Elliptic Curves over Finite Fields

Lehigh University

Tal Derei

September 14, 2022

Abstract

We present a primer into field arithmetic, quadratic and cubic extension arithmetic, and elliptic curve operations for the MNT4-753 and MNT6-753 pairing-friendly elliptic curves. These workloads can be implemented on parallel architectures like GPUs using a library called **Cuda-Fixnum**, a fixed-precision SIMD library that targets CUDA [1]. The library exposes an interface for performing modular arithmetic natively over vectors of n-bit integers, ranging from 32 - 2048 bits, on GPUs. The specifics of this library are beyond the scope of this paper.

1 Preliminaries: Pairing-friendly Elliptic Curves

Elliptic curves over finite fields serve as the basic building blocks for instantiating recursive zero-knowledge proof systems, i.e. verifying a zk-SNARK inside another zk-SNARK [2]. The first implementation of recursive proofs was demonstrated using a family of pairing-friendly curves called MNT curves [3] in "Scalable Zero Knowledge via Cycles of Elliptic Curves" by Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza [4]. Pairing-friendly curves are important in blockchains like Ethereum for cheaply performing cryptographic operations. Ethereum specifically implements a family of pairing-friendly elliptic curves called the Barreto-Naehrig (BN-256) curves. The Ethereum Virtual Machine (EVM) has optimized precompiled smart contracts for the BN-256 elliptic curve to perform the gas-efficient proof verification on-chain.

1.1 Finite Fields and Elliptic Curves

A finite field is a field that contains a finite number of elements and has a prime order. Let E $<F_q>$ denote an elliptic curve E as an algebraic curve of the form $y^2 = x^3 + ax + b \ (mod\ q)$ over a prime finite field F_q. Let $E(F_q)$ denote the group of points of E over F_q with cardinality p = $\#E(F_q)$. For this curve, we define F_q as the base field, and F_p as the scalar field.

Definition 1. An m-cycle of elliptic curves is a list of m distinct elliptic curves $E_1 <F_q>$, ... , $E_m <F_m>$ where F_1 , ... , F_m are prime, and the number of points on the curve satisfies the following relation [3]:

$$\#E_1(F_1) = q_2 \text{ , ... , } \#E_i(F_{q_1}) = q_i + 1 \text{ , ... , } \#E_m(F_{q_m}) = q_1$$

A cycle of elliptic curves is a list of elliptic curves over finite fields where the number of points on one curve E is the size of the field F of the next elliptic curve, and this cycle repeats. More concretely, the minimum cycle of curves is a pair of two pairing-friendly elliptic curves E_x and E_y such that: $E_x \lt F_p \gt$, where the curve E_x with prime order x is defined over finite field F_p, and $E_y \lt F_q \gt$, where the curve E_y with prime order y is defined over finite field F_q (where q and E_x have the same order).

With the definition above, we can construct pairing-friendly elliptic curves that yield efficient zk-SNARK constructions, and their cycles enable recursive composition of proofs and arbitrary statements [4]. For instance, consider the construction with the following properties:

1. A pairing-friendly elliptic curve with prime order p yields a SNARK construction that

can **prove** arbitrary computations in F_p (i.e. arithmetic over the scalar field F_p).

2. A SNARKs verification algorithm **verifies** these arbitrary computations over the base field F_q, and thus efficiently expressed as an F_q arithmetic circuit.

To summarize, a proof system (prover and verifier) is instantiated with a *single* pairing-friendly elliptic curve over the base field F_q. The core components (the arithmetic circuits and the prover/verifier) are performing computations over different fields.

- Circuits are performing arithmetic over the scalar field F_p (i.e. F_p is equivalent to the group order of the elliptic curve), thereby proving statements about computations over F_p.
- Prover and verifier generate proofs and perform efficient verification using the base field F_q.

To enable proof recursion involving multiple proofs, construct another elliptic curve with: **(1)** base field F_p enabling proof generation that can be efficient verified in the first curves F_p arithmetic circuit, and **(2)** scalar field F_q which has an F_q arithmetic circuit that can efficiently verify proofs from the first curve [8]. We're ultimately generating a proof on the first curve that can be efficiently verified on the second curve. The second curve can then generate another proof on top of that poof that can be verified by the first curve. And this cycle repeats.

Ethereum's ECDSA signature scheme is defined over the secp256k1 elliptic curve, and serves as a practical example to motivate this construction. Proving systems like Groth16 and PlonK can't be instantiated on top of secp256k1 since it's not pairing-friendly. Yet there are SNARK constructions using these recursive methods that enable rollup providers to encode and batch Ethereum signatures in a SNARK to be verified on chain. The final proof compresses thousands of other proofs, each containing thousands of signatures, in a recursive manner.

1.2 MNT Curves and Pairings

The most efficient SNARKs use pairings in their construction, and require the use of pairing-friendly elliptic curves. Therefore we use two elliptic curves E_r and E_q such that $|E_r|$ = r is defined over F_q and $|E_q|$ = q is defined over F_r. These curves are called "2-cycles" and support pairings, as illustrated below. [7].

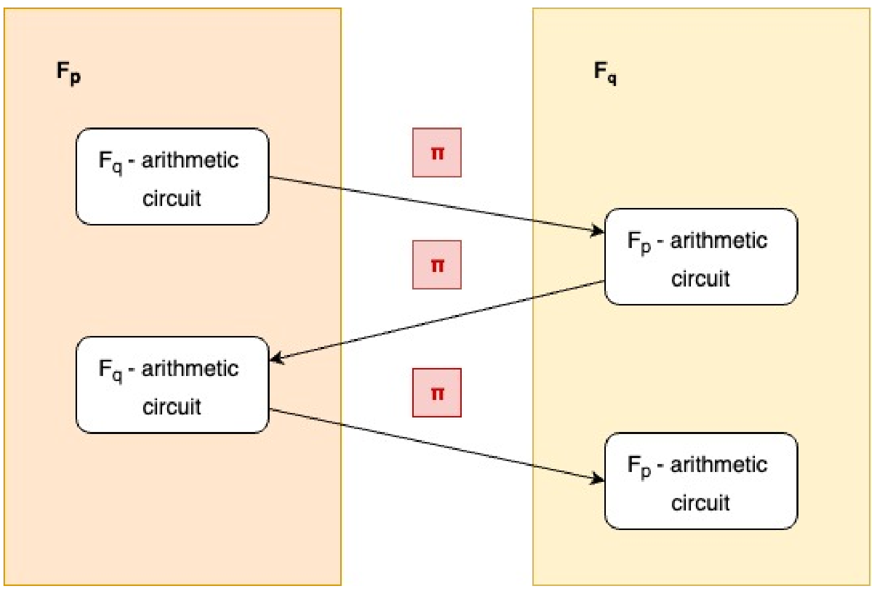

Figure 1: *Proof Recursion via Cycles of Two Elliptic Curves*

Definition 2. An elliptic curve E_x is "pairing-friendly" if the order x divides $q^k - 1$ for some $k \leq 50$. The embedding degree of E_x is defined to be the smallest such k value [5].

A pairing-friendly elliptic curve E has a billinear map e: G_1 x G_2 $\rightarrow$ G_T, where G_1 and G_2 are distinct prime-order r subgroups of E, and $G_T \subset F_q^k$ of the same order r [6]. Pairings are functions that map points on two distinct elliptic curves into a finite field. In production blockchain systems like Mina, a fully-succinct blockchain protocol based on zero-knowledge proofs for validating the entire chain state, MNT4 and MNT6 curves are used. These are known as "two-chains of elliptic curves" (i.e. 2-cycle of pairing-friendly elliptic curves) with embedding degrees 4 and 6, respectively. These embedding degrees determine the size of the finite fields (order of 768 bits) to achieve 128-bit security. Large base fields are required to achieve sufficient security, but result in slower arithmetic operations.

2 Mathematical Operations

We explore implementing field arithmetic, quadratic and cubic extension arithmetic, and elliptic curve operations for the MNT4-753 and MNT6-753 pairing-friendly elliptic curves.

2.1 Field Arithmetic

Traditional programming paradigms work with native 32-bit/64-bit integers. SNARK provers require integers that are much larger, on the order of 753-bits for MNT curves. We represent these integers in a special form called "Montgomery" representation for performing efficient multiplication. The Montgomery representation of the element x (e.g. 99) is $(xR)\ mod\ q$, where R = 2^{768}. This 753-bit integer is can be represented as an array of 12 64-bit integers (since 12 * 64 = 768 > 753), where each element of the array is called a "limb". For example, the Montgomery representation of the multiplication of two elements (A * B) mod q is:

$$\text{Let } A = (x * R)\ \%\ q$$

$$\text{Let } B = (y * R)\ \ \%\ q$$

$$A * B = ((x * R)\ mod\ q) * ((y * R)\ mod\ q) = (x * y * R^2)\ mod\ q$$

2.2 Quadratic and Cubic Extension Arithmetic

A field extension of a field F is another field F' which contains F. For example, the real numbers R is a field extension of rational numbers Q. Instead of multiplying field elements, we'll be multiplying elements in a "quadratic extension field". We start by picking number a in F_q which does not have a square root in F_q, e.g. 13. Then define the field called F_q[x]/(x^2 = 13). This is the field obtained by adding an "imaginary" square root x for 13 to F_q.

Definition 3. Let an element of F_{q^2} be a pair $(a_0, a_1$ where each of a_0 and a_1 are elements of the F_q. Addition and multiplication for F_q^2 is defined as follows [5]:

$$Add : (a_0 + a_1x) + (b_0 + b_1x) = (a_0 + b_0) + (a_1 + b_1)x \tag{1}$$

$$\begin{aligned} Mult : (a_0 + a_1x)(b_0 + b_1x) &= a_0b_0 + a_0b1x + b_0a1x + a_1b_1x^2 && (2) \\ &= a_0b_0 + a_0b1x + b_0a1x + 13a_1b_1 && (3) \\ &= (a_0b_0 + 13a_1b_1) + (a_0b_1 + b_0a_1)x && (4) \end{aligned}$$

The **pseudocode** for addition and multiplication in a quadratic extension field is as follows:

```
var alpha = fq(13);
var fq2_add = (a, b) => {
  return { a: fq_add(a.a0, b.a0), b: fq_add(a.a1, b.a1) };
};
var fq2_mul = (a, b) => {
  var a0_b0 = fq_mul(a.a0, b.a0); var a1_b1 = fq_mul(a.a1, b.a1);
  var a1_b0 = fq_mul(a.a1, b.a0); var a0_b1 = fq_mul(a.a0, b.a1);
  return {a0: fq_add(a0_b0, fq_mul(a1_b1, alpha)),a1: fq_add(a1_b0, a0_b1)};
}
```

Definition 4. The elements of the cubic extension field F_q^3 are of the form: $a_0 + a_1x + a_2x^2$. This is an extension of field F_q since it has x^3 elements, where each element is a tuple (a_0, a_1, a_2) from the field F_q [5].

$$Add : (a_0 + a_1x + a_2x^2) + (b_0 + b_1x + b_2x^2) \tag{5}$$

$$= (a_0 + b_0) + (a_1 + b_1)x + (a_1 + b_1)x^2 \tag{6}$$

$$Mult : (a_0 + a_1x + a_2x^2)(b_0 + b_1x + b_2x^2) \tag{7}$$

$$= a_0b_0 + a_0b1x + a_0b_2x^2 + a1b_0x + a_1b_1x^2 + a_1b_2x^3 + a_2b_0x^2 + a_2b_1x^3 + a_2b_2x^4 \tag{8}$$

$$= a_0b_0 + a_0b1x + a_0b_2x^2 + a1b_0x + a_1b_1x^2 + 11a_1b_2 + a_2b_0x^2 + 11a_2b_1 + 11a_2b_2x \tag{9}$$

$$= (a_0b_0 + 11a_1b_2 + 11a_2b_1) + (a_0b1 + a1b_0 + 11a_2b_2)x + (a_0b_2 + a_1b_1 + a_2b_0)x^2 \tag{10}$$

The **pseudocode** for addition and multiplication in a cubic extension field is as follows:

```
var alpha = fq(11);
var fq3_mul = (a, b) => {
  var a0_b0 = fq_mul(a.a0, b.a0);
  var a0_b1 = fq_mul(a.a0, b.a1);
  var a0_b2 = fq_mul(a.a0, b.a2);
  var a1_b0 = fq_mul(a.a1, b.a0);
  var a1_b1 = fq_mul(a.a1, b.a1);
  var a1_b2 = fq_mul(a.a1, b.a2);
  var a2_b0 = fq_mul(a.a2, b.a0);
  var a2_b1 = fq_mul(a.a2, b.a1);
  var a2_b2 = fq_mul(a.a2, b.a2);
```

```
  return {
    a0: fq_add(a0_b0, fq_mul(alpha, fq_add(a1_b2, a2_b1))),
    a1: fq_add(a0_b1, fq_add(a1_b0, fq_mul(alpha, a2_b2))),
    a2: fq_add(a0_b2, fq_add(a1_b1, a2_b0))
  };
};

var fq3_add = (a, b) => {
  return {
    a0: fq_add(a.a0, b.a0),
    a1: fq_add(a.a1, b.a1),
    a2: fq_add(a.a2, b.a2)
  };
};
```

2.3 Curve Operations

We're now able to perform group operations for elliptic curves. A single SNARK proving / verifying system is a pair of elliptic curves (G1, G2). Since we're optimizing it for both MNT4 and MNT6, we have 4 curves [5]:

1. MNT4 G1
2. MNT4 G2
3. MNT6 G1
4. MNT6 G2

Each curve is specified by a pair of two of these elements from the finite fields, specifically:

MNT4 G1: (Fq, Fq)

MNT4 G2: (Fq2, Fq2)

MNT6 G1: (Fq, Fq)

MNT6 G2: (Fq3, Fq3)

where G1 and G2 are cyclic groups of prime order q, with generator p.

3 Conclusion

Elliptic curves over finite fields are crucial building blocks in the context of building zero-knowledge proving systems. The family of MNT4 and MNT6 curves have special properties of being pairing-friendly cycles of elliptic curves that enable recursive proof composition.

4 References

[1] https://github.com/unzvfu/cuda-fixnum

[2] https://eprint.iacr.org/2022/586.pdf

[3] https://arxiv.org/pdf/1803.02067.pdf

[4] https://eprint.iacr.org/2014/595.pdf

[5] https://coinlist.co/build/coda/pages/theory

[6] https://eprint.iacr.org/2021/1359.pdf

[7] https://www.michaelstraka.com/posts/recursivesnarks/

[8] https://zcash.github.io/halo2/background/curves.html