BabySpartan: Lasso-based SNARK for non-uniform computation

Srinath Setty[*] Justin Thaler[†]

Abstract

Lasso (Setty, Thaler, Wahby, ePrint 2023/1216) is a recent lookup argument that ensures that the prover cryptographically commits to only "small" values. This note describes BabySpartan, a SNARK for a large class of constraint systems that achieves the same property. The SNARK is a simple combination of SuperSpartan and Lasso. The specific class of constraint systems supported is a generalization of so-called Plonkish constraint systems (and a special case of customizable constraint systems (CCS)). Whereas a recent work called Jolt (Arun, Setty, and Thaler, ePrint 2023/1217) can be viewed as an application of Lasso to *uniform* computation, BabySpartan can be viewed as applying Lasso to non-uniform computation.

1 Introduction

A succinct non-interactive argument of knowledge (SNARK) [Kil92, Mic94, GGPR13] allows an untrusted prover to prove that it knows a witness satisfying some property. Most SNARKs are designed by combining a protocol called a *polynomial IOP* with a *polynomial commitment scheme*. Popular examples of polynomial commitment schemes include KZG [KZG10] and FRI [BBHR18] (for univariate polynomials), and KZG+Gemini [BCHO22], Zeromorph [KT23], Ligero [AHIV17], Brakedown [GLS$^+$23], Hyrax [WTS$^+$18], and Bulletproofs/IPA [BCC$^+$16, BBB$^+$18] (for univariate or multilinear polynomials).

The notion of Plonkish constraint systems was introduced to roughly capture the most general class of constraint systems that Plonk [GWC19], a popular SNARK, can prove statements about. However, the polynomial IOP underlying Plonk requires the prover to commit to many random field elements, even when all variables in the witness are small. Such random field elements are much more expensive to commit to with many polynomial commitment schemes, especially those based on elliptic curve group operations, such as Bulletproofs/IPA, KZG and its variants, and Hyrax.

SuperSpartan [STW23a], a natural generalization of Spartan [Set20], is an alternative SNARK for that applies to a substantial generalization of Plonkish constraint systems. This generalization is called *customizable constraint systems* (CCS). In the case of "uniform" CCS instances, the Super-Spartan prover does *not* commit to any random values (we will define what we mean by uniform shortly). However, for non-uniform instances of CCS, the SuperSpartan prover does commit to a number of random values that is linear in the number of non-zero entries of the CCS constraint matrices.

In this note, we describe an alternative SNARK, which we refer to as BabySpartan, for Plonkish that avoids the prover committing to random values. This construction is a combination of SuperSpartan [STW23a] and Lasso [STW23b] (the latter is also a generalization of a core component of Spartan [Set20] to obtain a lookup argument). In particular, BabySpartan can be viewed as a SNARK for non-uniform circuits built from (indexed) lookup arguments.

We hope that BabySpartan makes it easier to combine the performance benefits of Lasso with existing tooling, which often involves non-uniform constraint systems.

[*]Microsoft Research
[†]a16 crypto research and Georgetown University

2 Preliminaries

2.1 Background on the sum-check protocol

Let g be an ℓ-variate polynomial over field $\mathbb{F}$. The sum-check protocol [LFKN90] is an interactive proof for proving claims of the form, where $T \in \mathbb{F}$:

$$T = \sum_{x \in \{0,1\}^\ell} g(x). \tag{1}$$

It consists of ℓ rounds, with the prover sending a univariate polynomial s_i in each round i, whereby the degree of s_i is at most the degree of g in its ith variable. At the end of the sum-check protocol, the verifier must evaluate $g(r)$ for a single $r \in \mathbb{F}^\ell$. Hence, from the verifier's perspective, the sum-check protocol is a reduction from the task of checking Equation (1), which involves evaluating g at 2^ℓ inputs and summing the results, to the potentially easier task of evaluating g at the single random point r.

Theorem 1. *The sum-check protocol is a perfectly complete interactive proof protocol for proving*

$$T = \sum_{x \in \{0,1\}^\ell} g(x),$$

with soundness error at most $\ell \cdot d/|\mathbb{F}|$.

The sum-check protocol is public-coin, which means it can be rendered non-interactive via the Fiat-Shamir transformation [FS86]. It is known to satisfy round-by-round soundness, and hence if the interactive protocol has soundness error $2^{-\lambda}$, the non-interactive protocol obtained by applying Fiat-Shamir has roughly λ bits of security [BCS16, CCH+19, CMS19].

An ℓ-variate polynomial is said to be multilinear if it has degree at most one in each variable. It is well-known that if $g(x) = \prod_{i=1}^{k} p_i(x)$ where each p_i is multilinear, then given as input $\{p_i(x) \colon x \in \{0,1\}^\ell, i = 1, \ldots, k\}$, the sum-check protocol prover applied to g can be implemented in $O_k(2^\ell)$ time, where the O_k notation hides a dependence on k that is at most quadratic [Tha13, STW23a].

2.2 Multilinear extensions

It is well-known that for any function $f \colon \{0,1\}^\ell \to \mathbb{F}$, there exists a unique ℓ-variate multilinear polynomial $\widetilde{f}$ such that $\widetilde{f}(x) = f(x)$ for all $x \in \{0,1\}^\ell$. We refer to $\widetilde{f}$ as the multilinear extension of f.

For a vector $a \in \mathbb{F}^n$, where n is a power of 2, we similarly define the multilinear extension $\widetilde{a} \colon \mathbb{F}^{\log n} \to \mathbb{F}$ as follows. Interpret a in the natural way as listing all n evaluations of a function with domain $\{0,1\}^{\log n}$, and define $\widetilde{a}$ to be the multilinear extension of this function.

Lagrange basis polynomials. For any $S \in \{0,1\}^\ell$, let

$$\chi_S(x) = \prod_{i=1}^{\ell} (x_i \cdot S_i + (1 - x_i) \cdot (1 - S_i))$$

denote the S'th multilinear Lagrange basis polynomial. For example, if $\ell = 4$ and $S = (0,1,1,0)$, then $\chi_S(x) = (1 - x_i) \cdot x_2 \cdot x_3 \cdot (1 - x_4)$.

The following lemma is also well-known. Below, we index entries of $a \in \mathbb{F}^n$ by integers in $\{0, 1, \ldots, n-1\}$, and, in the natural way, associate each such integer with a bit vector in $\{0,1\}^{\log n}$ and vice versa.

Lemma 1 (Multilinear Lagrange interpolation)**.** *For any vector $a \in \mathbb{F}^n$, $\widetilde{a}(r) = \sum_{i=0}^{n-1} a_i \cdot \chi_i(r)$.*

Let $\widetilde{\mathsf{eq}}_\ell \colon \mathbb{F}^\ell \times \mathbb{F}^\ell \to \mathbb{F}$ be the following multilinear polynomial:

$$\widetilde{\mathsf{eq}}_\ell(x, y) = \prod_{j=1}^{\ell} (x_j \cdot y_j + (1 - x_j) \cdot (1 - y_j)).$$

$\widetilde{\mathsf{eq}}_\ell$ is the unique multilinear polynomial satisfying, for all $x, y \in \{0,1\}^\ell$,

$$\widetilde{\mathsf{eq}}_\ell(x,y) = \begin{cases} 1 \text{ if } x = y \\ 0 \text{ otherwise.} \end{cases}$$

That is, $\widetilde{\mathsf{eq}}_\ell$ is the so-called multilinear extension of the equality function over $\{0,1\}^\ell \times \{0,1\}^\ell$. Note that for any $S \in \{0,1\}^\ell$, $\widetilde{\mathsf{eq}}_\ell(S,y) = \chi_S(y)$. We omit the subscript ℓ from $\widetilde{\mathsf{eq}}_\ell$ when ℓ is clear from context.

2.3 Customizable constraint systems

The following notion of customizable constraint systems (CCS) is defined in [STW23a]. CCS generalizes popular constraint systems including Plonkish, R1CS, and AIR without overheads.

Definition 2.1 (CCS)**.** *A CCS structure $\mathcal{S}$ consists of:*

- *size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ where $n > \ell$;*
- *a sequence of matrices $M_0, \ldots, M_{t-1} \in \mathbb{F}^{m \times n}$ with at most $N = \Omega(\max(m,n))$ non-zero entries in total;*
- *a sequence of q multisets $[S_0, \ldots, S_{q-1}]$, where an element in each multiset is from the domain*

  $$\{0, \ldots, t-1\}$$

  and the cardinality of each multiset is at most d.
- *a sequence of q constants $[c_0, \ldots, c_{q-1}]$, where each constant is from $\mathbb{F}$.*

A CCS instance consists of public input $x \in \mathbb{F}^\ell$. A CCS witness consists of a vector $w \in \mathbb{F}^{n-\ell-1}$. A CCS structure-instance tuple $(\mathcal{S}, x)$ is satisfied by a CCS witness w if

$$\sum_{i=0}^{q-1} c_i \cdot \bigcirc_{j \in S_i} M_j \cdot z = \mathbf{0}, \tag{2}$$

where

$$z = (w, 1, x) \in \mathbb{F}^n, \tag{3}$$

$M_j \cdot z$ denotes matrix-vector multiplication, $\bigcirc$ denotes the Hadamard product between vectors, and $\mathbf{0}$ is an m-sized vector with entries equal to the the additive identity in $\mathbb{F}$.

2.4 Polynomial IOPs and polynomial commitments

Roughly, a polynomial IOP is an interactive protocol where, in one or more rounds, the prover may send to the verifier a large polynomial g. Because g is so large, one does not wish for the verifier to read a complete description of g. Instead, in any efficient polynomial IOP, the verifier only "queries" g at one point (or a handful of points). This means that the only information the verifier needs about g to check that the prover is behaving honestly is one (or a few) evaluations of g.

In turn, a polynomial commitment scheme enables an untrusted prover to succinctly *commit* to a polynomial g, and later provide to the verifier any evaluation $g(r)$ for a point r chosen by the verifier, along with a proof that the returned value is indeed consistent with the committed polynomial. Essentially, a polynomial commitment scheme is exactly the cryptographic primitive that one needs to obtain a succinct argument from a polynomial IOP. Rather than having the prover send a large polynomial g to the verifier as in the polynomial IOP, the argument system prover instead cryptographically commits to g and later reveals any evaluations of g required by the verifier to perform its checks.

2.5 Indexed lookup arguments

A lookup argument allows an untrusted prover to commit to a vector $a \in \mathbb{F}^m$ and prove that all entries of a reside in some predetermined table $t \in \mathbb{F}^n$. In an indexed lookup argument, in addition to a commitment to $a \in \mathbb{F}^m$, the verifier is handed a commitment to a second vector $b \in \mathbb{F}^m$. The prover claims that for all $i = 1, \ldots, m$, $a_i = t_{b_i}$. We refer to a as the vector of *looked-up values*, and b as the vector of *indices.*

Definition 2.2 (Statement proven in an indexed lookup argument)**.** *Given commitment* cm_a *and* cm_b*, and a public array* T *of* N *field elements, represented as vector* $t = (t_0, \ldots, t_{N-1}) \in \mathbb{F}^N$ *to which the verifier has (possibly) been provided a commitment* cm_t*, the prover knows an opening* $a = (a_0, \ldots, a_{m-1}) \in \mathbb{F}^m$ *of* cm_a *and* $b = (b_0, \ldots, b_{m-1}) \in \mathbb{F}^m$ *of* cm_b *such that for each* $i = 0, \ldots, m-1$*,* $a_i = T[b_j]$*, where* $T[b_j]$ *is short hand for the* b_j*'th entry of* t*.*

Lasso [STW23b] gives an indexed lookup argument where the prover commits to a vector of length m, referred to as "read counts", and a second vector of length n, referred to as "final counts". When the prover is honest, all of the read counts and final counts are in $\{0, \ldots, m\}$,

3 BabySpartan: Lasso-based SNARK for non-uniform circuits

To describe BabySpartan, it is cleanest to first focus on a popular special case of CCS, namely R1CS. In this case, the prover commits to a witness vector vector $z \in \mathbb{F}^n$ and the prover's goal is to prove that $Az \circ Bz = Cz$ where A, B, and C are public $m \times n$ matrices and $\circ$ denotes the Hadamard (i.e., entry-wise) vector product. BabySpartan can be viewed as a simple modification of SuperSpartan to incorporate Lasso.

Let $a = Az$, $b = Bz$, and $c = Cz$. The SuperSpartan prover runs two instances of the *sum-check protocol* [LFKN90]. In the first instance, the verifier selects a random $\tau \in \mathbb{F}^{\log m}$, and applies the sum-check protocol to the polynomial

$$g(x) = \widetilde{\mathsf{eq}}(\tau, x) \left(\widetilde{a}(x) \cdot \widetilde{b}(x) - \widetilde{c}(x) \right),$$

using it to confirm that

$$0 = \sum_{x \in \{0,1\}^{\log m}} g(x).$$

If z satisfies the constraint system then this equality is guaranteed to hold, and if z does not satisfy the constraint system then this equality will fail to hold with probability at least $1 - 2\log m/|\mathbb{F}|$.

At the end of this first invocation of the sum-check protocol, the verifier has to evaluate $\widetilde{a}(r)$, $\widetilde{b}(r)$ and $\widetilde{c}(r)$ at a random point $r \in \mathbb{F}^{\log m}$. Hence, this first instance of the sum-check protocol reduced the task of checking that z satisfies the constraint system, to the task of evaluating the multilinear extensions of a, b, and c each at a random point $r \in \mathbb{F}^{\log m}$.

Lasso [STW23b] provides a protocol for this task when A, B, C have exactly one non-zero entry per row and were committed in a pre-processing phase (using any multilinear polynomial commitment scheme). So, BabySpartan simply invokes Lasso for this task. This completes a description of BabySpartan. For completeness, in the next section we unpack how Lasso works in this context.

3.1 Details of using Lasso to compute $\widetilde{a}(r)$, $\widetilde{b}(r)$, and $\widetilde{c}(r)$

Background on SuperSpartan. In SuperSpartan [STW23a], the verifier invokes the sum-check protocol a second time to reduce the task of computing $\widetilde{a}(r)$ to that of evaluating $\widetilde{A}(r, r')$ and $\widetilde{z}(r')$ for some random $r' \in \mathbb{F}^{\log n}$, and similarly for $\widetilde{B}$ and $\widetilde{C}$. $\widetilde{z}(r')$ can be obtained from the commitment to $\widetilde{z}$, via one evaluation query at evaluation point r'.

If $\widetilde{A}(r, r')$ can be evaluated in (poly)logarithmic time, then the verifier can compute this quantity on its own and still run in (poly)logarithmic time (this is what we mean above by "uniform" R1CS instances). But, for general matrices $A \in \mathbb{F}^{m \times n}$, the amount of time required to evaluate $\widetilde{A}(r, r')$ may be linear in the number of non-zero entries of A. In this case, SuperSpartan assumes that $\widetilde{A}$ has been committed in pre-processing by an honest party, using a so-called *sparse polynomial commitment scheme* that it describes, called Spark [Set20]. Spark provides a procedure to prove an evaluation of $\widetilde{A}(r, r')$. Unfortunately, this procedure requires the prover to commit to many random field elements (the number of committed random field elements is proportional to the number of non-zero entries of A). Lasso shows how to avoid the prover needing to commit to random field elements.

Details of Lasso. Lasso is a generalization of Spark. In both Spark and Lasso, the matrix A is committed with dense vectors in the same way. Specifically, the commitment is to two vectors u and v of length m. The first vector u has i'th entry equal to index (in $\{0, \ldots, n-1\}$) of the unique column with non-zero entry in row i of A. The second vector v has i'th entry equal to A_{i,u_i}). Note that u and v together completely specify the matrix A. In the context of a SNARK for R1CS, such a commitment is created by the verifier (or another trusted party) in a preprocessing step.

The proof of the value of $\widetilde{a}(r)$. Recall that $a = Az$, where A was committed in a preprocessing phase by an honest party as per the above paragraph, and $\widetilde{z}$ was committed by the prover using any polynomial commitment scheme.

Lasso uses the following expression for $\widetilde{a}$:

$$\widetilde{a}(x) = \sum_{i=1}^{m} \widetilde{\mathsf{eq}}(x, i) \cdot v_i \cdot z_{u_i}. \tag{4}$$

To see that Equation (4) holds, observe that the right hand side is clearly a multilinear polynomial in x. Moreover, it agrees with a for all inputs $x \in \{0,1\}^{\log m}$. This holds because, if A_j denotes the j'th column of z, then

$$a = Az = \sum_{i=0}^{m-1} A_{i,u_i} \cdot z_{u_i}.$$

And so the x'th entry of a equals

$$\sum_{i=1}^{m} \widetilde{\mathsf{eq}}(i, x) \cdot A_{i,u_i} \cdot z_{u_i} =$$
$$\sum_{i=1}^{m} \widetilde{\mathsf{eq}}(i, x) \cdot v_i \cdot z_{u_i}. \tag{5}$$

Hence, the right hand side of Equation (4) must equal the unique multilinear extension of a.

Accordingly, to prove what is the value of $\widetilde{a}(r)$, the Lasso prover commits to a vector $w \in \mathbb{F}^m$ whose i'th entry is purportedly z_{u_i}. Lasso then applies the sum-check protocol to the $(\log m)$-variate polynomial

$$g(y) = \widetilde{\mathsf{eq}}(r, y) \cdot \widetilde{v}(y) \cdot \widetilde{w}(y)$$

to confirm that Expression (5) equals the claimed value of $\widetilde{a}(r)$. At the end of the sum-check protocol, the verifier needs to evaluate g at a random point r'''. It can evaluate $\widetilde{\mathsf{eq}}(r, r''')$ on its own in $O(\log m)$ time, while $\widetilde{v}(r''')$ and $\widetilde{w}(r''')$ can be obtained from the commitments to $\widetilde{v}$ and $\widetilde{w}$ respectively.

All that remains is for the verifier to confirm that $\widetilde{w}(x) = z_{\widetilde{u}(x)}$ for all $x \in \{0,1\}^{\log m}$. Lasso gives a way to do this based on a technique called *offline memory-checking* [BEG+91, SAGL18, Set20, GLS+23].[1]

Specifically, mapping our setting to the indexed lookup argument in Lasso, z is the lookup table, u is the indices of the m lookups, and and w is the claimed results of the lookups.

[1]Spark that underlies SuperSpartan also uses the offline memory checking, but the lookup argument is applied on tables that contain evaluations of the $\widetilde{eq}$ polynomial (at certain randomly-chosen inputs), for the purpose of proving sparse polynomial evaluations. Even if z contains "small" field elements, these tables contain random field elements, so the lookup argument ends up requiring the prover to commit to random field elements.

For completeness, we sketch how the lookup argument works. Consider the length-m vector q whose i'th entry is the number of rows $j \leq i$ with $u_j = u_i$. The prover commits to a length-m vector y whose i'th entry is z_{u_i} (here, z can be thought of as a lookup table), as well as a length-m vector α whose i'th entry is the number of rows $j \leq i$ with $u_j = u_i$ (think of these as read-counts, i.e., the i'th row specifies which cell of z is read by the i'th read operation, and α_i describes how many times the memory cell read at time i has been read by earlier read operations), followed by a length-n vector β whose j'th entry is the total number of non-zero entries in column j of A (think of these values as final counts). Using these committed values, Lasso gives a way to confirm that $y_i = z_{u_i}$ for $i = 1, \ldots, m$. Roughly speaking, it confirms that the vector of (value, count) pairs returned by read operations is a permutation of the vector of (value, count) pairs obtained by combining the "initialization" (i.e., commitment to) the lookup table z with the (value, count) pairs obtained by incrementing each read-count by 1.

3.2 The SNARK

In general, our SNARK, BabySpartan, applies to CCS instances in which each constraint matrix $M_0, \ldots, M_{t-1}$ has exactly one non-zero entry per row. Our prior work [STW23a] described a simple and essentially costless transformation from Plonkish constraint systems to CCS that is guaranteed to yield CCS constraint matrices of this form.

Specifically, after the prover commits to the multilinear extension of a purported solution vector z for the CCS instance,[2] SuperSpartan [STW23a] invokes the sum-check protocol once to reduce the task of confirming that z is a solution, to the task of evaluating $\widetilde{a}_i(r)$ for a random $r \in \mathbb{F}^{\log m}$, where $a_i = M_i \cdot z$ for $i = 0, \ldots, t-1$. As explained in Section 3.1, Lasso directly gives a SNARK for computing $\widetilde{a}_i(r)$.

Note that multiple invocations of Lasso can be batched via standard techniques to ensure that the proof length of the various invocations of the sum-check protocol underlying Lasso does not grow with the number t of constraint matrices.

Pre-processing costs. In pre-processing, for each constraint matrix, an honest party commits to m column identifiers (i.e., the MLE of the vector u), m values (the MLE of the vector v specifying the non-zero entry in each row), m read-counts (the MLE of the vector α) and n final-counts (the MLE of the vector β).

Prover costs. The prover commits to (the MLE of) the solution vector z of length n, and for each constraint matrix it commits to (the MLE of) the length-m vector w of claimed results to the lookups into z.

Note that if the prover is honest, none of the committed vectors have any entries larger than those in z, or those of the constraint matrices, or the number m and n of rows and columns of the constraint matrices. This is what we mean when we say the prover only commits to small values. For all of these commitments (including the ones computed in pre-processing), any multilinear polynomial commitment scheme can be used.

The SNARK prover must evaluate $M_i \cdot z$ for each $i = 0, \ldots, t-1$. The number of field operations required is clearly proportional to the number of non-zero entries of the constraint matrices. On top of this, the Lasso prover performs $O(m+n)$ field operations for each of the t constraint matrices (this does not asymptotically increase prover field work beyond what is required simply to evaluate $M_i \cdot z$).

The prover provides one evaluation proof for each committed polynomial (for many polynomial commitment schemes, evaluation proofs can be batched, so that both prover and verifier roughly pay for only a single evaluation proof across all committed polynomials and evaluation points).

Verifier costs. The proof size $O(\log^2(n+m))$ field elements, plus the evaluation proof provided for each committed polynomial (see the remark above on batching these evaluation proofs across all committed polynomials). The verifier's runtime is $O(\log^2(n+m))$ field operations plus the cost of checking the evaluation proofs. The former can be reduced to $O(\log(m+n) \log\log(m+n))$ at the cost of a low-order additive increase in the number of elements the prover commits to (see [STW23b] for details).

[2]More precisely, if $z = (w, 1, x)$ for a public vector x, then the prover commits to $\widetilde{w}$, and $\widetilde{z}(r)$ can be evaluated efficiently with one evaluation query to $\widetilde{w}$. See [STW23a] for details.

4 Discussion

Comparison with Diamond and Posen [DP23]. In a new paper, Diamond and Posen [DP23] give an alternative SNARK for Plonkish constraint systems, in which, like BabySpartan, the prover only commits to small values. Their SNARK is based on Hyperplonk [CBBZ23] combined with Lasso. Diamond and Posen also give an improvement to the Ligero/Brakedown [AHIV17, GLS+23] hashing-based polynomial commitment scheme, enabling extremely fast commitments to "small" values. Their SNARK for Plonkish is motivated by ease of integration with their polynomial commitment scheme. In particular, it is convenient for them to view the witness vector z as itself being decomposed into columns, where each column potentially satisfies a different size bound. This makes Plonkish constraint systems a natural target, and Hyperplonk a natural starting point for their SNARK.

We think that BabySpartan is conceptually simpler and may have efficiency benefits when combined with elliptic-curve-based commitment schemes. We leave a careful comparison of Diamond and Posen's SNARK to BabySpartan to near-term future work.

Generalizing BabySpartan to handle arbitrary CCS. BabySpartan proves a particular class of CCS including the ones that we obtain from transforming Plonkish to CCS. One might wonder if it is possible to extend BabySpartan to handle general CCS instances in which a row of a CCS matrix might have more than one non-zero entry. There are some possible extensions, but they all end up having the prover pay cryptographic commitment costs proportional to the number of non-zero entries in the CCS matrices (i.e., there are no "free" additions or linear combinations), and require the prover to commit to intermediate state of the linear combinations (i.e., commit to "partial linear combinations").

One can achieve a similar effect by transforming general CCS instances to Plonkish instances (which introduces additional addition gates to emulate linear combinations) and then apply BabySpartan. Note that the generality of CCS still allows lower proving costs when SuperSpartan is applied to *uniform* instances of CCS (meaning that the multilinear extensions of the constraint matrices are efficiently evaluable by the verifier). This is because in this case SuperSpartan itself already has the prover only cryptographically commit to the witness. That is, the prover's cryptographic costs are proportional only to the size of the witness and not the number of non-zero entries (so the linear combinations are effectively "free").

Disclosures. Justin Thaler is a Research Partner at a16z crypto and is an investor in various blockchain-based platforms, as well as in the crypto ecosystem more broadly (for general a16z disclosures, see `https://www.a16z.com/disclosures/`.)

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthu Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2018.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2016.

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orru. Gemini: Elastic snarks for diverse environments. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2022.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive Oracle Proofs. In *Theory of Cryptography Conference (TCC)*, 2016.

[BEG$^+$91] Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1991.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2023.

[CCH$^+$19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2019.

[CMS19] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In *Theory of Cryptography Conference (TCC)*, pages 1–29, 2019.

[DP23] Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. Cryptology ePrint Archive, Paper 2023/1784, 2023.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings of the International Cryptology Conference (CRYPTO)*, pages 186–194, 1986.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2013.

[GLS$^+$23] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In *Proceedings of the International Cryptology Conference (CRYPTO)*, pages 193–226, 2023.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. ePrint Report 2019/953, 2019.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1992.

[KT23] Tohru Kohrita and Patrick Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. Cryptology ePrint Archive, Paper 2023/917, 2023.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 177–194, 2010.

[LFKN90] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, October 1990.

[Mic94] Silvio Micali. CS proofs. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1994.

[SAGL18] Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2018.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2020.

[STW23a] Srinath Setty, Justin Thaler, and Riad Wahby. Customizable constraint systems for succinct arguments. Cryptology ePrint Archive 2023/552, 2023.

[STW23b] Srinath Setty, Justin Thaler, and Riad S. Wahby. Lasso: Unlocking the lookup singularity. Cryptology ePrint Archive 2023/1216, 2023.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2013.

[WTS$^+$18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.