^{m-1}.$$

.

We summarize the resulting asymptotic performance after applying these algorithmic optimizations, assuming the field size is constant, and that the statement is size ℓ. TODO: improve this

	Setup	Prover	Verifier	Proof Size
BabySNARK	$O(m^3)$	$O(m^3)$	$O(\ell)$	$O(1)$
BabySNARK (opt)	$O(m \log m)$	$O(m \log m)$		

Table 1: Asymptotic performance analysis of babySNARK algorithms

Acknowledgements

We thank Dakshita Khurana and Charalampos Papamanthou for suggestions.

References

[1] Anca Nitulescu. A gentle introduction to snarks. `https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf`.

[2] Maksym Petkus. Why and how zk-snark works. *arXiv preprint arXiv:1906.07221*, 2019.

[3] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct nizk arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 532–550. Springer, 2014.

[4] Ariel Gabizon. On the security of the bctv pinocchio zk-snark variant. *IACR Cryptology ePrint Archive*, 2019:119, 2019.

[5] Bryan Parno. A note on the unsoundness of vntinyram's snark. Cryptology ePrint Archive, Report 2015/437, 2015. https://eprint.iacr.org/2015/437.

[6] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *Annual International Cryptology Conference*, pages 33–62. Springer, 2018.

[7] Jeremy Kun. A programmer's introduction to mathematics. https://pimbook.org/, 2018.

[8] Jeremy Kun. Programming with finite fields. https://jeremykun.com/2014/03/13/programming-with-finite-fields/, 2014.

[9] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-snarks. In *Annual International Cryptology Conference*, pages 698–728. Springer, 2018.

[10] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *Annual International Cryptology Conference*, pages 701–732. Springer, 2019.

[11] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. Cryptology ePrint Archive, Report 2019/1047, 2019. https://eprint.iacr.org/2019/1047.

[12] Ariel Gabizon. Moving snarks from the generic to algebraic group model. https://medium.com/@arielgabizon/moving-snarks-from-the-generic-to-algebraic-group-model-56549d60b90d, 2019.