Fast elliptic curve scalar multiplications in SN(T)ARK circuits

Liam Eagen[1], Youssef El Housni[2], Simon Masson[3], and Thomas Piellard[2]

[1] Alpen Labs
liameagen@protonmail.com
[2] Linea
youssef.elhousni@consensys.net
thomas.piellard@consensys.net
[3] ZKNox
simon.masson@protonmail.com

Keywords: SNARK, STARK, proof systems, elliptic curves, scalar multiplication

Abstract. Proof systems of arbitrary computations have found many applications in recent years. However, the proving algorithm has a consequent complexity tied to the size of the computation being proved. Thus, proving large computations is quite inefficient. One of these large computations is the scalar multiplication over an elliptic curve. In this work, we provide new techniques for reducing the time corresponding to proving a scalar multiplication, using integer lattice reduction or a (half) extended Euclidean algorithm in a ring of integers. We investigate optimizations in the case of small (complex multiplication) discriminant curves, and its generalization for multi scalar multiplications as used in signature verification. We provide an optimized Golang implementation for different elliptic curves in different proof systems settings. The speed-up in proving time is between 22% and 53% compared to the previous state-of-the-art.

1 Introduction

Over the past few years, proof systems have become an essential primitive for privacy-preserving and scalable applications in blockchains. A proof system is an interactive protocol in which one party, referred to as the prover, seeks to convince another party, known as the verifier, of the truth of a given statement. Proof systems are classified as non-interactive when no communication is required between the prover and the verifier beyond the transmission of the proof itself. Within the category of non-interactive proofs, particularly notable concepts for demonstrating computational integrity are the Succinct Non-interactive ARgument of Knowledge (SNARK) and Scalable Transparent ARgument of Knowledge (STARK). SNARKs and STARKs offer computationally sound proofs that are efficient to verify and compact in size relative to the statement being proved. Proof systems were first introduced in [18] and have since been extensively studied in both theoretical and practical contexts [20,25,17,5]. Recent advancements

have explored a wide range of settings, including cryptographic assumptions, asymptotic efficiency, implementation performance, and diverse applications. In this work, we use the terms SNARK and STARK interchangeably, as the techniques presented are agnostic to the specific choice of proof system. In the context of blockchain technology, the computations being proved are often large, making the proving algorithm computationally expensive and a critical target for optimization. One prominent example of such a resource-intensive computation is scalar multiplication over an elliptic curve. This operation is fundamental to applications such as blockchain rollups[1], Verkle tries[2], account abstraction[3], and proof recursion [8,9,27]. While efficient computation of scalar multiplication over elliptic curves is a well-established problem in cryptography, proving scalar multiplication using a SNARK introduces a novel challenge that paves the way for new research directions and implementation strategies.

Contributions. In this work, we investigate optimizations for proving scalar multiplications with a SNARK. Given the pre-computed resulting point Q (called a hint), and leveraging lattice reduction in $\mathbb{Z}$, we show how to turn proving a n-bit scalar multiplication $[k]P \stackrel{?}{=} Q$ in a curve E into mainly proving a $n/2$-bit double-scalar-multiplication $[k_1]P - [k_2]Q \stackrel{?}{=} 0_E$, regardless of the existence of an efficient endomorphism. When it exists, we show how to turn proving a n-bit scalar multiplication into mainly a $n/4$-bit quadruple-scalar-multiplication. Alternatively to using a lattice reduction in $\mathbb{Z}$, we propose a novel use in cryptography of the (half) extended Euclidean algorithm (rational reconstruction) in an order $\mathcal{O}$ of an Euclidean imaginary quadratic field, e.g. Eisenstein integers for elliptic curves of j-invariant 0. This alternative algorithm speeds up the witness generation (scalar decomposition) at the cost of slowing down the proof generation (increasing the decomposed sub-scalars bounds). This is useful in use-cases where a prover cannot invoke arbitrary hints. Finally, we provide a fast implementation in Golang using the `gnark` library for different elliptic curves in different proof systems settings. The circuit size speed-up from this work is summarized in Table 1.

Circuit	**Curve**	**Groth16 speed-up**	**PLONK speed-up**
Emulated	small discriminant	24%	42%
	generic	52%	50%
Native	small discriminant	48%	53%
	generic	22%	28%

Table 1: Scalar multiplication constraint speed-up using our work.

[1] `https://ethereum.org/en/developers/docs/scaling/zk-rollups/`.
[2] `https://ethereum.org/en/roadmap/verkle-trees/`.
[3] `https://ethereum.org/en/roadmap/account-abstraction/`.

Organization of the paper. In Section 2, we explain what are SNARK circuits and how to optimize them and we recall basics about elliptic curves and (multi) scalar multiplications. We also recall facts about the decomposition of scalars, useful for the results of Section 3, which is the core of this paper. We provide some comparison of the new circuits with the previous state-of-the-art implementation in Section 4, and conclude this work in Section 5.

2 Background

The results of this paper are related to the cost of elliptic curve scalar multiplication in SNARK circuits. In this section, we explain what are SNARK circuits and recall results about elliptic curves, in particular for simple and multi scalar multiplications. We present two techniques for decomposing scalars from a general point of view. These techniques allow the well-known GLV decomposition [16], that splits a scalar into two sub-scalars in order to reduce the cost of a scalar multiplication for small discriminant curves.

2.1 SNARK circuits

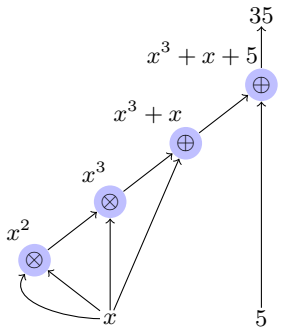
Given a public non-deterministic polynomial (NP) program F and inputs b, c, such that the program F satisfies the relation $F(b) = c$, a SNARK consists in proving this relation succinctly. It consists mainly of the `Prove` and `Verify` algorithms. The first algorithm requires generating an execution trace of the program F. This step is called witness generation. In this paper, we are interested in optimizing the `Prove` algorithm for the program F corresponding to a scalar multiplication over an elliptic curve. We do this by expressing F as an arithmetic circuit as efficiently as possible (low number of gates). This step is called *the arithmetization* of the program F.

Our techniques are valid for any SNARK (or STARK) but we implemented our work (see Section 4) for two widely used schemes: Groth16 [19] and PLONK [13]. Considering w the number of wires, m the number of multiplication gates, a the number of addition gates, ℓ = number of public inputs and $\mathtt{M}_{\mathbb{G}}$ a multiplication in $\mathbb{G}$, the cost of the `Prove` algorithm is given by $(3m + w - \ell)$ $\mathtt{M}_{\mathbb{G}}$ for Groth16, and $9(m + a)$ $\mathtt{M}_{\mathbb{G}}$ for PLONK. Note that we omit the number of FFTs in this estimation, as $\mathbb{G}$ multiplications dominate the overall cost of `Prove`. The better we arithmetize F, the faster the `Prove` algorithm is.

Arithmetization. The first step in proving an arbitrary program is to *arithmetize* it, that is to reduce the computation satisfiability to an intermediate representation satisfiability. Many problems in cryptography can be expressed as the task of computing some polynomials. Arithmetic circuits are the most standard model for studying the complexity of such computations.

An arithmetic circuit over the field $\mathbb{F}$ and the set of variables $X = \{x_0, \ldots, x_n\}$ is a directed acyclic graph such that the vertices are called gates, while the edges are called wires. Arithmetic circuits of interest to many proof systems and most

Fig. 1: Arithmetic circuit encoding the computation $x^3 + x + 5 = 35$ for which the (secret) solution is $x = 3$.



applicable to this work are those with two incoming wires and one out-coming wire (cf. Fig. 1 for an example).

SNARK-friendly computations. Many proof systems (e.g. Groth16 and PLONK) arithmetize programs as arithmetic circuits. The number of gates is what determines the prover complexity. For example, Groth16 prover complexity is dominated by some multi scalar multiplications in $\mathbb{G}$ of sizes m (the number of multiplication gates). With this in mind, multiplications by constants in $\mathbb{F}$, which are usually expensive in hardware, are essentially free. While more traditional hardware-friendly computations (e.g. XORing 32-bit numbers) are far more costly in this model. The following two observations, noted in earlier works [21], are the key to lower the number of gates of a circuit:

- The multiplication by constants in $\mathbb{F}$ is free,
- The verification can be sometimes simpler than forward computation. The circuits do not always have to compute the result, but can instead represent a verification algorithm. For example a multiplicative inversion circuit $(1/x \stackrel{?}{=} y)$ does not have to arithmetize the computation of the inversion $(1/x)$ but can instead consist of a single multiplication gate $(x \cdot y)$ on the value provided (pre-computed) by the prover (y) and checks the equality $(x \cdot y \stackrel{?}{=} 1)$.

This is basically a computation model where inversions cost as much as multiplications.

2.2 Elliptic curves

In this work, E is an ordinary elliptic curve defined over $\mathbb{F}_p$ for a large characteristic p. From Hasse theorem [29, page 138], the $\mathbb{F}_p$-rational points form a finite abelian group of order $\#E(\mathbb{F}_p) = p + 1 - t$ where t is the trace of the curve and satisfies $|t| \leq 2\sqrt{p}$. We consider a subgroup of prime order r on $E(\mathbb{F}_p)$, i.e. r divides $p + 1 - t$. In other words, let $P \in E(\mathbb{F}_p)$ a point of prime order r and

$\mathbb{G}$ the cyclic subgroup generated by P. We define the scalar multiplication as $[k]P$ for an integer $k < r$. In this work, N denotes the bit-size of r, and D the Complex Multiplication (CM) discriminant associated to E, i.e. the discriminant of the order $\mathrm{End}(E)$, the endomorphism ring of the curve as an order in the integer ring of $\mathbb{Q}(\sqrt{t^2 - 4p})$. We refer to [29] for a deeper introduction to the theory of elliptic curves.

2.3 Computing scalar multiplications

Single scalar multiplication. A scalar multiplication is the computation of $[k]P = P + \ldots + P$ (k times). The double-and-add technique given in Algorithm 1 has a complexity linear in N, the bit-size of the scalar k. We provide here an algorithm reading bits from the Most Significant Bit (MSB) to the Least Significant Bit (LSB).

Algorithm 1: ScalarMultiplication(k, P)

Input. A scalar k of N bits,
A point P.
Output. $Q = [k]P$.

$R \leftarrow P$
for b bit of k read from second MSB to LSB **do**
  $R \leftarrow [2]R$
  if $b = 1$ **then**
    $R \leftarrow R + P$
  end if
end for
return R

In cryptographic schemes where k is a secret, this algorithm is often computed in constant-time, and the conditional $R \leftarrow R + P$ is implemented consistently. In the context of SNARK circuits, conditional branching is not possible and an addition is always computed at every step of the loop. In total, the cost of a scalar multiplication is given by $(N-1)\mathbf{Add} + (N-1)\mathbf{Dbl}$.

Multi scalar multiplication. We consider two points P_1 and P_2, and two scalars k_1, k_2. In order to compute $[k_1]P_1 + [k_2]P_2$, it is possible to scan the bits of both scalars simultaneously and compute the result at the cost of one scalar multiplication (and an extra pre-computation). This improvement was originally proposed in [30] and [12], and it can be generalized to a multi scalar multiplication with n points, but the pre-computation cost increases. We provide Algorithm 2 for the generic case of n scalar multiplication.

Algorithm 2: MultiScalarMultiplication$((k_1, P_1), \ldots, (k_n, P_n))$

Input. A list of scalars $k_1, \ldots, k_n$ of N bits,
A list of points $P_1, \ldots, P_n$.
Output. $Q = [k_1]P_1 + \ldots + [k_n]P_n$.

Pre-computation table T.
for j between 0 and $2^n - 1$ **do**
  $T[j] \leftarrow \sum_{i=1}^{n} [e_i]P_i$ where $j = \sum_{i=1}^{n} e_i 2^{i-1}$.
end for

Main loop.
$R \leftarrow 0_E$
for $(b_1, \ldots, b_n)$ bits of $k_1, \ldots, k_n$ read from MSB to LSB **do**
  $R \leftarrow [2]R$
  $R \leftarrow R + T\left[\sum_{i=1}^{n} b_i 2^{i-1}\right]$
end for
return R

The cost of this algorithm is split into two: a pre-computation table of $2^n - n - 1$ elliptic curve additions (**Add**), and a main loop with $N - 1$ steps including a doubling (**Dbl**) and an addition. Adding the neutral element may occur with probability 2^{-n}, but in the context of SNARK circuits (as well as for constant time algorithms), additions will always be computed so we decided to keep $(N-1)$**Add** for the cost estimation. From now, we denote $\mathrm{MSM}(n, N)$ to be the cost of a multi scalar multiplication of n points and scalars of N bits:

$$\mathrm{MSM}(n, N) = (2^n - n - 1 + N - 1) \cdot \mathbf{Add} + (N - 1) \cdot \mathbf{Dbl}.$$

We recall in the next section a technique that speeds up a scalar multiplication by computing it as a multi scalar multiplication with halve size scalars. It was introduced in [16] by Gallant, Lambert and Vanstone.

GLV scalar multiplication. Let $\mathrm{End}(E) = \mathbb{Z} + \psi\mathbb{Z}$, and $\psi(P) \equiv [\lambda]P$, i.e. λ is the eigenvalue of ψ seen as a linear application on $E(\mathbb{F}_p)[r]$, the order-r subgroup defined over $\mathbb{F}_p$. The method introduced in [16] turns $[k]P$ into $[k_1]P + [k_2]\psi(P)$ with ψ efficiently computable. This applies to elliptic curves with small CM discriminant, where the endomorphism ψ has a tiny degree. The scalars k_1, k_2 can be constructed of size around $\sqrt{r}$. We dig the details of this decomposition from a generic point of view in Section 2.4. Note that for computing the multi scalar multiplication, it is required to compute $[\lambda]P = \psi(P)$, and it is expensive when the degree of ψ is large. The GLV technique is implemented in many libraries, and leads to almost 40% speed-up for scalar multiplication.

Example 1. Consider an elliptic curve defined over $\mathbb{F}_p$ by $y^2 = x^3 + b$. There is an efficient endomorphism $\psi : E \to E$ defined by $(x, y) \mapsto (\omega x, y)$ (and $0_E \mapsto 0_E$)

such that $\psi(P) = [\lambda]P$, where both ω and λ are cubic roots of unity in $\mathbb{F}_p$ and $\mathbb{F}_r$ respectively. Evaluating this endomorphism costs only one multiplication over $\mathbb{F}_p$. Using ψ, it is possible to turn $[k]P$ into $[k_1]P + [k_2]\psi(P)$ which costs $\mathrm{MSM}(2, N/2)$.

In the next sections, we dig the decomposition of a scalar from a general point of view. The GLV method can be obtained from it as a special case, and the results of Section 3 also use this technique.

2.4 Scalar decomposition using lattice reduction

In this section, we consider the generic decomposition of a scalar k. We look for a small solution $(x, y, z, t) \in \mathbb{Z}^4$ of

$$x + \lambda y - k(z + \lambda t) = 0 \bmod r,$$

where λ is fixed. Note that when λ is the eigenvalue of the endomorphism, finding a solution with $z = 1$ and $t = 0$ leads to the GLV decomposition.

We look for *small* solutions to this problem. The set of solutions is a lattice of small dimension, and it is possible to use algorithms of reduction in order to find *small* solutions. Minkowski's theorem ensure that in a lattice $\mathcal{L}$ of rank n, there exists a vector v of norm $|v|_\infty \leq \mathrm{Vol}(\mathcal{L})^{1/n}$, where $\mathrm{Vol}(\mathcal{L})$ is the volume of the lattice and can be computed as $\sqrt{\det(MM^t)}$ for a matrix M defining the lattice. If the lattice is described by a full rank matrix, the volume is simply the determinant. Finding a short vector of the lattice can be computed in polynomial time using for instance [23], but the obtained vector can be slightly larger. The obtained bound is $\mathrm{Vol}(\mathcal{L})^{1/n}/(\delta - 1/4)^{(n-1)/n}$, where δ is a parameter usually set to a value close to 1, often $\delta = 0.99$. However, we consider here only dimension up to 6 and algorithms such as LLL [23] work also well for this range of dimensions [26]. Though, the obtained vectors are short with respect to the Euclidean norm. While this computation is done out-of-circuit during the witness generation, it might be expensive depending on the context (see Section 4.1). We consider an alternative approach in Section 2.5 for a witness/proof generation trade-off that is easier to implement. We consider now specific values of the parameter λ that will be useful in Section 3.

Integer fraction decomposition. When $\lambda = 0$, the equation is simply $x - kz = 0 \bmod r$. It corresponds to a fraction decomposition of $k = x/z$ in $\mathbb{Z}/r\mathbb{Z}$. In this case, the solutions form a lattice of dimension 2 defined with

$$\begin{pmatrix} r & 0 \\ k & 1 \end{pmatrix}$$

After reducing this matrix, the resulting scalars are expected to be bounded by $r^{1/2}/(\delta - 1/4)^{1/2} \approx 1.16 r^{1/2}$ for $\delta = 0.99$. This case is studied in Section 3.1.

Simultaneous fraction decomposition. It is possible to solve two equations simultaneously, namely for two scalars k_1, k_2, we consider the equations $x_1 - k_1 z = 0 \bmod r$ and $x_2 - k_2 z = 0 \bmod r$. In other words, we write k_1 and k_2 as fractions modulo r with the same denominator z, and it corresponds to a lattice of dimension 3 defined with

$$\begin{pmatrix} r & 0 & 0 \\ 0 & r & 0 \\ k_1 & k_2 & 1 \end{pmatrix}$$

and the resulting scalars are expected to be bounded by $r^{2/3}/(\delta - 1/4)^{2/3} \approx 1.22 r^{2/3}$ for $\delta = 0.99$. This case is studied in Section 3.2.

Fraction decomposition in a quadratic extension. The solutions of the original equation $x + \lambda y - k(z + \lambda t) = 0 \bmod r$ is a lattice of dimension 4 defined with

$$\begin{pmatrix} r & 0 & 0 & 0 \\ -\lambda & 1 & 0 & 0 \\ k & 0 & 1 & 0 \\ 0 & 0 & -\lambda & 1 \end{pmatrix}$$

and a solution corresponds to a rational fraction $k = (x + \lambda y)/(z + \lambda t) \bmod r$, i.e. a fraction in $(\mathbb{Z}/r\mathbb{Z})[\lambda]$. Using a lattice reduction, the scalars are expected to be bounded by $r^{1/4}/(\delta - 1/4)^{3/4} \approx 1.25 r^{1/4}$ for $\delta = 0.99$. This case is studied in Section 3.3.

Simultaneous fraction decomposition in a quadratic extension. Generalizing this to two equations simultaneously is also possible. The two equations $(x_1 + \lambda y_1) - k_1(z + \lambda t) = 0 \bmod r$ and $(x_2 + \lambda y_2) - k_2(z + \lambda t) = 0 \bmod r$ define a lattice of dimension 6 defined by

$$\begin{pmatrix} r & 0 & 0 & 0 & 0 & 0 \\ -\lambda & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & r & 0 & 0 & 0 \\ 0 & 0 & -\lambda & 1 & 0 & 0 \\ k_1 & 0 & k_2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -\lambda & 1 \end{pmatrix}$$

and the resulting scalars are expected to be bounded by $r^{2/6}/(\delta - 1/4)^{5/6} \approx 1.28 r^{1/3}$. We study this case in Section 3.4.

In the next section, we investigate how to compute a scalar decomposition without lattice reduction. Instead, we consider the extended Euclidean algorithm in different rings. First, we recall how it works in $\mathbb{Z}$, and then apply the same idea on the norm Euclidean rings $\mathbb{Z}[j]$ and $\mathbb{Z}[\sqrt{-2}]$.

2.5 Scalar decomposition using the (half) extended Euclidean algorithm

A fraction decomposition can be obtained using the extended Euclidean algorithm (EEA), as described in [16, Sec. 4].

Decomposition in $\mathbb{Z}$. Running the extended Euclidean algorithm on (r, k) in $\mathbb{Z}$ produces a sequence of numbers s_i, t_i, r_i, starting with

$$s_0 = 1, t_0 = 0, r_0 = r,$$
$$s_1 = 0, t_1 = 1, r_1 = k,$$

and verifying

1. $s_i r + t_i k = r_i$
2. $r_i > r_{i+1} \geq 0$ for $i \geq 0$,
3. $|s_i| < |s_{i+1}|$ for $i \geq 1$,
4. $|t_i| < |t_{i+1}|$ for $i \geq 0$,
5. $r_i|t_{i+1}| + r_{i+1}|t_i| = r$ for $i \geq 1$.

Stopping the algorithm at the last step m where $r_m \geq \sqrt{r}$ ensures that $|t_{i+1}| < \sqrt{r}$. Since $r_{m+1} < \sqrt{r}$, we obtain $(x, z) := (r_{m+1}, t_{m+1})$ such that $x - zk = 0 \bmod r$, and $|(x, z)|_\infty < \sqrt{r}$.

This method allows in fact to find a short vector in the lattice $\mathcal{L}$, the kernel of

$$\mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{Z}/r\mathbb{Z}$$
$$(x, y) \longrightarrow x + ky$$

Note that the short vector that is found is below the Minkowski bound $\mathrm{Vol}(\mathcal{L})^{1/2} = \sqrt{r}$.

Decomposition in a norm Euclidean imaginary quadratic ring. In an imaginary quadratic ring $\mathcal{O}$ which is norm Euclidean (meaning that we can run the Greatest Common Divisor (GCD) algorithm using the norm $\mathcal{N}$ in the complex sense), if an ideal (r) splits as a product of two distinct prime ideals I_1 and I_2, we can apply the same idea as in $\mathbb{Z}$ to find a short vector in the kernel (which is a lattice $\mathcal{L}$) of

$$\mathcal{O} \times \mathcal{O} \longrightarrow \mathcal{O}/I_1 \cong \mathbb{Z}/r\mathbb{Z}$$
$$(x, y) \longrightarrow x + ky$$

Running the extended Euclidean algorithm on (r, k) in $\mathcal{O}$ produces a sequence of values s_i, t_i, r_i, starting with

$$s_0 = 1, t_0 = 0, r_0 = r,$$
$$s_1 = 0, t_1 = 1, r_1 = k,$$

and verifying

1. $s_i r + t_i k = r_i$
2. $\mathcal{N}(r_i) > \mathcal{N}(r_{i+1}) \geq 0$ for $i \geq 0$,
3. $\mathcal{N}(s_i) < \mathcal{N}(s_{i+1})$ for $i \geq 1$,
4. $\mathcal{N}(t_i) < \mathcal{N}(t_{i+1})$ for $i \geq 0$,
5. $\pm(r_i t_{i+1} - r_{i+1} t_i) = r$ for $i \geq 1$.

As in $\mathbb{Z}$, if we stop the algorithm at the last step m where $\mathcal{N}(r_m) \geq \sqrt{\mathcal{N}(r)} = \sqrt{r}$, we can find a short vector (t_{m+1}, r_{m+1}) satisfying $r_{m+1} - t_{m+1}k = 0 \bmod r$, where $\mathcal{N}(r_{m+1}) \leq \sqrt{\mathcal{N}(r)}$. From $\pm(r_{i-1}t_i - r_i t_{i-1}) = r$ for $i \geq 1$ and $\mathcal{N}(r_{i-1}) > \mathcal{N}(r_i)$ we have

$$\mathcal{N}(r) \geq \mathcal{N}(r_{i-1})\mathcal{N}(t_i) - \mathcal{N}(r_i)\mathcal{N}(t_{i-1}) \geq \mathcal{N}(r_{i-1})(\mathcal{N}(t_i) - \mathcal{N}(t_{i-1}))$$

Since $\mathcal{N}(r_i) \geq \sqrt{r}$ for $i \geq m$, we have for $i \geq m$

$$\sqrt{\mathcal{N}(r)} \geq (\mathcal{N}(t_i) - \mathcal{N}(t_{i-1}))$$

Since this inequality is true for all the first m steps and since $t_0 = 0$, summing all those inequalities gives

$$c\sqrt{\mathcal{N}(r)} \geq \mathcal{N}(t_i)$$

where c is the number of steps before we stop the algorithm. The number of steps when running the extended Euclidean algorithm on (a, b) with $\mathcal{N}(a) > \mathcal{N}(b)$ is at worst $\log_2(\mathcal{N}(a))$.

The bound $\sqrt{r}$ is far from the theoretical Minkowski bound $\mathrm{Vol}(\mathcal{L})^{1/4} = \sqrt[4]{r}$. To remedy to this, instead of running the extended Euclidean algorithm on (r, k), we can run it on (w, k) where w is a short vector in I_1, of size below $\sqrt{r}$, which is possible since I_1, I_2 are lattices of rank 2 and volume r. In this case, running the extended Euclidean algorithm on (w, k) would produce (t_{m+1}, r_{m+1}) such that

$$\mathcal{N}(t_{m+1}) < c\sqrt{\mathcal{N}(w)}$$
$$\mathcal{N}(r_{m+1}) < \sqrt{\mathcal{N}(w)}$$

and

$$r_{m+1} - t_{m+1}k = 0 \bmod w$$

and since w is I_1, this gives that

$$r_{m+1} - t_{m+1}k = 0 \bmod I_1$$

Decomposition in $\mathbb{Z}[j]$. In the ring of Eisenstein integers, the ideal (r) splits as the product of $I_1 := (r, j - \lambda)$ and $I_2 := (r, j - \lambda^2)$. To find a short vector w in I_1, we apply the half-GCD method for the decomposition in $\mathbb{Z}$ on (r, λ). It produces a vector (w_0, w_1) such that

$$w := w_0 + jw_1 \in I_1$$

since in $\mathbb{Z}[j]/(r, j - \lambda)\mathbb{Z} \cong \mathbb{Z}/r\mathbb{Z}$, $w = 0$. Also $|(w_0, w_1)|_\infty < \sqrt{r}$.

Now, running the extended Euclidean algorithm on (w, k) produces $x, y \in \mathbb{Z}[j]$ such that

$$x - kz = 0 \bmod (r, j - \lambda)$$

and such that $\mathcal{N}(x) < \sqrt{\mathcal{N}(w)}$ and $\mathcal{N}(z) < N\sqrt{\mathcal{N}(w)}$ where N is the bit-size of k.

For implementation purposes, we need to know the bit-size of the scalars x and z. For that we need to compare the norm $\mathcal{N}$ and maximum norm defined by $|a + jb|_\infty = \max\{|a|, |b|\}$. The comparison of those norms is described in the Appendix A. We obtain that

$$|x|_\infty \leq 2\sqrt{2}\sqrt[4]{r},$$
$$|z|_\infty \leq 2\sqrt{2}N\sqrt[4]{r}.$$

Decomposition in $\mathbb{Z}[\sqrt{-2}]$. When r splits in $\mathbb{Z}[\sqrt{-2}]$ as $I_1 := (r, \sqrt{-2} - \mu)$, $I_2 := (r, \sqrt{-2} - \nu)$, we proceed exactly in the same way as in $\mathbb{Z}[j]$, that is we first find a short vector $w \in I_1$ such that $|w|_\infty < \sqrt{r}$, and then we run the extended Euclidean algorithm on (w, k) in $\mathbb{Z}[\sqrt{-2}]$ to find (x, z) such that $x - kz = 0 \bmod (r, r - \mu)$ and such that $\mathcal{N}(x) < \sqrt{\mathcal{N}(w)}$ and $\mathcal{N}(z) < N\sqrt{\mathcal{N}(w)}$ where N is the bit-size of k. Again, comparing the norm $\mathcal{N}$ and the maximum norm defined by $|a + b\sqrt{-2}|_\infty = \max\{|a|, |b|\}$ (see Appendix A) shows that

$$|x|_\infty \leq 2\sqrt[4]{2}\sqrt[4]{r},$$
$$|z|_\infty \leq 2\sqrt[4]{2}N\sqrt[4]{r}.$$

3 Proving scalar multiplications

In this section, we explain how to apply scalar decomposition in order to speed up proving scalar multiplications in SNARKs. In the traditional context, when an efficient endomorphism is available, one can turn a single scalar multiplication i.e. $\mathrm{MSM}(1, N)$ into a $\mathrm{MSM}(2, N/2)$. In Sections 3.1 and 3.2, we present a technique to turn any $\mathrm{MSM}(1, N)$ into a $\mathrm{MSM}(2, N/2)$ regardless if the elliptic curve is equipped with an efficient endomorphism. In Sections 3.3 and 3.4, when the curve is equipped with a GLV endomorphism, we show how to turn a $\mathrm{MSM}(1, N)$ into a $\mathrm{MSM}(4, N/4)$.

3.1 Hinted simple scalar multiplication

In the context of SNARKs, the scalar multiplication $[k]P$ is rather *verified* than computed. In this context it is possible to hint the output $Q = [k]P$, that is the prover computes the scalar multiplication outside of the SNARK circuit. Then the verifier checks whether if $[k]P - Q = 0_E$ inside the SNARK circuit. This way, it is possible to write k as $x/z \bmod r$ with x, z small as in Section 2.4.

$$[k]P = Q \iff [x]P - [z]Q = 0$$

This computation can be done with a double scalar multiplication as in Section 2.3. From Section 2.4, the resulting scalars x, z are expected to be bounded by $1.16\sqrt{r}$. However, the verifier has to additionally check that the decomposition is correct, i.e. $k \cdot z = x \bmod r$.

In the traditional context of computing $[k]P$, the GLV endomorphism is a way to introduce a second point $\phi(P)$ which is cheap to compute to turn $[k]P$ into $[k_1]P + [k_2]\phi(P)$ where k_1, k_2 are small. In the SNARK context, we use the resulting point Q as this second point to achieve the same speed-up. In terms of cost, this technique enables a verification in almost $\mathrm{MSM}(2, N/2)$. We note that a similar idea for verifying signatures was introduced in [2] in a different context. In the next section, we investigate how to use this technique for a double scalar multiplication.

Example 2. Consider the following toy elliptic curve: $E/\mathbb{F}_{103} : y^2 = x^3 + 5$. It forms a group of prime order 97. Let $P = (38, 94)$ be a point on the curve. Given $Q = [11]P = (95, 27)$, proving $[11]P = Q$ is equivalent to proving $[2]P - [9]Q = 0_E$.

3.2 Hinted double scalar multiplication

The previous technique can be generalized for a *double* scalar multiplication. The computation of $[k_1]P_1 + [k_2]P_2$ can be decomposed using the simultaneous fraction decomposition of Section 2.4 in order to write $k_1 = x_1/z \bmod r$ and $k_2 = x_2/z \bmod r$. Then,

$$[k_1]P_1 + [k_2]P_2 = Q \iff [x_1]P_1 + [x_2]P_2 - [z]Q = 0$$

and the scalars x_1, x_2 and z are expected to be bounded by $1.22r^{2/3}$. The computation of the MSM of size 2 can be done using a MSM of size 3 with scalar of size almost $2/3$ of the original scalar size. We estimate that this leads to a 25% improvement for a signature verification. In the next section, we investigate the case of GLV optimization.

Example 3. Let $P_1 = (38, 94)$ and $P_2 = (31, 82)$ be two points on the curve $E(\mathbb{F}_{103})$. Given $Q = [75]P_1 + [35]P_2 = (8, 38)$, proving $[75]P_1 + [35]P_2 = Q$ is equivalent to proving $[-18]P_1 + [11]P_2 + [8]Q = 0_E$.

3.3 Hinted GLV scalar multiplication

It is possible to use the technique described in Section 3.1 together with the GLV speed-up. Instead of writing $k = x/z \bmod r$, we decompose the scalar in $(\mathbb{Z}/r\mathbb{Z})[\lambda]$ as in Section 2.4:

$$k = \frac{x + \lambda y}{z + \lambda t} \bmod r$$

leading to a four-dimensional multi scalar multiplication:

$$[k]P = Q \iff [x]P + [y]\psi(P) - [z]Q - [t]\psi(Q) = 0$$

Using a lattice reduction, a solution (x, y, z, t) can be found with expected norm bounded by $1.25\sqrt[4]{r}$. Finally, we obtain an overall cost of almost MSM$(4, N/4)$, improving the scalar multiplication verification for small discriminant curves.

Example 4. The curve $E(\mathbb{F}_{103})$ has the same form as Example 1. Hence, it has an efficient endomorphism ψ with $\lambda = 61$ and $\omega = 46$. Let $P = (38, 94) \in E$. Given $Q = [56]P = (8, 65)$, proving $[56]P = Q$ is equivalent to proving $[1]P + [1]\psi(P) + [1]Q - [1]\psi(Q) = 0_E$.

3.4 Hinted GLV double scalar multiplication

An approach similar to Section 3.2 can be combined with the GLV optimization. As in Section 2.4, we write

$$k_1 = \frac{x_1 + \lambda y_1}{z + \lambda t} \bmod r, \qquad k_2 = \frac{x_2 + \lambda y_2}{z + \lambda t} \bmod r$$

and so

$$[k_1]P_1 + [k_2]P_2 = Q \iff [x_1]P_1 + [y_1]\psi(P_1) + [x_2]P_2 + [y_2]\psi(P_2) - [z]Q - [t]\psi(Q) = 0.$$

From Section 2.4, we expect x_i, y_i, z, t to be bounded by $1.28r^{1/3}$. The final cost is almost MSM(6, N/3), slightly better than MSM$(4, N/2)$ when the double scalar multiplication is computed using the GLV optimization.

Example 5. Let $P_1 = (38, 94)$ and $P_2 = (31, 82)$ in $E(\mathbb{F}_{103})$. Given $Q = [16]P_1 + [92]P_2 = (57, 2)$, proving $[16]P_1 + [92]P_2 = Q$ is equivalent to proving $[2]P_1 + [-1]\psi(P_1) + [1]P_2 + [-2]\psi(P_2) - [2]Q - [1]\psi(Q) = 0_E$.

Note 1. In this section, we only focused on elliptic curves defined over $\mathbb{F}_p$. Galbraith, Lin, and Scott [14,15] generalized the GLV method for a large class of elliptic curves over $\mathbb{F}_{p^2}$, referred to as GLS curves. Our technique applies similarly to this generalization. It can turn proving a N-bit scalar multiplication over a GLS curve into proving a $N/8$-bit multi scalar multiplication of size 8.

Note 2. The difference between our method and the (traditional) GLV method is threefold:

1. In GLV, for the scalar decomposition, we usually start by pre-computing a short basis using a lattice reduction given the curve order r and the endomorphism eigenalue λ. This is a one-time setup per elliptic curve. Then we find a close vector to the scalar k in the lattice using Babai rounding. In our case, it is the lattice reduction that gives us directly the scalar decomposition. The reduction takes k and r as inputs and has to be performed for every new scalar multiplication.
2. In GLV, the scalar decomposition (Babai rounding) is part of the running time of the scalar multiplication algorithm. In our case, the SNARK circuit only verifies that the (pre-computed) decomposition is correct.
3. In 2-dimensional GLV, we need an efficient endomorphism. In our case, we don't. In 4-dimensional GLV, we need two endomorphisms. In our case, we only need one.

4 Implementation results

For the implementation, we choose two widely used proof systems: Groth16 and PLONK. As shown in Section 2.1, we are interested in reducing the number of gates of the arithmetic circuit corresponding to a scalar multiplication. For Groth16, we only consider multiplication gates and for PLONK we consider both addition and multiplication gates.

4.1 Considerations

Arithmetizing computations for SNARK proving is different than traditionally carrying plain computations as shown in Section 2.1.

Arithmetizing elliptic curve operations. Since inversions cost as much as multiplications here, we use affine coordinates to double and add points on the elliptic curve in short Weierstrass form. When a doubling is followed by an addition i.e. $[2]R + P$ we instead compute $(R + P) + R$ omitting the computation of the y-coordinate of $R + P$ as pointed out in [10].

With these considerations, we can implement Algorithm 1 in a SNARK but conditional branching (If/Else) is not possible in SNARK circuits so this is replaced by constant window table lookups inside the circuit. This can be achieved using polynomials which vanish at the constants that aren't being selected, i.e. a 1-bit table lookup $R \leftarrow k_i \cdot R + (1 - k_i) \cdot (R + P)$. Hence this double-and-add algorithm requires N doubling, N additions and N 1-bit table lookup where N is the bit-size of the scalar. This can be extended to windowed double-and-add, i.e. scanning more than a bit per iteration using larger window tables, but the number of gates of the lookup table increases exponentially.

Since we start with $R \leftarrow 0_E$ it is infeasible to avoid conditional branching because affine formulas are incomplete. Instead, we scan the bits right-to-left and assume that the first bit k_0 is 1 (so that we start at $R \leftarrow P$), we double the input point P in the accumulator T in this algorithm and finally conditionally subtract (using the 1-bit lookup) P if k_0 was 0, as shown in Algorithm 3.

Algorithm 3: ScalarMultiplication(k, P)

Input. A scalar k of N bits,
A point P.
Output. $Q = [k]P$.
$R \leftarrow P$
$T \leftarrow [2]P$
for b bit of k read from second LSB to MSB **do**
  $R \leftarrow b \cdot R + (1 - b) \cdot (R + T)$
  $T \leftarrow [2]T$
end for
$k_0 \leftarrow k \bmod 2$
$R \leftarrow k_0 \cdot R + (1 - k_0) \cdot (R - P)$
return R

With these considerations, implementing the GLV scalar multiplication in a SNARK is also feasible as demonstrated in [9, Alg. 1]. We further extend this technique by using a 16-bit lookup, as show in the attached code (See Section 4.2). While the cost of lookups grows in SNARK circuits, we empirically found that, in the case of emulated arithmetic (elliptic curve base field is different from the SNARK field), it yields the best performances. The scalar decomposition is verified inside the SNARK circuit. However, sub-scalars can be negative. In this case the hint returns positive sub-scalars and a bit to indicate when to inverse points instead using a 1-bit lookup.

When the curve is endomorphism-equipped, we implement a MSM of size 4 using the 16-bit lookup as shown in the attached code (See Section 4.2).

For curves in twisted Edwards form, since formulae are complete, we implement a 2-bit windowed scalar multiplication as shown in [31] and [3, Alg. 17]. For a MSM of size 4, the cost of the 16-bit lookup is expensive in native arithmetic compared to the cost of point arithmetic. This makes the algorithm in Section 3.3 costlier than the one in Section 3.1 for Bandersnatch curve as shown in Table 6. In native SNARK circuits, we it is better to always use Section 3.1 regardless of the existence of an efficient endomorphism.

Proving and witness generation times trade-off. In blockchain applications, e.g. StarkNet [4], to prevent Denial of Service (DOS) attacks, arbitrary hints are not allowed. A malicious prover can run an infinite loop as a hint and cause a DOS. Only whitelisted hints by StarkNet are allowed. A developer submits their hint as a transaction calldata [5] which costs fees and is limited in size, inherently limiting the DOS surface. In this case, we would like to trade off the cost of the witness generation for the cost of the proving. In this case, for the scalar decomposition hint, we use the half-GCD algorithm in $\mathbb{Z}[j]$ developed in Section 2.5 instead of the lattice reduction techniques shown in Section 2.4. However, this increases the bounds on the sub-scalars as seen in Section 2.5 resulting in slower proving but cheaper calldata per transaction.

4.2 Implementation and benchmarks

Our code is available in the following GitHub repository:

`https://github.com/yelhousni/scalarmul-in-snark`.

We choose to implement our work for some of the widely used elliptic curves in SNARKs and blockchains (See Table 2).

We use the `gnark` library [6] in Golang to implement the circuits. We refer by r1cs (rank-1 constraint system) to Groth16 constraints and by scs (sparse constraint system) to the PLONK constraints. We consider two settings:

- Setting (i): The emulated case. We use a SNARK over an arbitrary field and we prove scalar multiplication over a given elliptic curve by emulating

[4] `https://www.starknet.io/`.

[5] `https://learnevm.com/chapters/fn/calldata`.

Curve[5]	Endomorphism?	Form
BN254 (Ethereum)	✓	Weierstrass
BLS12-381 [7]	✓	Weierstrass
BW6-761 [11]	✓	Weierstrass
Secp256k1 (Bitcoin)	✓	Weierstrass
P-256 (SEC 2)	✗	Weierstrass
P-384 (SEC 2)	✗	Weierstrass
Jubjub [31]	✗	twisted Edwards
Bandersnatch [24]	✓	twisted Edwards

Table 2: Some elliptic curves used in SNARKs and blockchains.

the base field arithmetic in the SNARK arbitrary field (see [22,1] for details on field emulation in SNARKs). Table 3 reports our results for curves with an efficient endomorphism (setting (i)-a) and Table 4 for curves without endomorphism (setting (i)-b).

– Setting (ii): The native case. We use a SNARK over a fixed field that matches the base field of the given elliptic curve we want to prove scalar multiplications over. Table 5 reports our results for curves with an efficient endomorphism (setting (ii)-a) and Table 6 for curves without endomorphism (setting (ii)-b).

Curve	Previous work ([9, Alg. 1])	This work (Sec. 3.3)	Speed-up
BN254	381467 scs	220436 scs	42%
	78246 r1cs	59351 r1cs	24%
BLS12-381	539973 scs	307045 scs	43%
	110928 r1cs	84508 r1cs	24%
BW6-761	1367067 scs	765544 scs	44%
	295194 r1cs	212659 r1cs	28%
Secp256k1	385461 scs	223188 scs	42%
	78940 r1cs	60089 r1cs	24%

Table 3: Implementation results for setting (i)-a.

Curve	Previous work (Alg. 3)	This work (Sec. 3.1)	Speed-up
P-256	612759 scs	294128 scs	52%
	157685 r1cs	78940 r1cs	50%
P-384	1233998 scs	588942 scs	52%
	325974 r1cs	159073 r1cs	51%

Table 4: Implementation results for setting (i)-b.

Curve	Previous work ([3, Alg. 17])	This work (Sec. 3.1)	Speed-up
Jubjub	5863 scs	4549 scs	22%
	3314 r1cs	2401 r1cs	28%

Table 5: Implementation results for setting (ii)-a.

Curve	Previous work ([24, Sec. 4.3])	This work (Sec. 3.3)	slow-down
Bandersnatch	4712 scs	8519 scs	**80%**
	2621 r1cs	4038 r1cs	**54%**

Table 6: Implementation results for setting (ii)-b.

In this section, we only implemented single scalar multiplication circuits. The practical and the theoretical speed-ups roughly match, but the number of constraints is slightly more complicated to analyse. The look-up table access has a cost in SNARKs that becomes significant when the table size increases. Moreover, for the emulated case, the cost of the non-native arithmetic operations become predominant. We did not investigate the implementation for multi scalar multiplications as SNARK circuit in `gnark`, but expect to get a speed-up only for $n = 2$. In systems such as PLONK or Groth16, reading points of the pre-computation table requires some constraints, and the overall circuit size may increase in consequence. A similar result is expected for scalar multiplication using GLS decomposition, where the scalar can be decomposed with 8 sub-scalars.

5 Conclusion

In this work, we investigate new techniques for proving scalar multiplications in the context of generic proof systems. While the computation of scalar multiplications has been optimized in the last decades, the case of a verification has not been investigated. Using the hint of the output point, it is possible to apply decomposition of the scalars in order to speed up the proving time. Table 7 provides the expected size of scalars and cost for simple and double scalar multiplications (SM and 2MSM resp.) in the case of small (GLV) and large (generic) discriminant curves. The obtained scalar sizes were obtained using a reduction of the lattice with respect to the Euclidean norm. Thus, the maximum norm is slightly larger than the Minkowski bound. A refined lattice reduction would be possible in order to reduce further the scalar sizes.

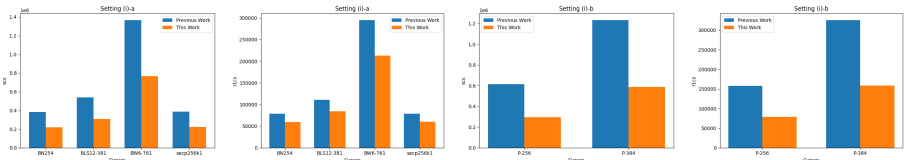
We implemented the different techniques for elliptic curves used in production in blockchains and obtained significant speed-ups, as shown in Figure 2.

As we have shown in Section 2.3, multi scalar multiplications based on Strauss's [30] trick speeds up the algorithmic cost but the pre-computation table cost grows exponentially in n. Hence, we expect to benefit from our optimizations for a double scalar multiplication ($n = 2$) but not for higher multi scalar multiplication ($n \geq 3$). However, for future work, considering in SNARK circuits different

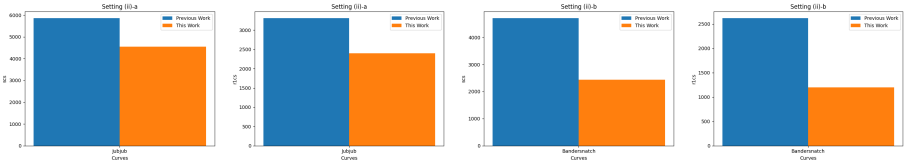
[5] The parameters of the elliptic curves can be found in `https://neuromancer.sk/std/`.

Algorithm	Previous work		This work	
	Cost	Scalar bound	Cost	Scalar bound
SM	MSM$(1, N)$	r	MSM$(2, \lceil N/2 \rceil)$	$1.16 \cdot r^{1/2}$
GLV SM	MSM$(2, \lceil N/2 \rceil)$	$1.16 \cdot r^{1/2}$	MSM$(4, \lceil N/4 \rceil)$	$1.25 \cdot r^{1/4}$
2MSM	MSM$(2, N)$	r	MSM$(3, \lceil 2N/3 \rceil)$	$1.22 \cdot r^{2/3}$
GLV 2MSM	MSM$(4, \lceil N/2 \rceil)$	$1.16 \cdot r^{1/2}$	MSM$(6, \lceil N/3 \rceil)$	$1.28 \cdot r^{1/3}$

Table 7: In-circuit scalar multiplication cost.



(a) Setting (i)-a scs (b) Setting (i)-a r1cs (c) Setting (i)-b scs (d) Setting (i)-b r1cs



(e) Setting (ii)-a scs (f) Setting (ii)-a r1cs (g) Setting (ii)-b scs (h) Setting (ii)-b r1cs

Fig. 2: Implementation results.

algorithms for larger sizes such as Pippenger's [4, Sec. 4] or Bos–Coster's [28, Sec. 4] algorithms may lead to promising results.

Acknowledgments. We express our gratitude to 0xPARC since this work partially started while the first and second authors (in alphabetical order) were at the 0xPARC residency in San Francisco. We would also like to thank Renaud Dubois and Olivier Bégassat for fruitful discussions.

References

1. Ambrona, M., Firsov, D., Querejeta-Azurmendi, I.: Efficient foreign-field arithmetic in PLONK. Cryptology ePrint Archive, Paper 2025/695 (2025), `https://eprint.iacr.org/2025/695`
2. Antipa, A., Brown, D.R.L., Gallant, R., Lambert, R., Struik, R., Vanstone, S.A.: Accelerated verification of ECDSA signatures. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 307–318. Springer, Berlin, Heidelberg (Aug 2006). `https://doi.org/10.1007/11693383_21`
3. Aranha, D.F., El Housni, Y., Guillevic, A.: A survey of elliptic curves for proof systems. DCC **91**(11), 3333–3378 (2023). `https://doi.org/10.1007/s10623-022-01135-y`

4. Bernstein, D.J., Doumen, J., Lange, T., Oosterwijk, J.J.: Faster batch forgery identification. In: Galbraith, S.D., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 454–473. Springer, Berlin, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34931-7_26
5. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Goldwasser, S. (ed.) ITCS 2012. pp. 326–349. ACM (Jan 2012). https://doi.org/10.1145/2090236.2090263
6. Botrel, G., Piellard, T., El Housni, Y., Kubjas, I.: Consensys/gnark: v0.12.0 (Jan 2025). https://doi.org/10.5281/zenodo.5819104, https://doi.org/10.5281/zenodo.5819104
7. Bowe, S.: BLS12-381: New zk-SNARK elliptic curve construction. Zcash blog (March 11 2017), https://blog.z.cash/new-snark-curve/
8. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: ZEXE: Enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy. pp. 947–964. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00050
9. Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021 (2019), https://eprint.iacr.org/2019/1021
10. Eisenträger, K., Lauter, K., Montgomery, P.L.: Fast elliptic curve arithmetic and improved Weil pairing evaluation. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 343–354. Springer, Berlin, Heidelberg (Apr 2003). https://doi.org/10.1007/3-540-36563-X_24
11. El Housni, Y., Guillevic, A.: Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 20. LNCS, vol. 12579, pp. 259–279. Springer, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-65411-5_13
12. ElGamal, T.: On computing logarithms over finite fields. In: Williams, H.C. (ed.) CRYPTO'85. LNCS, vol. 218, pp. 396–402. Springer, Berlin, Heidelberg (Aug 1986). https://doi.org/10.1007/3-540-39799-X_28
13. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019), https://eprint.iacr.org/2019/953
14. Galbraith, S.D., Lin, X., Scott, M.: Endomorphisms for faster elliptic curve cryptography on a large class of curves. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 518–535. Springer, Berlin, Heidelberg (Apr 2009). https://doi.org/10.1007/978-3-642-01001-9_30
15. Galbraith, S.D., Lin, X., Scott, M.: Endomorphisms for faster elliptic curve cryptography on a large class of curves. Journal of Cryptology **24**(3), 446–469 (Jul 2011). https://doi.org/10.1007/s00145-010-9065-y
16. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer, Berlin, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_11
17. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC. pp. 99–108. ACM Press (Jun 2011). https://doi.org/10.1145/1993636.1993651
18. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. **18**(1), 186–208 (1989). https://doi.org/10.1137/0218012

19. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Berlin, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_11
20. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC. pp. 723–732. ACM Press (May 1992). https://doi.org/10.1145/129712.129782
21. Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., shelat, a., Shi, E.: C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093 (2015), https://eprint.iacr.org/2015/1093
22. Kubjas, I.: Notes about optimizing emulated pairing (part 1). https://hackmd.io/@ivokub/SyJRV7ye2 (2023)
23. Lenstra, H.j., Lenstra, A., LovÃ¡sz, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**, 515–534 (1982), http://eudml.org/doc/182903
24. Masson, S., Sanso, A., Zhang, Z.: Bandersnatch: a fast elliptic curve built over the BLS12-381 scalar field. DCC **92**(12), 4131–4143 (2024). https://doi.org/10.1007/s10623-024-01472-0
25. Micali, S.: CS proofs (extended abstracts). In: 35th FOCS. pp. 436–453. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365746
26. Nguyen, P.Q., Stehlé, D.: Low-dimensional lattice basis reduction revisited. ACM Trans. Algorithms **5**(4) (Nov 2009). https://doi.org/10.1145/1597036.1597050, https://doi.org/10.1145/1597036.1597050
27. Prover, L.: Linea prover documentation. Cryptology ePrint Archive, Paper 2022/1633 (2022), https://eprint.iacr.org/2022/1633
28. de Rooij, P.: Efficient exponentiation using procomputation and vector addition chains. pp. 389–399 (1995). https://doi.org/10.1007/BFb0053453
29. Silverman, J.H.: The arithmetic of elliptic curves, Graduate texts in mathematics, vol. 106. Springer (1986)
30. Strauss, E.G.: Addition chains of vectors (problem 5125). American Mathematical Monthly **70**(114), 806–808 (1964)
31. ZCash: What is jubjub? https://z.cash/technology/jubjub/ (2021)

A Comparison of norms

A.1 Comparison of $\mathcal{N}$ and the maximum norm in $\mathbb{Z}[j]$

We compare both norms with $|\cdot|_2$ defined by $|x+jy|_2 = \sqrt{x^2+y^2}$. Recall that $\mathcal{N}(a+jb) = \sqrt{a^2+b^2-ab}$, so it is the norm in the complex sense. From the triangle inequality, followed by Cauchy inequality, we obtain that

$$|a+jb|_\infty \leq |a|_\infty|1|_\infty + |b|_\infty|1|\infty \leq \sqrt{2}|a+jb|_2,$$
$$\mathcal{N}(a+jb) \leq \mathcal{N}(1)\mathcal{N}(a) + \mathcal{N}(1)\mathcal{N}(b) \leq \sqrt{2}|a+jb|_2.$$

Now, on the circle $|x|_2 = 1$, the norm $\mathcal{N}$ reaches its minimum $\frac{1}{\sqrt{2}}$ at $\frac{1}{\sqrt{2}} + \frac{j}{\sqrt{2}}$, and so

$$\frac{1}{\sqrt{2}}|\cdot|_2 \leq \mathcal{N}(\cdot).$$

The same reasoning shows that

$$\frac{1}{\sqrt{2}}|\cdot|_2 \leq |\cdot|_\infty.$$

A.2 Comparison of $\mathcal{N}$ and the maximum norm in $\mathbb{Z}[\sqrt{-2}]$

Here again, $|a+b\sqrt{-2}|_\infty = \max\{|a|,|b|\}$ and $\mathcal{N}(a+\sqrt{-2}b) = \sqrt{a^2+2b^2}$. Again, the triangle inequality and the Cauchy inequality shows that

$$\begin{aligned}\mathcal{N}(x) &\leq \sqrt{2}|x|_2,\\ |x|_\infty &\leq \sqrt{2}|x|_2.\end{aligned}$$

Now, on the circle $|x|_2 = 1$, we see that the norm $\mathcal{N}$ reaches its minimum 1 at $1+0\sqrt{-2}$, and $|\cdot|_\infty$ reaches its minimum $\frac{1}{\sqrt{2}}$ at $\frac{1}{\sqrt{2}}+\sqrt{-2}\frac{1}{\sqrt{-2}}$, and so we obtain

$$\begin{aligned}|\cdot|_2 &\leq |\cdot|_\infty,\\ \frac{1}{\sqrt{2}}|\cdot|_2 &\leq \mathcal{N}(\cdot).\end{aligned}$$