Efficient Foreign-Field Arithmetic in PLONK

Miguel Ambrona
Input Output Global
Spain
miguel.ambrona@iohk.io

Denis Firsov
Input Output Global
Estonia
denis.firsov@iohk.io

Iñigo Querejeta-Azurmendi
Input Output Global
Spain
querejeta.azurmendi@iohk.io

Abstract

PLONK is a prominent universal and updatable zk-SNARK for general circuit satisfiability, which allows a *prover* to produce a short certificate of the validity of a certain statement/computation. Its expressive model of computation and its highly efficient verifier complexity make PLONK a powerful tool for a wide range of blockchain applications.

Supporting standard cryptographic primitives (such us ECDSA over SECP256k1) or advanced recursive predicates (e.g. incrementally verifiable computation) on a SNARK presents a significant challenge. It requires so-called *foreign-field arithmetic* (enforcing constraints over algebraic fields that differ from the SNARK native field) which was previously believed to incur an overhead of two or three orders of magnitude.

We build on the techniques by Lubarov and Baylina and observe that, by considering tight bounds on their encoding of foreign-field multiplication, the number of PLONK constraints can be significantly reduced. We show that these techniques also extend to elliptic curve emulation, with an overhead of just one order of magnitude (with respect to its native counterpart). We validate soundness and completeness of our main results in EasyCrypt. Finally, we implement an open-source library with support for foreign-field arithmetic. Our experimental results showcase the generality of our techniques and confirm their suitability for real-world applications.

Contents

1 Introduction

SNARKs [GGPR13, BCG+13, PHGR13, Gro16], succinct non-interactive arguments of knowledge, are a class of cryptographic schemes that allow a *prover* to produce a certificate of the validity of a certain statement/computation, which can then be publicly verified. Remarkably, the proof size and verification complexity are sublinear (if not constant) in the size of the statement being proven. SNARK proofs can be optionally performed in *zero-knowledge* [GMR85, BFM88] mode, which will ensure that the certificate does not reveal any additional information beyond the fact that the statement holds.

PLONK [GWC19] is a universal and updatable zk-SNARK for general circuit satisfiability. PLONK is based on a very powerful model of computation with so-called *custom gates* which allow for a very versatile and expressive encoding of constraints. This efficient arithmetization has led to its widespread adoption in many state-of-the-art blockchain projects. PLONK has been integrated into systems like Zcash [HBHW], Mina [BMRS20], the Dusk Network [MKF21], Anoma [GYB21], or Midnight [Mid24], among many others.

Indeed, PLONK (and SNARKs in general) have become a fundamental tool in applications requiring verifiable computation or privacy, particularly in *blockchain* technology. In the context of *rollups*, projects such as Aztec [Wil18], zkSync [Mat23], and Polygon [Pol23] leverage SNARKs to bundle multiple transactions[1] into a single proof, significantly reducing on-chain computation and gas costs. Privacy-focused blockchains like Zcash [HBHW], Dusk [MKF21], or Midnight [Mid24] use SNARKs to enable confidential transactions, allowing users to shield sensitive financial data while maintaining cryptographic integrity. SNARKs are also playing an increasingly important role in proofs of identity, particularly in privacy-preserving frameworks. With the growing adoption of self-sovereign identity systems (SSI) and digital identity standards such us eID or ePassport, SNARKs offer a unique way of proving ownership or possession of certain credentials without revealing any unnecessary personal information.

Another very prominent application of SNARKs is *recursion*. The high efficiency of SNARK verifiers (with sublinear, even constant, verification complexity) makes it possible to involve the SNARK verifier in the NP predicate being proven. *Incrementally verifiable computation* (IVC) [Val08] is one of the applications of recursion and enables continuous verification of an ongoing computation without reprocessing previous steps. With IVC one can generate a succinct proof of the validity of a blockchain state. Such proof guarantees the validity of the whole chain, but can be verified very efficiently (its verification complexity is independent of the chain length). The Mina [BMRS20] blockchain is an example of IVC put into practice. Recursion is the Holy Grail of cryptographic proofs, enabling unbounded scalability, constant verification time, and

[1] Or even smart contract executions, in so-called ZK virtual machines (zkVMs).

seamless composability. However, achieving *efficient* recursion is extremely challenging in practice, given the high cost of the prover and the fact that SNARK verification typically involves operations that are costly to express in the SNARK's model of computation.

Actually, most of the above-mentioned compelling applications rely on modeling complex predicates within a SNARK. These often involve standard cryptographic primitives like ECDSA signatures over SECP256k1, combined with hash functions such as SHA256 or Keccak [BDPV13]. Encoding these primitives as a SNARK computation is inherently challenging and can introduce substantial computational costs. Nevertheless, these requirements are driven by the specific use case and switching to more SNARK-friendly primitives may not always be a feasible alternative.

Foreign-Field Arithmetic. Every SNARK has its own dedicated model of computation, typically a set of multivariate polynomial equations (of a predefined shape) over a finite field K. Performing arithmetic over K, known as *the native field*, is highly efficient. However, encoding computations over a different field with polynomial constraints over K can result significantly more involved. This is known as *foreign-field arithmetic*, also referred to as *wrong-field arithmetic* and was believed to incur an overhead of two or three orders of magnitude [DH20].

In order to circumvent the problem of emulating foreign-fields, some attempts try to use less standard but SNARK-friendly primitives such as algebraic hash functions [GKR+21, GLR+20, GKL+22, GKS23] or *embedded elliptic curves* (whose base field is the SNARK native field). In the case of recursion, a very clever and popular workaround [ECC21] consists of running two SNARKs over a so-called *cycle of curves*: a pair of elliptic curves such that the scalar field of one is the base field of the other and vice versa. This cyclic structure allows them to encode part of the recursive verification circuit in one SNARK and the rest in the other. This, however, leads to an extremely complex and error-prone recursive predicate. Besides, these schemes must give up constant verification complexity as the existing cycles of curves are not equipped with an efficient pairing in both curves.[2]

Designing techniques for emulating foreign-field arithmetic over different fields more efficiently is extremely valuable, as it can lead to conceptually simpler, more elegant, and less error-prone recursive SNARKs. It also serves to support the standard cryptographic primitives required by many real-world applications.

1.1 Our contributions

We pursue the study of foreign-field emulation in PLONK's model of computation. We present several new and very general techniques for encoding emulated field and elliptic curve operations.

Field emulation. We build on the techniques by Lubarov and Baylina [LB22]. We observe that, by considering tighter bounds in their analysis of foreign-field multiplication, the number of PLONK constraints can be dramatically reduced.

We also present a novel way of encoding simpler operations like addition, subtraction, negation, or multiplication by constants, which does not require range-checks. This new encoding takes advantage of the fact that these operations are linear with respect to the representation of emulated field elements in limbs form.

Elliptic curves emulation. We demonstrate that the above techniques extend to the case of elliptic curve emulation. Instead of enforcing EC operations via black-box use of foreign-field addition and multiplication, we develop dedicated custom foreign-field identities. This, combined with new ideas for maximizing the use of *incomplete* (but more efficient) building blocks, leads to an efficient encoding of non-native EC operations.

For example, we can model a multi-scalar multiplication (with full-size scalars) of ℓ points of the SECP256k1 curve, emulated over the BLS12-381 scalar field, with about $3900\ell + 5000$ PLONK rows (on an architecture of 9 witness columns and range-check lookups of 16 bits). This is arguably only one order of magnitude more costly than the same operation over a curve whose base field is the native field (e.g. Jubjub [ECC18] in this case).

Formal verification of our results. We formulate and prove our main results in EasyCrypt, to validate their soundness and completeness within a rigorous formal framework. We believe this effort eliminates potential soundness vulnerabilities that often emerge when working with such complex sets of equations. We hope that this level of assurance boosts trust in our approach and encourages its wider adoption. Our formal proofs of soundness and completeness (in EasyCrypt) can be found in GitHub [AFQ25].

Implementation and evaluation. We implement our foreign-field emulation techniques on top of the Halo2 Rust library [ECC21]. Our framework is highly versatile and supports the emulation of different fields with configurable parameters. Our implementation leverages advanced PLONK features, including custom gates and lookup tables, to efficiently encode our foreign-field arithmetic equations. To demonstrate the generality of our approach, we evaluate our techniques by computing KZG-based PLONK proofs over the BN254 and BLS12-381 curves, implementing two use cases: (i) verification of standard signature schemes (such as ECDSA over SECP256k1); (ii) self-recursion, where the SNARK predicate includes the verification of a SNARK proof. Our benchmarks demonstrate the efficiency of our techniques across various emulation settings, confirming their suitability for real-world applications. Our implementation will be released as open-source software.

1.2 Technical overview and related work

The state-of-the-art techniques for emulating integer and foreign-field arithmetic [LB22, Gab22, SQ22] are based on the idea that an equality over the integers can be enforced through equalities modulo a set M of auxiliary moduli. In virtue of the Chinese Remainder Theorem:

$$\forall m \in M.\ x = 0 \pmod{m} \quad \Longleftrightarrow \quad x = 0 \pmod{\mathrm{LCM}(M)}\ .$$

This equation, combined with additional bounds on the magnitude of x, e.g. imposing that $|x| < \mathrm{LCM}(M)$, imply that x must be the integer 0. Naturally, through algebra over the integers, one can easily emulate algebra over any prime field.

We will constrain an expression x to be zero modulo several auxiliary moduli M, one of which will of course be the native modulus

[2] Some examples are the Pasta cycle [Hop20], used by Mina [BMRS20], which does not have an efficient pairing in either curve; or the Pluto-Eris cycle [Hop21], where Pluto has a pairing but Eris does not.

of the SNARK. However, how come it is possible to constrain an equality modulo other moduli? Was not that the problem in the first place? It turns out this is possible if the moduli are small or structured enough so that the terms in the equation are guaranteed not to *"wrap-around"* the native modulus. If our modulus of interest were of this amenable form, it would be very easy to emulate arithmetic over it as one custom identity would probably be enough. Unfortunately, the modulus we are interested in will often be similar in size to our native modulus, or even larger, requiring a more complex approach.

There exist different approaches for choosing the moduli in M. As we mentioned, all of them use the native modulus (which is "free"), but they differ in how the remaining moduli are chosen.

- The approach by Aztec [Gab22] and the one by O(1) Labs [SQ22, Lab23] only use one extra auxiliary modulus of the form 2^t, for a sufficiently large t. The fact that this modulus is a power of the base of representation of integers in limbs leads to important cancellations (large powers of the base are congruent to 0 modulo 2^t), which makes it possible to assert the corresponding equalities modulo 2^t. This design usually requires t to be quite large and involves many expensive range-checks. Also, note that this approach is *ad hoc* and highly specialized to the concrete emulation case.
- Polygon [LB22] uses a significantly different approach. Instead of having one big additional auxiliary modulus, they use many small ones. This design choice is driven by the fact that they use STARKs and their native modulus is relatively small (64-bits long). Not only is the contribution of the native modulus to boosting the LCM less impactful, but also the auxiliary moduli need to be significantly smaller in order to avoid wrap-arounds. To achieve a better performance, Lubarov and Baylina [LB22] consider the possibility of choosing the set of auxiliary moduli probabilistically. This allows them to use fewer auxiliary moduli than necessary to strictly enforce the desired constraints. The randomized choice of M makes it infeasible for a malicious prover to leverage the fact that the system is under-constrained. Let us highlight the generality of this method over the other *ad hoc* approaches: a single implementation (parametric on the set of auxiliary moduli) can be used for any emulation scenario.

In this work, our techniques are inspired by the approach by Lubarov and Baylina [LB22]. We observe that by computing tight bounds, the number of necessary auxiliary moduli can be dramatically reduced. In most cases just one extra auxiliary modulus is enough to *deterministically* enforce the desired constraints. On the other hand, their approach (with loose bounds) requires dozens of auxiliary moduli and only achieves probabilistic soundness.

Concretely, their techniques require analyzing equations that involve terms of the form:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}((B^{i+j} \;\%\; q) \;\%\; m)x_i y_j \ ,$$

where q is the emulated modulus, B is a constant (the base of representation in limbs form), m is an auxiliary modulus and x_i, y_i are further restricted to be in the range $[0, B)$. A simple upper-bound for the above term, used by Lubarov and Baylina, is n^2B^2m. This bound leads to the conclusion that $4n^2B^2m < p$ is sufficient for avoiding wrap-arounds in their concrete equation, here p is the native modulus.[3] Equivalently, it is enough to select $m < p/(4n^2B^2)$.

We observe that $(B^{i+j} \;\%\; q) \;\%\; m$ can be very small, depending on the value of q with respect to the base B or depending on the value of m. We argue that, considering the tight bound $(B-1)^2\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}((B^{i+j} \;\%\; q) \;\%\; m)$, the condition on m can be significantly relaxed, because the bound may be small even when m is large. Manually handling these tight bounds can be tedious and error-prone, but the benefits they offer are well worth the effort. We recommend computing them programmatically and we also employ formal verification to ensure the soundness of our conclusions.

Based on this idea, we propose a revisited and more efficient encoding of foreign-field multiplication. Using similar techniques, we also develop dedicated foreign-field identities for modeling elliptic curve operations over foreign base fields.

2 Preliminaries

Notation. For $m, n \in \mathbb{N}$ with $m \leq n$, the range $\{m, m+1, \ldots, n\}$ is denoted by $[m, n]$. We use $[n]$ as a shorthand for $[1, n]$ and $[m, n)$ as a shorthand for $[m, n-1]$. We denote by $\mathbb{Z}_n$ the ring of integers modulo n. Elements in $\mathbb{Z}_n$, unless stated otherwise, are represented canonically as integers in the range $[0, n)$. Equality modulo $n \in \mathbb{N}$ is denoted by $=_n$, i.e. for any $a, b \in \mathbb{Z}$, $a =_n b$ iff there exists $k \in \mathbb{Z}$ such that $a - b = kn$. Equality over the integers is sometimes explicitly denoted by $=_{\mathbb{Z}}$. We denote by $\%$ the *modulo operation*, that is, $n \;\%\; m$ is the remainder of dividing n by m.

PLONK's arithmetization. One of the remarkable features of PLONK is its versatile and powerful arithmetization, the model of computation in which NP relations are described.

In a nutshell, a PLONK "circuit" is a table T whose entries take values over a finite field K (the so-called *native field*). Let ℓ be the number of rows in T and m be the number of columns. Let $T_{i,j}$ be the table entry at row i and column j. In this model, NP instances are represented as partially-filled tables, whereas the NP witness of a given instance is a valid completion of the remaining entries. A completion is valid if it satisfies all the constraints imposed by the arithmetization.

These constraints, often referred to as *custom gates*, are specified through low-degree polynomial identities of the form:

$$\forall i \in [\ell].\, f(T_{i,1}, \ldots, T_{i,m}) =_K 0 \ .$$

That is, they enforce polynomial relations between the values in each row of the table. It is common to have *multi-row identities*, i.e. polynomial constraints that span over several adjacent rows. For example:

$$\forall i \in [\ell].\, f(T_{i,1}, \ldots, T_{i,m}, T_{i+1,1}, \ldots, T_{i+1,m}) =_K 0 \ .$$

(The table is cyclic, i.e. the last row has a next one: the first row.) Multi-row identities can be considered as long as the number of rows they span over is small, since it affects proof size and verifier complexity.

[3] The factor of 4 is due to the fact that the above was not the only term in the equation.

In order for this model of computation to be non-trivial, PLONK is equipped with a *permutation argument* that allows one to introduce so-called *copy-constraints* that enforce some values from different table entries to be the same. Furthermore, PLONK incorporates more advanced tools such as *lookup arguments*, which can be used to efficiently enforce constraints of the form "this table entry must take a value among this set of pre-established values".

For the purpose of understanding this work, simply note that PLONK's model of computation allows one to define custom low-degree polynomial identities over the native field, which can be enforced very efficiently across all the rows in the table. In our implementation, $K \coloneqq \mathbb{Z}_p$ for some prime p; all our identities span over 3 rows or less and their polynomial degree is at most 6. Furthermore, we use lookups for performing efficient t-bits range-checks, i.e. asserting that some table entries belong to the range $[0, 2^t)$.

3 Foreign-field arithmetic

From now on, we will denote by p the native (prime) modulus and by q the one we want to emulate, which may be any natural number (not necessarily prime). Our techniques are generic with respect to the relative size of p and q. For that, integers over $\mathbb{Z}_q$ will be represented in *limbs form* in a certain base B.

Let n be the lowest integer such that $B^n \geq q$, we represent an integer $x \in \mathbb{Z}_q$ with n limbs as $(x_{n-1}, \ldots, x_0)_B$, where $x_i \in [0, B)$ for every i, and $x = \sum_i B^i x_i$. This way, the values of x_i fit in the native field $\mathbb{Z}_p$ (if B is sufficiently small, thus n sufficiently large). As we will see, we will require that B be significantly lower than $\sqrt{p}$ in order to avoid wrap-arounds; otherwise, the upper-bound in (4) would exceed p. This means that at least 2 limbs will be necessary unless $q \ll \sqrt{p}$ (if $\sqrt{p} < q < p$, there would be enough space to represent integers in $\mathbb{Z}_q$ with native scalars over $\mathbb{Z}_p$, but there must exist extra room for "computing").

3.1 Foreign-field multiplication

Given three integers x, y, z in limbs form, we would like to assert that they are in a multiplicative relation $x \cdot y =_q z$, that is:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} B^{i+j} x_i y_j \;-\; \sum_{i=0}^{n-1} B^i z_i \;=_q\; 0 \;.$$

Note that we highlight variables (in blue) to differentiate them from constants. The above equality can be enforced with an equality over the integers by exhibiting $k \in \mathbb{Z}$ such that:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} (B^{i+j} \,\%\, q) x_i y_j \;-\; \sum_{i=0}^{n-1} (B^i \,\%\, q) z_i \;=_{\mathbb{Z}}\; k \cdot q \;. \qquad (1)$$

The left-hand side of this equation can be lower-bounded by $-nqB$ and upper-bounded by $n^2 q B^2$. These are the bounds considered in [LB22]. They are simple and elegant, but not tight. In this work, we consider the following tight bounds instead:

$$\text{lower-bound:} \quad -(B-1)\sum_{i=0}^{n-1}(B^i \,\%\, q)$$

$$\text{upper-bound:} \quad (B-1)^2 \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(B^{i+j} \,\%\, q)$$

These tight bounds can be significantly lower than their non-tight counterparts, e.g. note that $(B^{i+j} \,\%\, q)$ can be much lower than q if the base is chosen appropriately.

These bounds allow us to bound k in the range $[k_{\mathsf{min}}, k_{\mathsf{max}}]$, where k_{min} and k_{max} are the result of dividing the above lower-bound and upper-bound (respectively) by q (ignoring the remainder[4]). The value of k will be enforced (via range-checks) to be in such range in order to assert bounds on the right-hand-side of equation (1). However, the range-check protocol that we consider in this work can only assert ranges of the form $[0, 2^t)$ for $t \in \mathbb{N}$. We can transform the range $[k_{\mathsf{min}}, k_{\mathsf{max}}]$ into this form via the change of variables $u \coloneqq k - k_{\mathsf{min}}$. This way, equation (1) becomes:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(B^{i+j} \,\%\, q) x_i y_j - \sum_{i=0}^{n-1}(B^i \,\%\, q) z_i - (u + k_{\mathsf{min}}) \cdot q \;=_{\mathbb{Z}}\; 0 \;. \qquad (2)$$

Let u_{max} be the lowest power of 2 such that $u_{\mathsf{max}} > k_{\mathsf{max}} - k_{\mathsf{min}}$, we can now range-check u in $[0, u_{\mathsf{max}})$. We stress that this range-check on u is performed in order to have reliable bounds on the left-hand side of equation (2). The range on u could be larger, but that would lead to looser bounds, potentially requiring more auxiliary moduli. On the other hand, the range on u should not be smaller, because that could hinder completeness (soundness would not be affected).

We will assert equation (2) modulo p (the native modulus) and also modulo m for every auxiliary modulus in a set M. If $\mathrm{LCM}(M \cup \{p\})$ is large enough, namely it satisfies the below inequalities, then (in virtue of the Chinese Remainder Theorem) the assertion of equation (2) modulo every m in $M \cup \{p\}$ will imply that the equation holds over the integers, as desired.

$$(B-1)\sum_{i=0}^{n-1}(B^i \,\%\, q) + (u_{\mathsf{max}} + k_{\mathsf{min}}) \cdot q < \mathrm{LCM}(M \cup \{p\})$$

$$\wedge \quad (B-1)^2 \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(B^{i+j} \,\%\, q) - k_{\mathsf{min}} \cdot q < \mathrm{LCM}(M \cup \{p\}) \;.$$

3.1.1 Asserting equation (2) over an auxiliary modulus. We are ready to add constraints which enforce that equation (2) is satisfied modulo our auxiliary moduli in M. Let $m \in M$, we start by reducing the coefficients once again, this time modulo m. As before, we will exhibit an integer ℓ and enforce that:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}((B^{i+j} \,\%\, q) \,\%\, m) x_i y_j \;-\; \sum_{i=0}^{n-1}((B^i \,\%\, q) \,\%\, m) z_i$$
$$- \; u \cdot (q \,\%\, m) \;-\; (k_{\mathsf{min}} \cdot q) \,\%\, m \;-\; \ell \cdot m \;=_{\mathbb{Z}}\; 0 \;. \qquad (3)$$

We can now establish bounds on ℓ:

[4] By "ignoring the remainder" we mean taking the floor on positive numbers or the ceiling on negative ones.

$$\ell_{\min} \coloneqq -\frac{(B-1)\sum_i((B^i \,\%\, q) \,\%\, m) + u_{\max}(q \,\%\, m) + (k_{\min}\, q) \,\%\, m}{m},$$
$$\ell_{\max} \coloneqq \frac{(B-1)^2 \sum_i \sum_j((B^{i+j} \,\%\, q) \,\%\, m) \;-\; (k_{\min} \cdot q) \,\%\, m}{m}.$$

Again, by applying a shift to ℓ in order to get a range of the proper form, let $v \coloneqq \ell - \ell_{\min}$ and let $v_{\max}$ be the lowest power of 2 such that $v_{\max} > \ell_{\max} - \ell_{\min}$. We will range-check v in the range $[0, v_{\max})$. This allows us to establish the following bounds for the left-hand side of equation (3).

lower-bound:

$$-(B-1)\sum_{i=0}^{n-1}((B^i \,\%\, q) \,\%\, m) \;-\; u_{\max} \cdot (q \,\%\, m) \;-\; (k_{\min} \cdot q) \,\%\, m \;-\; (v_{\max} + \ell_{\min}) \cdot m \;,$$

upper-bound:

$$(B-1)^2 \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}((B^{i+j} \,\%\, q) \,\%\, m) \;-\; (k_{\min} \cdot q) \,\%\, m \;-\; \ell_{\min} \cdot m \;. \tag{4}$$

If p is strictly greater than both bounds in absolute value, there will be no wrap-around and equation (3) should hold over the integers, as desired.

3.1.2 Putting it all together. Given x and y in limbs form, assumed to be well-formed, i.e. whose limbs are in the range $[0, B)$, the multiplication gate between x and y is enforced as follows.

The prover will exhibit limbs z_i for the result z, as well as a value u and values v_m for every auxiliary modulus $m \in M$. Then, we enforce the following identity, corresponding to the check modulo p:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(B^{i+j} \,\%\, q)x_i y_j \;-\; \sum_{i=0}^{n-1}(B^i \,\%\, q)z_i \;-\; (u + k_{\min}) \cdot q \;=_p\; 0 \;.$$

Furthermore, for every $m \in M$, the following identity will be enforced:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}((B^{i+j} \,\%\, q) \,\%\, m)x_i y_j \;-\; \sum_{i=0}^{n-1}((B^i \,\%\, q) \,\%\, m)z_i \;-\; u \cdot (q \,\%\, m) - (k_{\min} \cdot q) \,\%\, m - (v_m + \ell_{\min}) \cdot m \;=_p\; 0 \;.$$

Finally, the following range-checks will also be enforced:

$$z_i \in [0, B) \;\; \forall i \in [0, n) \quad u \in [0, u_{\max}) \quad v_m \in [0, v_{m\,\max}) \;\; \forall m \in M \;.$$

3.1.3 Soundness and completeness of multiplication. The following theorems ensure the soundness and completeness of our encoding of foreign-field multiplication. Both theorems are stated with respect to any positive natural numbers p, q, B, n such that $B^n \geq q$, and any set of positive integers M satisfying the conditions below.

Define:

$$k_{\min} \coloneqq -\frac{(B-1)\sum_{i=0}^{n-1}(B^i \,\%\, q)}{q}$$
$$k_{\max} \coloneqq \frac{(B-1)^2 \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(B^{i+j} \,\%\, q)}{q}$$
$$\ell_{\min}(m) \coloneqq -\frac{(B-1)\sum_i((B^i \,\%\, q) \,\%\, m) + u_{\max}(q \,\%\, m) + (k_{\min}\, q) \,\%\, m}{m}$$
$$\ell_{\max}(m) \coloneqq \frac{(B-1)^2 \sum_i \sum_j((B^{i+j} \,\%\, q) \,\%\, m) \;-\; (k_{\min} \cdot q) \,\%\, m}{m}.$$

Also, let t be the lowest integer such that $k_{\max} - k_{\min} < 2^t$. Analogously, for every $m \in M$, define t_m as the lowest integer satisfying $\ell_{\max}(m) - \ell_{\min}(m) < 2^{t_m}$. We require:

$$(B-1)\sum_{i=0}^{n-1}(B^i \,\%\, q) + (2^t + k_{\min}) \cdot q \;<\; \mathrm{LCM}(M \cup \{p\})$$

$$(B-1)^2 \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(B^{i+j} \,\%\, q) - k_{\min} \cdot q \;<\; \mathrm{LCM}(M \cup \{p\}) \,.$$

And, for every $m \in M$:

$$(B-1)\sum_{i=0}^{n-1}((B^i \,\%\, q) \,\%\, m) \;+\; 2^t \cdot (q \,\%\, m) + (k_{\min} \cdot q) \,\%\, m \;+\; (2^{t_m} + \ell_{\min}(m)) \cdot m \;<\; p$$

$$(B-1)^2 \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}((B^{i+j} \,\%\, q) \,\%\, m) - (k_{\min} \cdot q) \,\%\, m \;-\; \ell_{\min}(m) \cdot m \;<\; p \;.$$

We are ready to state the following theorem on the *soundness* of foreign-field multiplication. It guarantees that the only solutions satisfying the system of constraints resulting from our encoding are precisely those that semantically correspond to multiplication modulo q.

Theorem 3.1 (Soundness of Multiplication). *For all integers* $\{x_i, y_i, z_i \in [0, B)\}_{i=0}^n$, $u \in [0, 2^t)$, *and* $\{v_m \in [0, 2^{t_m})\}_{m \in M}$ *with:*

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(B^{i+j} \,\%\, q)x_i y_j \;-\; \sum_{i=0}^{n-1}(B^i \,\%\, q)z_i \;-\; (u + k_{\min}) \cdot q \;=_p\; 0$$

and such that for all $m \in M$*:*

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}((B^{i+j} \,\%\, q) \,\%\, m)x_i y_j \;-\; \sum_{i=0}^{n-1}((B^i \,\%\, q) \,\%\, m)z_i \;-\; u \cdot (q \,\%\, m) - (k_{\min} \cdot q) \,\%\, m - (v_m + \ell_{\min}(m)) \cdot m \;=_p\; 0 \;,$$

it holds:

$$\Big(\sum_{i=0}^{n-1} B^i x_i\Big)\Big(\sum_{i=0}^{n-1} B^i y_i\Big) \;=_q\; \sum_{i=0}^{n-1} B^i z_i \;.$$

Finally, our *completeness* theorem guarantees that it is always possible to satisfy the system of constraints for values of that are in a multiplicative relation modulo q.

Theorem 3.2 (Completeness of Multiplication). *For all integers* $\{x_i, y_i, z_i \in [0, B)\}_{i=0}^{n}$, *such that*

$$\Big(\sum_{i=0}^{n-1} B^i x_i\Big)\Big(\sum_{i=0}^{n-1} B^i y_i\Big) \;=_q\; \sum_{i=0}^{n-1} B^i z_i \;,$$

there exist integers $u \in [0, 2^t)$, *and* $\{v_m \in [0, 2^{t_m})\}_{m \in M}$, *satisfying:*

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(B^{i+j} \;\%\; q)x_i y_j \;-\; \sum_{i=0}^{n-1}(B^i \;\%\; q)z_i \;-\; (u + k_{\mathsf{min}}) \cdot q \;=_p\; 0$$

and such that for all $m \in M$*:*

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}((B^{i+j} \;\%\; q) \;\%\; m)x_i y_j \;-\; \sum_{i=0}^{n-1}((B^i \;\%\; q) \;\%\; m)z_i$$
$$-\; u \cdot (q \;\%\; m) - (k_{\mathsf{min}} \cdot q) \;\%\; m - (v_m + \ell_{\mathsf{min}}(m)) \cdot m \;=_p\; 0 \;.$$

We prove both theorems in EasyCrypt. The proof will be publicly available.

3.2 Foreign-field addition, subtraction and other operations

Most references from the literature focus on describing how to implement foreign field multiplication, since addition is a simpler case that can be addressed in a similar way [Lab23].

In this work, however, we propose a novel method for performing modular addition. We will simply add two modular integers limb-wise (over the native field). This is highly efficient, as it can be performed with native constraints and does not require range-checks. Addition can be performed limb-wise, because the representation in base B is linear in the following sense:

$$\sum_{i=0}^{n-1} B^i x_i \;+\; \sum_{i=0}^{n-1} B^i y_i = \sum_{i=0}^{n-1} B^i (x_i + y_i)$$

But how about well-formedness? The resulting limbs $x_i + y_i$ may not be well-formed (they may exceed the base B). In order to address this problem, we will have a normalization procedure that takes a possibly non-well-formed modular integer and produces a well-formed one encoding the same value. This is possible as long as the limb values of the non-well-formed integer have not exceeded a certain limit (the so-called maximum limb value, L).

Implementing subtraction with this limb-wise method seems problematic, as computing $x_i - y_i$ with native constraints may wrap-around under the native zero. However, this will not be a problem because, after all, every single identity is enforced modulo p. What is important is to keep track of proper bounds on the integer value of every limb. For example, if we subtract two well-formed limbs $x_i, y_i \in [0, B)$, the resulting limb $x_i - y_i$ is known to be in the range $(-B, B)$. Our normalization procedure is applicable as long as these bounds are contained in the interval $[-L, L]$.

We note that before a multiplication or a comparison, a normalization will be necessary. However, one can perform several additions/subtractions without normalizing.

3.2.1 Normalization. Our mechanism for normalization will introduce a custom identity similar to the one for foreign-field multiplication. Given a possibly non-well-formed integer x, normalization will produce an equivalent well-formed modular integer z by asserting that $z_i \in [0, B)$ for all $i \in [0, n)$ and:

$$\sum_{i=0}^{n-1}(B^i \;\%\; q)x_i \;-\; \sum_{i=0}^{n-1}(B^i \;\%\; q)z_i \;=_{\mathbb{Z}}\; k \cdot q \;. \tag{5}$$

Assuming that we know bounds on the possibly not well-formed x_i and that these bounds fall[5] in the range $[-L, L]$, we can lower-bound the LHS of equation (5) by $-(L+B)\sum_{i=0}^{n-1}(B^i \;\%\; q)$; and upper-bound it by $L\sum_{i=0}^{n-1}(B^i \;\%\; q)$. After having bound the left-hand side, we can proceed as in Section 3.1 to enforce the identity over a set of auxiliary moduli. We omit the details here for simplicity. Since this identity is linear (it has degree 1, unlike the multiplication one), the LCM threshold will be lower and we may need fewer auxiliary moduli to enforce it. However, this depends on how big L is. For $L < B^2$, this identity can almost certainly be enforced with the same auxiliary moduli used for multiplication. This is because L is the dominant factor in the bounds for the left-hand side of equation (5) and if $L \approx B^2$, the bounds would be comparable to those of the multiplication equation (1) whose terms are of the form $(B^{i+j} \;\%\; q)x_i y_i$ (and x_i, y_i can be upper-bounded by B). Note that a bound of $L \approx B^2$ is quite generous in the sense that it allows one to perform B additions in a row before normalizing.

3.2.2 Other operations. Other functions like addition by constant or multiplication by constant can be easily enforced limb-wise. For the former, perform limb-wise addition with the limbs of the constant. For the latter, multiply every limb by the constant of interest. The limb values should never exceed L, so this method for multiplying by a constant cannot be used if the constant is greater than L/B, in that case one would need to use standard multiplication (Section 3.1). Table 1 summarizes the process of performing these limb-wise operations and how the limb bounds should be updated in each case.

Division can be modeled as a multiplication (and a non-zero assertion) and inversion can be captured similarly (where the dividend is the constant one).

3.2.3 Comparisons. Observe that, unless the emulated modulus is a perfect power of the base B, there will be values that admit at least two different representations in limbs form. This is not a problem for the custom gates developed so far, they work flawlessly with the double representation. However, when comparing two modular integers, extra care is needed. We cannot simply compare the integers limb-wise, as two different sets of limbs could be encoding the same modular integer. A simple way of performing sound comparisons (when the emulated modulus is prime) is to exhibit a modular inverse of the difference of two integers. Such inverse would exist if and only if they are really different. This, however, requires a subtraction, a normalization and a multiplication.

3.2.4 The unique-zero representation technique. When q is such that $B^n < 2q$, there exist integers with only one representation. One such element is $q-1$ (also known as -1). In this case, we suggest the following modification to the encoding in order to speed-up comparisons with 0, which are highly important, as they are the basis for other functions like division or (dis)equalities.

[5] An integer whose limbs exceed L in magnitude can no longer be normalized with this identity.

Table 1: Simple modular operations that can be performed limb-wise. Each operation takes as input x in limbs form (and y or κ when applicable) and produces a new modular integer z. Limb lower/upper bounds are respectively annotated with an under/upper bar over the limb.

Operation	Limb-wise computation	New limb bounds	
Addition	$z_i = x_i + y_i$	$\underline{z}_i = \underline{x}_i + \underline{y}_i$	$\bar{z}_i = \bar{x}_i + \bar{y}_i$
Subtraction	$z_i = x_i - y_i$	$\underline{z}_i = \underline{x}_i - \bar{y}_i$	$\bar{z}_i = \bar{x}_i - \underline{y}_i$
Negation	$z_i = -x_i$	$\underline{z}_i = -\bar{x}_i$	$\bar{z}_i = -\underline{x}_i$
Addition of constant	$z_i = x_i + \kappa_i$	$\underline{z}_i = \underline{x}_i + \kappa_i$	$\bar{z}_i = \bar{x}_i + \kappa_i$
Multiplication by constant	$z_i = x_i \cdot \kappa$	$\underline{z}_i = \underline{z}_i \cdot \kappa$ if $\kappa \geq 0$, else $\bar{z}_i \cdot \kappa$.	$\bar{z}_i = \bar{z}_i \cdot \kappa$ if $\kappa \geq 0$, else $\underline{z}_i \cdot \kappa$.

Instead of encoding integers as before, let limbs $(x_{n-1}, \ldots, x_0)$ represent integer $1 + \sum_i B^i x_i$. This way, 0 inherits the unique representation of -1 of our previous encoding, allowing for limb-wise comparisons with zero. (Because the representation of 0 is now unique, a well-formed integer is zero iff all its limbs are the limbs of 0.)

This subtle change would introduce variations in the multiplication identity:

$$\left(1 + \sum_{i=0}^{n-1} B^i x_i\right)\left(1 + \sum_{i=0}^{n-1} B^i y_i\right) - \left(1 + \sum_{i=0}^{n-1} B^i z_i\right) =_q 0 \ ,$$

now expands to

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} B^{i+j} x_i y_j + \sum_{i=0}^{n-1} B^i (x_i + y_i) - \sum_{i=0}^{n-1} B^i z_i =_q 0 \ .$$

Observe the extra $\sum_i B^i(x_i+y_i)$ term in the identity, with respect to equation (1). The normalization identity remains the same as the extra +1 of x gets canceled with the extra (negated) +1 of $-z$.

This new encoding would also affect all functions that operate limb-wise. They would need to be slightly modified in order to account for this new representation. For example, addition would become limb-wise addition $z_i = x_i + y_i$ for all limbs, except for the least-significant limb, $z_0 = x_0 + y_0 + 1$, which would include an extra shift of +1.

4 Foreign-field elliptic curve operations

We now consider the problem of emulating group operations on an elliptic curve E defined over finite field $\mathbb{Z}_q$, with equation $y^2 =_q x^3 + ax + b$. We assume that the curve (or the relevant subgroup) has no low-order points. In particular, that there are no points of order 2 or 3. It will be explicit in the analysis of *double* why this assumption is needed. Note that this assumption is satisfied by all curves of cryptographic interest, which define a cyclic group of (large) prime order.

We could leverage the addition and multiplication modulo q that we have developed in Section 3 in order to enforce constraints for elliptic curve addition, doubling, etc. For example, for point addition $R \coloneqq P + Q$, we could perform:

$$\lambda \coloneqq \frac{y_Q - y_P}{x_Q - x_P} \qquad x_R \coloneqq \lambda^2 - x_Q - x_P \qquad y_R \coloneqq \lambda(x_P - x_R) - y_P \ .$$

This would require 5 subtractions, 5 normalizations, 3 multiplications and 1 non-zero assertion. Similarly, for point doubling $R \coloneqq 2P$, we could perform:

$$\lambda \coloneqq \frac{3\,{x_P}^2 + a}{2\,y_P} \qquad x_R \coloneqq \lambda^2 - 2\,x_P \qquad y_R \coloneqq \lambda\,(x_P - x_R) - y_P \ .$$

Nevertheless, we can do better. Given how versatile the approach from Section 3 is, we suggest to implement elliptic curve operations in an alternative way. We will define foreign-field custom gates enforcing the following identities:

$$\begin{aligned}
\text{curve membership} \qquad & {y^2} =_q x^3 + a\,x + b \\
\lambda\text{-slope assertion} \qquad & y_Q - y_P =_q (x_Q - x_P)\,\lambda \\
\lambda\text{-tangent assertion} \qquad & 3\,{x_P}^2 + a =_q 2\,y_P\,\lambda \\
\lambda^2 \text{ assertion} \qquad & \lambda^2 =_q x_P + x_Q + x_R
\end{aligned}$$

4.1 Implementing the custom identities

Unlike the two custom identities from Section 3, (multiplication and normalization), which have degree 2 and 1 (respectively), the custom identity for *curve membership* has degree 3. This is undesirable because it may require a larger FFT domain and because it would involve a cubic number of cross-limb products. This problem can be mitigated by introducing a third variable z, representing (and enforced elsewhere to be equal to) x^2. This will reduce its degree to 2.

We describe the four identities in detail. We do not use the *unique-zero representation* here, but note that the identities could be easily adapted to this alternative representation. The identities (here expressed with $=_q$) can be enforced with equations modulo p as in Section 3: by exhibiting an integer k, the quotient of the corresponding expression (resulting of moving all terms to the left-hand side) on division by q, which gives an equation over the integers. Such equation over the integers can be enforced by verifying it with respect to sufficiently many auxiliary moduli. We omit the full details here (and keep the $=_q$ description of the identities) for the sake of space and simplicity.

4.1.1 Identity for curve membership. This foreign-field identity asserts curve membership of a point (x, y) as long as z is instantiated with a well-formed limbs-representation of x^2 (a condition that can be enforced with an additional multiplication identity).

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} B^{i+j} y_i y_j - \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} B^{i+j} x_i z_j - \sum_{i=0}^{n-1} a B^i x_i - b =_q 0 \ . \quad (6)$$

4.1.2 Identity for λ-slope assertion. This foreign-field identity asserts that the slope between points P and Q equals λ. (As long as $x_P \neq_q x_Q$.) Note that this identity does not depend on the curve parameters (a, b). Furthermore, this identity does not guarantee that the points differ in the x-coordinate (this may be enforced elsewhere). Here, y_{Q_i} represents the i-th limb of the limb-representation of the y-coordinate of Q. Other values are defined analogously. Variable $\mu \in \{-1, 1\}$ can be used to flip the sign of the first summand, in which case the equation asserts that the slope between P and $-Q$ equals λ. This way, this identity can be used for checking λ with respect to both the inputs and the output of an addition.

$$\mu \sum_{i=0}^{n-1} B^i y_{Q_i} - \sum_{i=0}^{n-1} B^i y_{P_j} - \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} B^{i+j}(x_{Q_i} - x_{P_i})\lambda_j =_q 0 \ . \tag{7}$$

4.1.3 Identity for λ-tangent assertion. This foreign-field identity asserts that λ is the slope of the tangent to the curve at point P. (As long as P is in the curve.)

$$3\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} B^{i+j} x_{P_i} x_{P_j} + a - 2\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} B^{i+j} y_{P_i}\lambda_j =_q 0 \ . \tag{8}$$

4.1.4 Identity for λ^2 assertion. This foreign-field identity asserts that, under the assumption λ is the correct slope between points P and Q, then x_R is the correct x-coordinate of the addition $P + Q$.

$$\sum_{i=0}^{n-1} B^i x_{P_i} + \sum_{i=0}^{n-1} B^i x_{Q_j} + \sum_{i=0}^{n-1} B^i x_{R_j} - \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} B^{i+j}\lambda_i\lambda_j =_q 0 \ . \tag{9}$$

The above identities will be further multiplied by a Boolean term that allows us to "disable" the check conditionally on its value. Note that this term does not have an impact in the magnitude of the bounds for the foreign-field identities.

In the following, for the sake of simplicity we will write:

(a) $(x^3 + ax + b - y^2)\,\sigma =_q 0$ as a shorthand for equation (6),
(b) $(\mu\, y_Q - y_P - \lambda(x_Q - x_P))\,\sigma =_q 0$ as a shorthand for equation (7),
(c) $(3{x_P}^2 + a - 2y_P\lambda)\,\sigma =_q 0$ as a shorthand for equation (8),
(d) $(x_P + x_Q + x_R - \lambda^2)\,\sigma =_q 0$ as a shorthand for equation (9),

all of them conditioned on bit $\sigma \in \{0, 1\}$.

4.2 Witnessing a point

We will represent a point with two emulated modular integers (x, y) and a Boolean value β indicating whether the point is the identity. This mean, a point is represented with $2n + 1$ PLONK native values, where n is the number of limbs necessary to express x (or y) in the chosen base. We have the convention that if $\beta =_p 1$, the point is the identity, regardless of the values of (x, y). This way, we can enforce (with PLONK constraints) that a point is valid by asserting that:

$$(xz + ax + b - y^2)(1 - \beta) =_q 0 \qquad \text{and} \qquad x^2 - z =_q 0 \ ,$$

where z is an auxiliary modular integer. The former is the actual foreign-field identity that we will implement for curve membership. The latter can be expressed with the foreign-field multiplication identity from Section 3.1.

Double$(P \coloneqq (x_P, y_P, \beta_P))$:

Require: P is on the curve or it is the identity point

Ensure: $R = 2P$

1: exhibit the result (x_R, y_R, β_R) ▷ $2P$
2: exhibit slope λ ▷ $(3{x_P}^2 + a)/\,2y_P$ (or arbitrary if $P = \mathcal{O}$)
3: assert $\beta_R =_p \beta_P$
4: assert $(3{x_P}^2 + a - 2y_P\lambda)(1 - \beta_P) =_q 0$
5: assert $(x_P + x_P + x_R - \lambda^2)\,(1 - \beta_P) =_q 0$
6: assert $(-y_R - y_P - \lambda(x_R - x_P))(1 - \beta_P) =_q 0$
7: return $R \coloneqq (x_R, y_R, \beta_R)$

Figure 1: Point doubling enforced through the custom identities from Section 4.1.

In curves like BLS12-381, where the group of interest is a subgroup of the underlying curve, it may be necessary to perform further checks that guarantee that the witnessed point belongs to the desired subgroup. These checks can be performed with a scalar multiplication by a constant (the algorithm from Figure 4 can be optimized when the scalars are constant) and only need to be performed once, when "loading a point", as all point operations are closed in the subgroup.

4.3 Point doubling

Figure 1 describes how point doubling can be enforced through our foreign-field identities.

The assertion in Step 3 is justified by our assumption that there are no order-2 points in the curve (or at least in the relevant cryptographic subgroup), which guarantees that the result of doubling is identity point iff the input is the identity point.

The remaining assertions are dedicated to the case where P is not the identity. (Note that all of them are disabled if $\beta_P =_p 1$.) Step 4 guarantees that λ is the slope of the tangent of the curve at P (which is on the curve by assumption). Once λ is correctly constrained, Step 5 ensures that x_R is the correct x-coordinate of $2P$. Finally, Step 6 constrains y_R such that the slope between P and $(x_R, -y_R)$ equals λ, as desired. Importantly, Step 6 requires $x_P \neq x_R$, because we are modeling a division as a multiplication (so the denominator cannot be zero). Note that $x_P \neq x_R$ is guaranteed by our assumption that there do not exist order-3 points, thus $2P \neq -P$. That, combined with the fact that at this "branch" we have $P \neq \mathcal{O}$, implies $2P \neq \pm P$, and thus $x_R \neq x_P$, as desired. Observe that assertions (4)-(6) combined implicitly imply that the constrained coordinates (x_R, y_R) satisfy the curve equation, or $\beta_R = 1$.

4.4 Point addition

Figures 2 and 3 describe how point addition can be enforced through our foreign-field identities. The former is incomplete, but more efficient.

IncompleteAdd($P \coloneqq (x_P, y_P, \beta_P), Q \coloneqq (x_Q, y_Q, \beta_Q)$):

Require: P, Q satisfy the curve equation ($P, Q \neq_q \mathcal{O}$) and $x_P \neq x_Q$

Ensure: $R = P + Q$

1: exhibit the result (x_R, y_R, β_R) ▷ $P + Q$
2: exhibit slope λ ▷ $(y_Q - y_P) \,/\, (x_Q - x_P)$
3: assert $\beta_R =_p 0$
4: assert $y_Q - y_P - \lambda(x_Q - x_P) =_q 0$
5: assert $x_P + x_Q + x_R - \lambda^2 =_q 0$
6: assert $-y_R - y_P - \lambda(x_R - x_P) =_q 0$
7: return $R \coloneqq (x_R, y_R, \beta_R)$

Figure 2: Point incomplete addition enforced through the identities from Section 4.1.

CompleteAdd($P \coloneqq (x_P, y_P, \beta_P), Q \coloneqq (x_Q, y_Q, \beta_Q)$):

Require: P, Q are on the curve or the identity point

Ensure: $R = P + Q$

1: exhibit the result (x_R, y_R, β_R) ▷ $P + Q$
2: assert $\beta_R(1 - \beta_R) =_p 0$
3: assert $\beta_P =_p 1 \implies R = Q$
4: assert $\beta_Q =_p 1 \implies R = P$
5: assert $P = -Q \implies \beta_R =_p 1$
6: assert $P = Q \;\wedge\; \beta_P, \beta_Q =_p 0 \implies R = \mathsf{Double}(P)$
7: assert $x_P \neq_q x_Q \wedge \beta_P, \beta_Q =_p 0 \Rightarrow R = \mathsf{IncompleteAdd}(P, Q)$
8: return $R \coloneqq (x_R, y_R, \beta_R)$

Figure 3: Point incomplete addition enforced through the identities from Section 4.1.

4.4.1 Incomplete addition. The routine from Figure 2 is incomplete in the sense that it requires the given points not be the identity nor share the same x-coordinate.

These requirements can be enforced with PLONK constraints elsewhere if necessary, or may be skipped if a certain invariant guarantees that they are met. In particular, adding constraints that enforce P (respectively Q) is not the identity comes "for free" in PLONK, as this can be encoded via "copy-constraints" between β_P (respectively β_Q) and a PLONK native value with a constant value of 0. On the contrary, the check $x_P \neq_q x_Q$ will have some cost, so it is preferable to skip it when possible. The cost of this check can be lowered if the *unique-zero representation* of modular integers is used.

The assertion in Step 3 guarantees that the resulting point will not be the identity. This preserves completeness given our preconditions. Step 4 guarantees that λ is the slope of the line intersecting P and Q (which are on the curve by assumption). Once λ is correctly constrained, Step 5 ensures that x_R is the correct x-coordinate of $P + Q$. Finally, Step 6 constrains y_R such that the slope between P and $(x_R, -y_R)$ equals λ, as desired. Observe that these three assertions combined implicitly imply that the constrained coordinates (x_R, y_R) satisfy the curve equation.

4.4.2 Complete addition. The routine from Figure 3 implements complete point addition. It is significantly more costly than its incomplete counterpart, so it should only be used when strictly necessary.

Steps 3 and 4 model the cases where one of the inputs is the identity point. Constraint $\beta_P =_p 1 \implies R = Q$ is a shorthand for a *conditional assertion*, which can be modeled with the following equations:

$$\beta_P(x_R - x_Q) =_q 0, \quad \beta_P(y_R - y_Q) =_q 0 \quad \text{and} \quad \beta_P(\beta_R - \beta_Q) =_p 0 \;,$$

where, in turn, $\beta_P(x_R - x_Q) =_q 0$ is short for $\beta_P(x_{Ri} - x_{Q_i}) =_p 0$ for every limb i. Steps 5 and 6 model the case where the inputs share the same x coordinate. Here, $P = -Q$ represents a Boolean variable that is 1 iff $x_P - x_Q =_q 0$ and $y_P + y_Q =_q 0$. Finally, when none of the above cases applies, all the preconditions of IncompleteAdd apply.

4.5 Multi-scalar multiplication

We now use the above ingredients for implementing (multi-)scalar multiplication via double-and-add.

Given the high cost of complete additions, we would like to use incomplete additions when possible. However, the preconditions of incomplete addition may not hold in the middle of the double-and-add loop. A common technique to overcome this challenge [DH20] is to use a random point as the initial value of the double-and-add accumulator. Such point is freely chosen by the prover. Importantly, the probability that a precondition (of incomplete addition) is violated in the loop is negligible over the choice of this point, which makes this method statistically complete. On the downside, for soundness, we must subtract the (scaled) random point from the result of the MSM and constraining such correction term to be the scaled version of the initial accumulator can be quite expensive.

We propose a novel idea that makes the random term in the accumulator invariant throughout the loop. This prevents it from scaling, thus significantly simplifying the final correction step. Let bases $P_1, \ldots, P_\ell \in E(\mathbb{Z}_q)$ and scalars $s_1, \ldots, s_\ell \in \mathbb{Z}$. We will compute $\sum_j s_j P_j$ via a windowed double-and-add loop with w-bits windows. Let the prover select a random point U and set the initial accumulator of the double-and-add loop to be $\mathsf{acc} \coloneqq \ell U$. Also, define $V \coloneqq (2^w - 1)\, U$.[6] On the i-th iteration of the double-and-add loop, we will double acc w times and then, for every $j \in [\ell]$, add $s_{j,i} P_j - V$, where $s_{j,i} \in [0, 2^w)$ is the i-th (little-endian) window of scalar s_j.[7] After this, the random term in the accumulator becomes:

$$2^w(\ell U) - \ell V \;=\; 2^w(\ell U) - \ell(2^w - 1)U \;=\; \ell U \;,$$

which makes the random term invariant (ℓU) at the beginning of every iteration.

[6] The consistency of acc (resp. V) with respect to U needs to be enforced with constraints, but this is relatively cheap given that ℓ (resp. $2^w - 1$) is constant and small.

[7] The values $k P_j - V$ can be tabulated for every $k \in [0, 2^w)$, see Step 4 of Figure 4.

Table 2: Cost of encoding a multi-scalar multiplication of ℓ SECP256k1 points (by full-size scalars) emulated over BLS12-381 or BN254, with different emulation parameters.

Emulation parameters (for SECP256k1 base field)			PLONK circuit architecture		PLONK proof size	
# of limbs n	Base B	Auxiliary moduli M	# of columns	# of rows for ℓ-MSM	BLS12-381	BN254
3	2^{86}	$\{2^{86},\ 2^{86}-1\}$	8	$5600\ell + 7300$	4.92 KB	4.40 KB
4	2^{64}	$\{2^{128}\}$	9	$\boxed{3900\ell + 5000}$	5.09 KB	4.56 KB
5	2^{52}	$\{2^{156}\}$	11	$4060\ell + 5500$	5.37 KB	4.81 KB
6	2^{44}	$\{2^{172}\}$	13	$4360\ell + 5850$	5.65 KB	5.06 KB

MultiScalarMul($\{P_j \coloneqq (x_{P_j}, y_{P_j}, \beta_{P_j})\}_{j\in[\ell]}, \{s_{j,i}\in\mathbb{Z}\}_{j\in[\ell]}^{i\in[T]}$):

Require: P_j is on the curve (and $P_j \neq \mathcal{O}$) $\forall j \in [\ell]$

Ensure: $R = \sum_j s_j P_j$, where $s_j \coloneqq \sum_i 2^{w(i-1)} s_{j,i}$, $\forall j \in [\ell]$

1: choose a point U ▷ uniformly, for statistical completeness
2: assert $\beta_U =_p 0$ and ${x_U}^3 + a x_U + b - {y_U}^2 =_q 0$
3: set $V \coloneqq (2^w - 1)\, U$
4: set $\mathsf{Table}_j \coloneqq [-V,\ P_j - V,\ 2P_j - V,\ \ldots,\ (2^w-1)P_j - V]$ for every $j \in [\ell]$
5: set acc $\coloneqq \ell\, U$
6: **for** $i = 1$ to T **do**
7: **for** _ $= 1$ to w **do** ▷ double
8: set acc $\coloneqq$ Double(acc)
9: **for** $j = 1$ to ℓ **do** ▷ and add
10: set aux $\coloneqq \mathsf{Table}_j[s_{j,i}]$ ▷ enforced with a lookup
11: assert $x_{\text{acc}} \neq_q x_{\text{aux}}$
12: assert acc $\neq \mathcal{O}$
13: set acc $\coloneqq$ IncompleteAdd(acc, aux)
14: set $R \coloneqq$ CompleteAdd(acc, $-\ell U$)
15: return R

Figure 4: Multi-scalar multiplication. Scalars are encoded as T little-endian integer windows in the range $[0, 2^w)$, which *in-circuit* can be represented as native values, since $2^w \ll p$.

The full algorithm is described in Figure 4. The tables in Step 4 can be computed with incomplete addition by adding P_j to the previous table element (the first being $-V$). It can be shown that all the preconditions of IncompleteAdd are satisfied across the table computation as long as $x_{P_j} \neq_q x_V$, which may be asserted explicitly, only once per base P_j. Similarly, multiplications by a small constant term like in Steps 3 and 5 can be mainly computed with incomplete addition. We denote by $-V$ the opposite of V, which can be obtained by negating the y coordinate of V, i.e., as $(x_V, -y_V, \beta_V)$, see Section 3.2 for details about foreign-field negation (over $\mathbb{Z}_q$). Importantly, the last addition in Step 14 must be performed with CompleteAdd as nothing prevents the final result R from being the identity point.

Note that the algorithm presented in Figure 4 is compatible with further optimizations. An example is the the well-known optimization based on the the GLV endomorphism [GLV01], which we have also implemented in our library.

5 Formal Verification

We recognize that our approach for FFA presented in Section 3 is involved and mission critical at the same time. In such situations doing manual pen-and-paper proofs might lead to oversights and subtle mistakes in the implementation. As a result, we decided to use EasyCrypt theorem prover to implement our definitions and formally verify our main results.

More specifically, we formalized correctness of our approach by splitting it into soundness and completeness theorems as presented in Section 3.1.3. In our formalization [AFQ25]. we reused the EasyCrypt's multi limb library which was initially developed for the Jasmin workbench. This considerably simplified the development and with the use of built-in SMT solvers we managed to tame the complexity of the proofs so that the entire formalization is about 1K LOC.

We are currently working on formalizing the correctness of the ECC algorithms from Section 4.

6 Performance Evaluation

In this section, we present the experimental evaluation of our approach through a series of case studies. All benchmarks were conducted on a commodity laptop, a MacbookPro with *Apple M3 Pro* chip and 16GB of RAM.

Our implementation, which will be made publicly available, is built on the PSE's fork of Halo2 [ECC21], utilizing the KZG commitment scheme [GWC19]. The emulation method is designed to be highly flexible, allowing parameterization over various emulation curves and settings, such as limb size and the number of limbs, which influence the circuit architecture. Additionally, our implementation is parametric over the "parent curve" (the curve on which the proof is generated) enabling its reuse for generating proofs that can be verified using precompiles in Ethereum [But13], Cardano [DGKR18], Solana [Yak18], and other platforms.

We begin by introducing the circuit architecture for different emulation parameters, specifically for SECP256k1, and analyze the cost in terms of rows, columns, and proof size for proving the

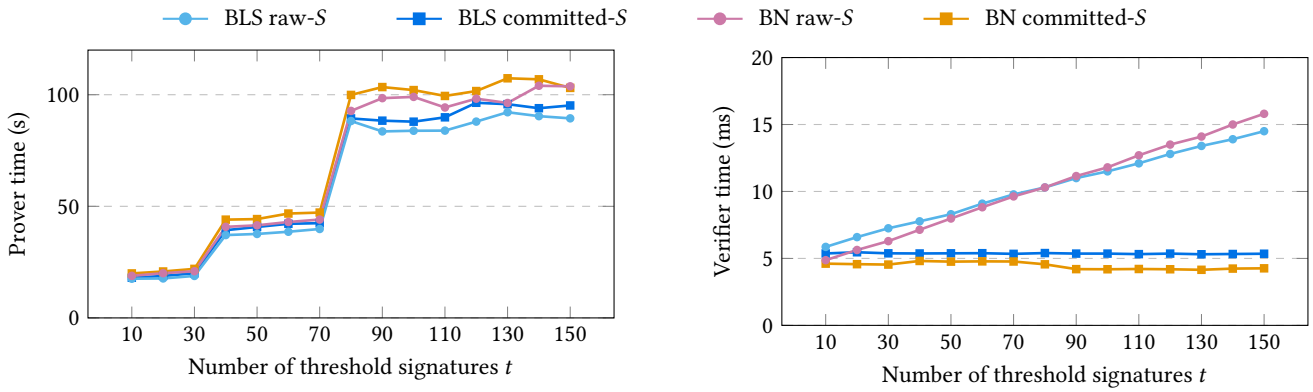

Figure 5: Prover and verifier times of a proof of knowledge of t-out-of-$2t$ ECDSA signatures (over SECP256k1) on a public message; emulated over BLS12-381 and BN254. We evaluate two approaches: (i) the set S of verifying keys is passed directly as public inputs, (ii) S is passed in committed form (as its Poseidon hash).

execution of a multi-scalar multiplication (MSM). Next, we examine the verification of ECDSA signatures over the SECP256k1 curve and extend our approach to support threshold ECDSA signatures. We then evaluate the performance of self-recursion by executing the PLONK verifier within a circuit, emulating curve operations using the techniques outlined in this work. All our benchmarks are conducted a KZG-based PLONK where the underlying curve is BLS12-381 and BN254, to assess performance across different settings.

In the following benchmarks, all our custom identities span over at most 3 rows[8] and their polynomial degree is at most 6. We use lookups for performing efficient t-bits range-checks. In particular, we run 4 parallel lookups, which allow us to assert that 4 table entries (per row) belong to the range $[0, 2^t)$, for some row-dependent $t \leq 16$.

6.1 Emulation parameters

Table 2 presents a comparison of various emulation configurations for performing SECP256k1 multi-scalar multiplications (MSMs) emulated over the scalar field of BLS12-381 and the scalar field of BN254. Notably, in this case, the emulation parameters are the same in both emulation scenarios. This is because (for the same emulation scenario) the choice of emulation parameters is primarily dictated by the size of the native field and, in this case, the size of both scalar fields is about 32 bytes.

Our analysis indicates that the most efficient configuration consists of 4 limbs (utilizing 9 columns) and requires $3900\ell + 5000$ rows for an MSM of size ℓ. Given this, we adopt the 4-limb configuration for all subsequent benchmarks to ensure optimal performance while maintaining consistency across evaluations.

6.2 Threshold ECDSA over SECP256k1

In this section we showcase the performance of our techniques with benchmarks on proving knowledge of ECDSA signatures on a public message. Recall the definition of ECDSA signatures, defined over an elliptic curve (SECP256k1 in this case) of order p with designated generator G. For key generation, one samples a secret key $\mathsf{sk} \leftarrow_\$ \mathbb{Z}_p$ and defines the verifying key as $\mathsf{vk} \coloneqq \mathsf{sk}\, G$.

$\underline{\mathsf{ECDSA.Sign}}(\mathsf{sk}, \mathsf{msg})$:
- $k \leftarrow_\$ \mathbb{Z}_p$
- $K \coloneqq k\, G$
- $r \coloneqq K_x \bmod p$
- $s \coloneqq k^{-1}(\mathsf{Hash}(\mathsf{msg}) + \mathsf{sk} \cdot r)$
- return (r, s)

$\underline{\mathsf{ECDSA.Verify}}(\mathsf{vk}, \mathsf{msg}, (r, s))$:
- assert $r, s \in [0, p)$
- $u_1 \coloneqq \mathsf{Hash}(\mathsf{msg}) \cdot s^{-1} \pmod p$
- $u_2 \coloneqq r \cdot s^{-1} \pmod p$
- $K \coloneqq u_1\, G + u_2\, \mathsf{vk}$
- accept iff $r = K_x \bmod p$

To isolate the impact of the foreign-field arithmetic in ECDSA verification, we benchmark a scenario where the message hash is treated as a public input, eliminating the need for SHA-256 computations *in-circuit*. We implement *ad-hoc threshold multi-signatures* (ATMS) as described by Gaži, Kiayias, and Zindros [GKZ18]. ATMS enable threshold signatures without a setup ceremony, where an untrusted third party can generate the threshold public key, and threshold signatures can be produced non-interactively. The relation being proved is:

$$\mathrm{PoK}\left\{ \{\sigma_i, \mathsf{vk}_i\}_{i=1}^{t} : \forall i \in [t], \begin{array}{l} \mathsf{ECDSA.Verify}(\mathsf{vk}_i, \mathsf{msg}, \sigma_i) = 1 \\ \mathsf{vk}_i \in S \ \wedge\ \mathsf{vk}_i \neq \mathsf{vk}_j\ \forall j \in [t] \setminus i \end{array} \right\},$$

where S is a set of k designated verifying keys. This relation can be described as a proof of knowledge of *t-out-of-k signatures on a public message* msg.

Naturally, the set S and the message msg are *public inputs* to the underlying circuit. This makes the verifier complexity linear in $k = |S|$. In order to have constant verification complexity, we consider another version of the same predicate where set S is a witness, which is publicly quantified as a commitment to it (computed with the SNARK-friendly Poseidon hash function [GLR+20]). Namely:

$$\mathrm{PoK}\left\{ (S, \{\sigma_i, \mathsf{vk}_i\}_{i=1}^{t}) : \begin{array}{l} \mathsf{Poseidon}(S) = \mathsf{com}_S \\ \mathsf{ECDSA.Verify}(\mathsf{vk}_i, \mathsf{msg}, \sigma_i) = 1\ \ \forall i \\ \forall i,\ \mathsf{vk}_i \in S \ \wedge\ \mathsf{vk}_i \neq \mathsf{vk}_j\ \forall j \neq i \end{array} \right\},$$

[8] The number of columns is minimized in order to meet this condition. We typically have $2n{+}1$ columns where n is the number of limbs for the emulation, which is enough.

Table 3: Benchmarks on self-recursion over BN254 and BLS12-381, emulating foreign-field operations with our techniques.

Curve	Proof size	Verifier	Prover	# rows	# cols
BLS12-381	6.10 KB	11 ms	35 s	241 K	15
BN254	4.71 KB	9 ms	29 s	177 K	11

we coin this the *committed-S* version of the predicate.

Figure 5 presents the performance of the prover and verifier for the two version of our threshold ECDSA predicate, The experiments are conducted over BLS12-381 and BN254 as the underlying KZG curves.

The "hops" in prover time are due to the need of a new larger power-of-2 domain (in order to perform FFTs, the number of rows in PLONK is always padded to the next power of 2). Also, observe how the committed-*S* version is slightly more costly for the prover, as the predicate involves an extra Poseidon hash. On the other hand, the verifier time is constant in the committed-*S* version and linear otherwise, as expected.

This example showcases a powerful application of our techniques. It allows one to emulate e.g. Bitcoin signatures over frameworks that do not support the SECP256k1 curve natively. Furthermore, it provides a rich threshold policy for signatures schemes that may not allow for this by design, and it enjoys constant verification complexity on the number of verified signatures.

6.3 Self-recursion

Self-recursion is the concept of "verifying yourself", i.e. building a SNARK circuit that involves one or several copies of its own SNARK verifier. Self-recursion can be used for proof composition, incrementally verifiable computation [Val08] or proof-carrying data [BCMS20], with multiple applications like scalable rollups or even succinct blockchains.

We implement self-recursion using our techniques for encoding the foreign-field elliptic curve operations from the verifier circuit. We use the techniques introduced by Bowe, Grigg, and Hopwood [BGH19, BCMS20] that move part of the verifier logic "out of circuit" via an accumulator scheme (in our case, for the KZG polynomial commitment scheme). This allows us to ignore the pairing in the recursive logic and defer its evaluation until the very end (the final *off-circuit* verifier). A detailed explanation of these techniques is beyond the scope of this section, and we refer the reader to the original works for a comprehensive discussion. In our experiments, we consider a small *application function* that simply increases a counter, thus most of the logic corresponds to the *in-circuit* recursive verifier. We evaluate the proving times over BLS12-381 and BN254 as the underlying curves. BLS12-381 is self-emulated with parameters $B = 2^{56}$, $n = 7$ and $M = \{2^{136}, 2^{136}-2\}$, whereas BN254 is self-emulated with $B = 2^{52}$, $n = 6$ and $M = \{2^{142}\}$. Table 3 summarizes our results. We highlight that proving a full recursive step only requires about half a minute in both emulation scenarios.

Implementing recursion by emulating foreign-field arithmetic is elegant, conceptually simpler and less error-prone than other approaches based on cycles of curves [ECC21, BMRS20]. It leads to very short proofs and is applicable to scenarios where the there is a built-in proving system that cannot be altered.

References

[AFQ25] Miguel Ambrona, Denis Firsov, and Iñigo Querejeta-Azurmendi. Soundness and completeness of ffa, 2025. https://github.com/input-output-hk/efficient-ffa/releases/tag/sound-and-complete-ffa. Accessed on April 16, 2025.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Berlin, Heidelberg, August 2013.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 1–18. Springer, Cham, November 2020.

[BDPV13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, Berlin, Heidelberg, May 2013.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019.

[BMRS20] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Mina: Decentralized cryptocurrency at scale, 2020. https://docs.minaprotocol.com/static/pdf/technicalWhitepaper.pdf. Accessed on April 16, 2025.

[But13] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2013. https://ethereum.org/en/whitepaper/. Accessed on April 16, 2025.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Cham, April / May 2018.

[DH20] Ying Tong Lai Daira Hopwood. Deep dive on halo 2, 2020. https://raw.githubusercontent.com/daira/halographs. Accessed on April 16, 2025.

[ECC18] Electric Coin Company ECC. Jubjub: A twisted Edwards curve for zk-SNARKs, 2018. https://zips.z.cash/protocol/protocol.pdf. Accessed on April 16, 2025.

[ECC21] Electric Coin Company ECC. Halo2: A plonk-based proof system, 2021. https://github.com/zcash/halo2. Accessed on April 16, 2025.

[Gab22] Ariel Gabizon. Aztec emulated field and group operations, 2022. https://hackmd.io/@relgabizon/B13JoihA8. Accessed on April 16, 2025.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Berlin, Heidelberg, May 2013.

[GKL+22] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Reinforced concrete: A fast hash function for verifiable computation. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1323–1335. ACM Press, November 2022.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021.

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A faster version of the poseidon hash function. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 177–203. Springer, Cham, July 2023.

[GKZ18] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. Cryptology ePrint Archive, Report 2018/1239, 2018.

[GLR+20] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a generalization of substitution-permutation networks: The HADES design strategy. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 674–704. Springer, Cham, May 2020.

[GLV01] Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 190–200. Springer, Berlin, Heidelberg, August 2001.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016.

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

[GYB21] Christopher Goes, Awa Sun Yin, and Adrian Brink. Anoma: Undefining money: A protocol for private, asset-agnostic digital cash and n-party bartering, 2021. https://anoma.network/papers/whitepaper.pdf. Accessed on April 16, 2025.

[HBHW] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specifiation. https://zips.z.cash/protocol/protocol.pdf. Accessed on April 16, 2025.

[Hop20] Daira Hopwood. The Pasta curves for Halo 2 and beyond, 2020. https://electriccoin.co/blog/the-pasta-curves-for-halo-2-and-beyond. Accessed on April 16, 2025.

[Hop21] Daira Hopwood. Pluto-Eris: a half-pairing cycle of elliptic curves, 2021. https://github.com/daira/pluto-eris. Accessed on April 16, 2025.

[Lab23] O(1) Labs. Foreign field addition rfc (mina book), 2023. https://o1-labs.github.io/proof-systems/kimchi/foreign_field_add.html. Accessed on April 16, 2025.

[LB22] Daniel Lubarov and Jordi Baylina Melé. Casting out primes: Bignum arithmetic for zero-knowledge proofs. Cryptology ePrint Archive, Report 2022/1470, 2022.

[Mat23] Matter Labs. *zkSync Documentation*, 2023. https://docs.zksync.io/. Accessed on April 16, 2025.

[Mid24] Midnight Network. Nightpaper: A litepaper introducing midnight, 2024. https://midnight.network/whitepaper. Accessed on April 16, 2025.

[MKF21] Toghrul Maharramov, Dmitry Khovratovich, and Emanuele Francioni. The dusk network whitepaper, 2021. https://dusk.network/uploads/The_Dusk_Network_Whitepaper_v3_0_0.pdf. Accessed on April 16, 2025.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

[Pol23] Polygon Labs. *Polygon zkEVM Documentation*, 2023. https://docs.polygon.technology/zkEVM/. Accessed on April 16, 2025.

[SQ22] Joseph Spadavecchia and Anaïs Querol. Non-native field and group operations, 2022. https://hackmd.io/XZUHHGpDQsSOs0dUGugB5w. Accessed on April 16, 2025.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, March 2008.

[Wil18] Zachary J. Williamson. Aztec network (white paper), 2018. https://github.com/AztecProtocol/AZTEC/blob/master/AZTEC.pdf. Accessed on April 16, 2025.

[Yak18] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain, 2018. https://solana.com/solana-whitepaper.pdf. Accessed on April 16, 2025.