

Verifying elliptic curves in Welder

Rodrigo Raya

École Polytechnique Fédérale de Lausanne

June 21, 2017

- 1 Introduction
- 2 Welder fundamentals
- 3 Elliptic curves notions
- 4 The naive approach
- 5 The tactic
- 6 Towards the proof of associativity
- 7 Future work

- 1 Introduction
- 2 Welder fundamentals
- 3 Elliptic curves notions
- 4 The naive approach
- 5 The tactic
- 6 Towards the proof of associativity
- 7 Future work

Why verifying elliptic curves?

- Shorter key-lengths.
- Savings in bandwidth.
- Better time efficiency.
- Good support to do cryptography online.

Threats to elliptic curves

- Side-channel attacks.
- Use of Edwards form to prevent them.
- NIST curves are suspected to provide back-doors.
- *The math is good, but math has no agency. Code has agency, and the code has been subverted.
(Bruce Schneier)*
- Alternatives to NIST curves: Curve25519 used in OpenSSH.

- HACL* library: a verified library that includes all the primitives from a reduced cryptographic API: NaCl.
- The first approach to verify the full byte-level scalar multiplication operation.
- Further work is needed to verify pre-computation steps of the Ed25519 signature protocol.

- Verified Software Toolchain.
- So far, they verified SHA-256 or HMAC.

*The Verified Software Toolchain project assures with machine-checked proofs that the assertions claimed at the top of the toolchain really hold in the machine-language program, running in the operating-system context.
(Project homepage)*

Our tools

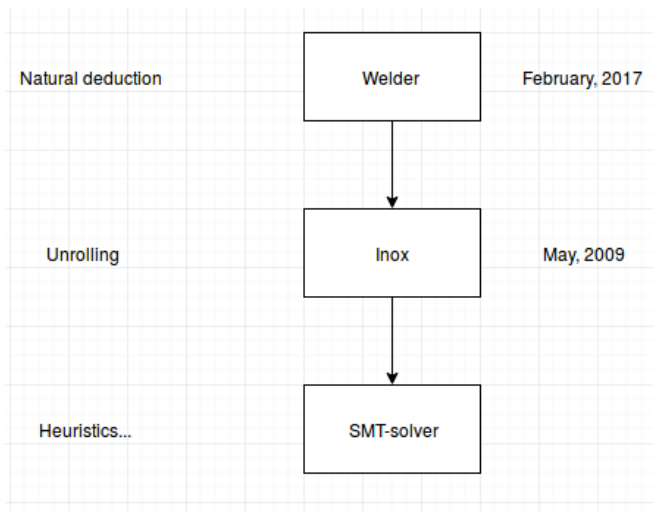


Figure : Welder-Inox-solver stack

Advantages and disadvantages of our tools

- Scala has a direct translation to JavaScript while F* currently doesn't.
- Although Inox is consolidated, Welder still needs some repairs, as we will see...

- 1 Introduction
- 2 **Welder fundamentals**
- 3 Elliptic curves notions
- 4 The naive approach
- 5 The tactic
- 6 Towards the proof of associativity
- 7 Future work

Natural deduction rules

Name	Logic rule	Welder construct
Conjunction introduction	$\frac{A \quad B}{A \wedge B}$	andI
Conjunction elimination	$\frac{A \wedge B}{A}$	andE
Disjunction introduction	$\frac{A}{A \vee B}$	orI
Disjunction introduction	$\frac{A}{B \vee A}$	orI
Disjunction elimination	$\frac{A \vee B \quad \begin{array}{c} [A] \\ C \end{array} \quad \begin{array}{c} [B] \\ C \end{array}}{C}$	orE
Implication introduction	$\frac{\begin{array}{c} [A] \\ B \end{array}}{A \Rightarrow B}$	implI
Implication elimination	$\frac{A \Rightarrow B \quad A}{B}$	implE
Negation introduction	$\frac{\begin{array}{c} [B] \\ \perp \end{array}}{\neg B}$	notI
Negation elimination	$\frac{\neg(\neg B)}{B}$	notE

Table : Basic inference rules in Welder

Natural deduction rules

Name	Logic rule	Condition	Welder construct
\forall introduction	$\frac{A}{\forall x.A}$	$x \notin \text{free}(\text{assumptions}(A))$	forallI
\forall elimination	$\frac{\forall x.A}{A[t/x]}$	$A[t/x]$ is the substitution of x by term t	forallE
\exists introduction	$\frac{A[t/x]}{\exists x.A}$	$A[t/x]$ is the substitution of x by term t	existsI
\exists elimination	$\frac{\exists x.A \wedge \delta, A \vdash B}{\delta \vdash B}$	$x \notin \text{free}(\delta) \cup \text{free}(B)$	existsE

Table : Quantified rules in Welder

Natural deduction rules

- Two powerful mechanisms: structural and natural induction.
- One challenge: automate them at the Welder level.
- Natural deduction and goal seeking concepts mixed.
- We developed our own working notes.

Welder-Inox Documentation

Release 1

Rodrigo Raya

Figure : Welder-Inox documentation



Welder is not sound!

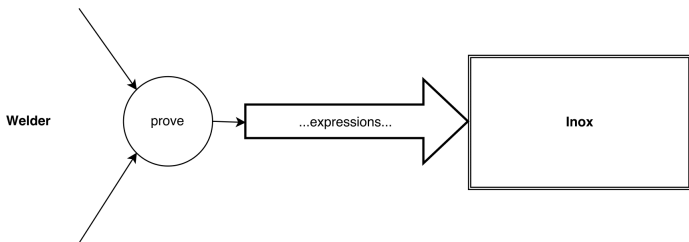


Figure : The source of unsoundness in Welder

Example of unsoundness

```
val lemma = prove(  
  Assume(BooleanLiteral(false), BooleanLiteral(false))  
)  
val theorem = prove(  
  E(BigInt(1)) == E(BigInt(2)), lemma  
)  
// Prints Success(Theorem(1 == 2))
```

Useless constructs

```
val naturalADT = new ADTConstructor(  
  natural, Seq(), None,  
  Seq(ValDef(natElement, IntegerType)),  
  Set(HasADTInvariant(isNatID))  
)  
val expr:Expr = T(natural)()(E(BigInt(-1)))
```


- 1 Introduction
- 2 Welder fundamentals
- 3 Elliptic curves notions**
- 4 The naive approach
- 5 The tactic
- 6 Towards the proof of associativity
- 7 Future work

Definition (Elliptic curve)

Given a field K one can define an elliptic curve $E(K)$ as the set

$$\{(x, y) : x, y \in K \wedge y^2 = x^3 + Ax + B\} \cup \{\infty\}$$

where the discriminant of the equation, $4A^3 + 27B^2$, is different from zero.

- $4A^3 + 27B^2 \neq 0$ curve non-singular. Discrete logarithm-hard.
- Characteristic $q \neq 2, 3$. Discrete-logarithm "easy" (recent research).

Group law graphic idea

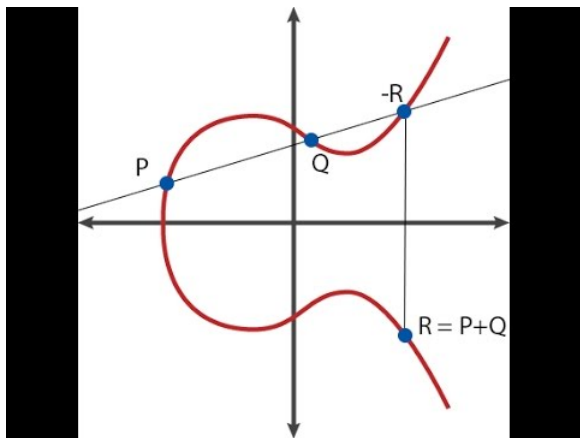


Figure : Graphical representation of the addition on an elliptic curve

Definition (Secant case)

Take $P_1, P_2 \in E(K) \setminus \{\infty\}$ of the form $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ then:

- If $x_1 \neq x_2$

$$P_1 + P_2 = (x_3, y_3) = \begin{cases} m = \frac{y_2 - y_1}{x_2 - x_1} \\ x_3 = m^2 - x_1 - x_2 \\ y_3 = m(x_1 - x_3) - y_1 \end{cases}$$

- If $x_1 = x_2 \wedge y_1 \neq y_2$ then $P_1 + P_2 = \infty$

Definition (Tangent case)

Take $P_1, P_2 \in E(K) \setminus \{\infty\}$ of the form $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ then:

- If $P_1 = P_2 \wedge y_1 \neq 0$

$$P_1 + P_2 = (x_3, y_3) = \begin{cases} m = \frac{3x_1^2 + A}{2y_1} \\ x_3 = m^2 - 2x_1 \\ y_3 = m(x_1 - x_3) - y_1 \end{cases}$$

- If $P_1 = P_2 \wedge y_1 = 0$ then $P_1 + P_2 = \infty$
- $\forall P \in E(K), P + \infty = P$

Theorem (*Group properties of the addition over an elliptic curve*)

The addition of points on an elliptic curve $E(K)$ satisfies:

- *Commutativity:* $\forall P_1, P_2 \in E(K). P_1 + P_2 = P_2 + P_1$
- *Existence of neutral element:* $\forall P \in E(K) P + \infty = P$
- *Existence of opposite elements:*
 $\forall P \in E(K) \exists P' \in E(K). P + P' = \infty$
- *Associativity:*
 $\forall P_1, P_2, P_3 \in E(K). (P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$

Therefore, $(E(K), +)$ is an abelian group with neutral element ∞ .

- *Of course there a lot of cases to consider...But in a few days you will be able to check associativity using these formulas. So we need say nothing more about the proof of the associative law!*
(Tate and Silverman)
- It has been noted that expanding the resulting polynomial equation into monomials would involve some 10^{25} terms...

- 1 Introduction
- 2 Welder fundamentals
- 3 Elliptic curves notions
- 4 The naive approach**
- 5 The tactic
- 6 Towards the proof of associativity
- 7 Future work

- We define an abstract field instead of doing integer arithmetic.
- Class hierarchies limited to one abstract parent and a sequence of constructors.
- No sub-typing between constructors.

Abstract field hierarchy

```
abstract class Element()  
final case class Zero() extends Element  
final case class One() extends Element  
final case class notZeroOne() extends Element  
def nonZero(e: Expr): Expr =  
  e.isInstanceOf(One) || e.isInstanceOf(notZeroOne)
```

Abstract and infix operations

```
val add: Expr =  
  Variable(FreshIdentifier("add"), (F, F) =>: F)  
  
implicit class Infix(private val lhs: Expr) extends AnyVal  
{  
  def ^+(rhs: Expr): Expr = addFunction(lhs, rhs)  
  ...  
}
```

A first algebra exercise

- We introduce the field axioms and deduced theorems from them.
- We get the precise hypothesis for each lemma.
- For instance, when do we use that characteristic $\neq 2, 3$
- Perhaps a unification procedure could automate the process.

A first algebra exercise: the axioms

Name	Axiom
addAssociative	$\forall x, y, z \in F. (x + y) + z = x + (y + z)$
addNeutralElement	$\forall x \in F. x + 0 = 0 + x = x$
addOppositeElement	$\forall x \in F. x + \text{opp}(x) = \text{opp}(x) + x = 0$
addCommutative	$\forall x, y \in F. x + y = y + x$
multAssociative	$\forall x, y, z \in F. (xy)z = x(yz)$
multNeutralElement	$\forall x \in F. x1 = 1x = x$
multInverseElement	$\forall x \in F. x \text{inv}(x) = \text{inv}(x)x = 1$
multCommutative	$\forall x, y \in F. xy = yx$
isDistributive	$\forall x, y, z \in F. x(y + z) = xy + xz \wedge (x + y)z = xz + yz$
notCharacteristic2	$1 + 1 \neq 0$
notCharacteristic3	$1 + 1 + 1 \neq 0$

Table : List of the axioms used in the field theory

A first algebra exercise: the theorems

Name	Theorem
uniqueAddNeutral	$\forall y \in F. (\forall x \in F. x + y = y + x = y) \implies y = 0$
uniqueMultInverse	$\forall x, y, z \in F. (x \neq 0 \wedge (xy = yx = 1) \wedge (xz = zx = 1)) \implies y = z$
uniqueAddOpposite	$\forall x, y, z \in F. ((x + y = y + x = 0) \wedge (x + z = z + x = 0)) \implies y = z$
oppositeOfZero	$opp(0) = 0$
oppositeInvolution	$\forall x \in F. opp(opp(x)) = x$
simplification	$\forall x \in F. x = 2x \implies x = 0$
zeroDivisor	$\forall x \in F. x0 = 0 = 0x$
oppositeOfAdd	$\forall x, y \in F. -(x + y) = (-y) + (-x)$
oppositeOfMult	$\forall x, y, z \in F. -(xy) = (-x)y = x(-y)$
integralDomain	$\forall x, y \in F. x, y \neq 0 \implies xy \neq 0$
cancellation	$\forall x, y, z \in F. x + z = y + z \implies x = y$
inverseOfMult	$\forall x, y \in F. x, y \neq 0 \implies (xy)^{-1} = y^{-1}x^{-1}$
zeroNotMinusOne	$opp(1) \neq 0$
inverseOfMinusOne	$inv(-1) = -1$
oppositeofInverse	$\forall x \in F. -(x)^{-1} = (-x)^{-1}$
exp1	$\forall h \in F, n \in \mathbb{N}. h^{n+1} = hh^n$
exp2	$\forall h \in F, i, j \in \mathbb{N}. h^{i+j} = h^i h^j$
doubleXZero	$\forall x \in F. 2x = 0 \implies x = 0$

Table : List of the theorems deduced in the field theory

Elliptic curve theorems proved directly

- Commutativity, neutral element and opposite element are easy to proof.
- One case of commutativity needs rewriting.

Commutativity: rewriting needed

```
//Proof that lambda,x,y don't change when interchanging  
//x1,x2 and y1,y2  
val lambda = (y2 - y1) / (x2 - x1)  
val x = lambda^2 - x1 - x2  
val y = lambda * (x1 - x) - y1  
finite(x,y)
```


Harder theorems: associativity

- Associativity is much harder.
- We solve it in three steps:
 - A normalizer procedure.
 - A rewriter procedure.
 - Rationalizing the equations involved.

- 1 Introduction
- 2 Welder fundamentals
- 3 Elliptic curves notions
- 4 The naive approach
- 5 The tactic**
- 6 Towards the proof of associativity
- 7 Future work

Let us fix:

- The underlying ring R .
- A set of variables $V = (v_i)_{i \in \{0, \dots, k-1\}}$.
- A set of values $N = (n_i)_{i \in \{0, \dots, k-1\}}$.
- A function $A : V \rightarrow N$ such that $A(v_i) = n_i$.

And let formalize the equations we have into polynomial equalities.

A polynomial term over V is one of the following:

- an element of R
- an element of V
- $opp(x)$ where x is a polynomial over V
- $x + y$, $x * y$ where x, y are polynomials over V
- x^n where x is a polynomial over V and $n \in \mathbb{N}$

The set of polynomial terms over V is denoted $\tau(V)$.

Evaluation of polynomial terms

The evaluation function *evalp* of a polynomial term $q \in \tau(V)$ is defined as:

$$\text{evalp}(q, A) = \begin{cases} q & \text{if } q \in R \\ n & \text{if } q \in V \wedge A(q) = n \\ \text{opp}(\text{evalp}(r, A)) & \text{if } q = \text{opp}(r) \\ \text{evalp}(r, A) + \text{evalp}(s, A) & \text{if } q = r + s \\ \text{evalp}(r, A) \cdot \text{evalp}(s, A) & \text{if } q = r * s \\ \text{evalp}(r, A)^n & \text{if } q = x^n \end{cases}$$

We define a normal form for the derived polynomials. This form ensures some properties:

- $\forall f \in \tau(V). evalp(f, A) = evalh(norm(f, V), N)$
- $\forall f, g \in \tau(V). norm(f) = norm(g) \implies evalp(f, A) = evalp(g, A)$

Motivation: Horner polynomial form

- $p(x) = a_0 + a_1 \cdot x + \cdots + a_n \cdot x^n$
- $p(x) = a_0 + x \cdot (a_1 + x \cdot (\cdots + x(a_{n-1} + a_n \cdot x) \cdots))$
- $b_n = a_n$
 $b_{n-1} = a_{n-1} + b_n \cdot x_0$
 \dots
 $b_0 = a_0 + b_1 \cdot x_0$
- We don't seek efficiency but a systematic way to reason about polynomials.

A compact representation

For the multivariate case, the non-constant polynomial is split in two cases called *POW* and *POP*.

- $X^4 + 1$
- $PX(PX(PX(PX(P_c 1)0)0)0)1)$
- We can do better if we add a power index.
- $PX^4(P_c 1) 1$

This will be summarized as $POW(i, p, q)$ and represents polynomial $px^i + q$.

For the multivariate case, the non-constant polynomial is split in two cases called *POW* and *POP*.

- *POP* allows to construct an onion-like polynomial by skipping not desired variables.
- Given list of variables V , $POP(i, p)$ will drop the first i variables and interpret (when evaluating) polynomial p from variable v_i .

A sparse Horner form (SHF) is one of the following:

- An element of R .
- Given $i \in \mathbb{N}$ and p a SHF then $POP(i, p)$ is a SHF.
- Given $i \in \mathbb{N}$ and p, q SHF then $POW(i, p, q)$ is a SHF.

Sparse Horner normal form

A sparse Horner normal form (SHNF) is one of the following SHF:

- An element of R .
- Given $i \in \mathbb{N}^*$ and p a SHNF of the form $POW(j, q, r)$ then $POP(i, p)$ is a SHNF.
- Given $i \in \mathbb{N}^*$ and p, q SHNF where p is not zero and not of the form $POW(j, r, 0)$ then $POW(i, p, q)$ is a SHNF.
- We will denote by \mathcal{H} the set of all SHNFs.
- Conditions can be translated in everyday language.

Operations on normal forms and its properties

We define operations \oplus, \ominus, \otimes and power on \mathcal{H} and we proof that:

- \mathcal{H} is closed under the operations.
- *evalh* preserves normality.

Lemma

Let $x, y \in \mathcal{H}$. Then:

1. $x \oplus y \in \mathcal{H}$
2. $\text{evalh}(x \oplus y, N) = \text{evalh}(x, N) + \text{evalh}(y, N)$.
3. $\ominus x \in \mathcal{H}$
4. $\text{evalh}(\ominus x, N) = \text{opp}(\text{evalh}(x, N))$
5. $x \otimes y \in \mathcal{H}$
6. $\text{evalh}(x \otimes y, N) = \text{evalh}(x) \cdot \text{evalh}(y, N)$
7. $x^k \in \mathcal{H}$
8. $\text{evalh}(x^k, N) = \text{evalh}(x, N)^k$

A problem with this proof

- We are having problems with the verification of property 6.
- Properties 1,2 and 5 are very costly (triple induction).
- It is possible that the solver gets stuck due to the exponential blowup of unrolling so many recursive functions.
- If z3 gets stuck trying to verify some property, then Inox won't be able to recover and the proof will fail. This can happen when the formula becomes too hard for z3 to reason about.

Definition (Normalization procedure)

Let $x \in \tau(V)$, compute the norm of x as follows:

$$\text{norm}(x, V) = \begin{cases} x \in R & x \\ x = v_i & \text{pop}(i, \text{POW}(1, 1, 0)) \\ x = \text{opp}(y) & \ominus \text{norm}(y, V) \\ x = y + z & \text{norm}(y, V) \oplus \text{norm}(z, V) \\ x = y * z & \text{norm}(y, V) \otimes \text{norm}(z, V) \\ x = y^k & \text{norm}(y, V)^k \end{cases}$$

Theorem

Let $f \in \tau(V)$ be a polynomial expression and reset A codomain to N . Then, we have $norm(f, V) \in \mathcal{H}$ and $evalh(norm(f, V), N) = evalp(f, A)$.

Assuming $norm(f_1, V) = norm(f_2, V)$ we have $evalp(f_1, A) = evalh(norm(f_1, V), N) = evalh(norm(f_2, V), N) = evalp(f_2, A)$ We can do better than that:

Theorem

Let $f, g \in \tau(V)$ where V is a fixed set of variables. Then we have

$$norm(f, V) = norm(g, V) \iff \forall N. evalp(f, A) = evalp(g, A)$$

Where V, N are finite sets of same size and A denotes the mapping $V \rightarrow N$ such that $v_i \rightarrow n_i$.

- 1 Introduction
- 2 Welder fundamentals
- 3 Elliptic curves notions
- 4 The naive approach
- 5 The tactic
- 6 Towards the proof of associativity**
- 7 Future work

Let us fix:

- Three points $P_1 = (X_1, Y_1)$, $P_2 = (X_2, Y_2)$ and $P_3 = (X_3, Y_3)$.
- Variables given by a list $V = (Y_0, Y_1, Y_2, X_0, X_1, X_2)$.
- Values given by the corresponding list $N = (y_0, y_1, y_2, x_0, x_1, x_2)$.
- A function $A : V \rightarrow R$ such that $A(v_i) = n_i$.

- There is still one degree of freedom in the equations.
- Equations of the form: $y_j^2 = x_j^3 + Ax + B$.
- $rewrite(h, j)$ rewrites polynomial $h \in \mathcal{H}$ for equation numbered by j .
- We do so for every equation for $\sigma \in \tau(V)$:

$$reduce(\sigma) = rewrite(rewrite(rewrite(norm(\sigma, V), 0), 1), 2)$$

- $split(h, j, k)$ is the core of the rewriting procedure.
- It will leave us with no expression that depends on a term y_j^2 , but a term that does not contain variable y_j and a term that is linear on variable y_j .
- Uses equations of the curve as rewrite rules.

Definition (Splitting procedure)

Let $h \in \mathcal{H}$, $j \in \{0, 1, 2\}$ and $k \in \mathbb{N}$:

- If $h \in \mathbb{Z} \vee j < k$ then $(h, 0)$
- If $j \geq k$, $h = POP(i, p) \wedge (p_0, p_1) = split(p, j, k + i)$ then

$$(pop(i, p_0), pop(i, p_1))$$

- If $j \geq k$, $h = POW(i, p, q)$, $(p_0, p_1) = split(p, j, k)$ and $(q_0, q_1) = split(q, j, k + 1)$ then
 - If $j > k$ then $(pow(i, p_0, q_0), pow(i, p_1, q_1))$.
 - If $j = k \wedge i$ is even then

$$((\Theta^{\frac{i}{2}} \otimes p_0) \oplus pop(1, q_0), (\Theta^{\frac{i}{2}} \otimes p_1) \oplus pop(1, q_1))$$

- If $j = k \wedge i$ is odd then

$$((\Theta^{\frac{i+1}{2}} \otimes p_1) \oplus pop(1, q_0), (\Theta^{\frac{i-1}{2}} \otimes p_0) \oplus pop(1, q_1))$$

Definition (Rewrite procedure)

Let $h \in \mathcal{H}$, $j \in \{0, 1, 2\}$, $(h_0, h_1) = \text{split}(h, j, 0)$ and $\sigma \in \tau(V)$ then:

$$\text{rewrite}(h, j) = h_0 \oplus (h_1 \otimes \text{norm}(Y_j, V))$$

Properties to proof in this part

Name	Theorem
Θ -evaluation	$evalh(\Theta, N^{(j)}) = x_j^3 + Ax_j + B$
Split evaluation	$evalh(h, N^{(k)}) = evalh(h_0, N^{(k)}) + y_j \cdot evalh(h_1, N^{(k)})$
Rewrite evaluation	$evalp(rewrite(h, j), A) = evalp(h, A)$
Reduce evaluation	$reduce(\sigma) = reduce(\tau) \implies evalp(\sigma) = evalp(\tau)$

Table : Properties for the reduction procedure

Rationalizing equations by projecting points

- Up to now we assumed the coefficients of our equations were in a ring.
- A rationalizer procedure is not fully satisfactory. Based on heuristics.
- Define a partial addition operation for computing the addition with ring coefficients.
- To transform the equations use a projection-like mapping.
- Restrict to not infinite points (triples with non-zero last component).

Rationalizing equations by projecting points

Let F^{3*} be the set of triples over F with non-zero last component.

Definition (Projection mapping)

Projection is a mapping $p : F^{3*} \rightarrow F^2$ such that:

$$p((u, v, w)) = \left(\frac{u}{w^2}, \frac{v}{w^3} \right)$$

In this case we say that (u, v, w) is a representative of $p((u, v, w))$ where the canonical representative for point (x, y) is $(x, y, 1)$.

Definition (Addition on \mathbb{F}^{3*})

Given $P, Q \in \mathbb{F}^{3*}$ their addition $P \oplus Q = (m', n', z')$ is defined as:

- If $P = Q = (m, n, z)$ then

$$z' = 2nz$$

$$w' = 3m^2 + Az^4$$

$$m' = w'^2 - 8n^2m$$

$$n' = 4n^2w'(m - w'^2 + 8n^2m) - 8n^4$$

Definition (Addition on \mathbb{F}^{3*})

Given $P, Q \in \mathbb{F}^{3*}$ their addition $P \oplus Q = (m', n', z')$ is defined as:

- If $P = (x, y, 1)$ and $Q = (m, n, z) \neq P$ then

$$z' = z(m - xz^2)$$

$$m' = (n - yz^3)^2 - (m - xz^2)^2(m + xz^2)$$

$$n' = (n - yz^3)(xz'^2 - m') - yz'^3$$

Lemma (Relation with the curve addition)

Let $\mathcal{P}, \mathcal{Q} \in \mathbb{F}^{3}$ and set $P = p(\mathcal{P}), Q = p(\mathcal{Q})$. Assume \oplus is defined for \mathcal{P} and \mathcal{Q} then:*

$$P + Q = p(\mathcal{P} \oplus \mathcal{Q})$$

Rationalizing equations by projecting points

- Want to work on equations and not on actual values.
- The rationalizer procedure has to be extended taking coordinates as variables and operations between them as polynomial equations.
- The lemmas that precede can be generalized directly if we take now p' to be the composition of the previous p with the *evalp* function:

$$p' : \tau(V)^{3*} \rightarrow \mathbb{F}^2$$

such that

$$p'((\mu, \nu, \zeta)) = p(\text{evalp}((\mu, \nu, \zeta)))$$

Where *evalp* acts in each of the components of the triple.

Rationalizing equations by projecting points

Finally, define when a triple $(\mu, \nu, \zeta) \in \tau(V)$ represent a point of the curve when evaluated:

Definition (Curve-point projection)

Given $(\mu, \nu, \zeta) \in \tau(V)$, consider the polynomial:

$$\tau_{\mu, \nu, \zeta} = \nu^2 - (\mu^3 + A\mu\zeta^4 + B\zeta^6)$$

We say that (μ, ν, ζ) is a curve-point projection if $\text{reduce}(\tau_{\mu, \nu, \zeta}) = 0$.

This definition gives us the desired property:

Lemma

If (μ, ν, ζ) is a curve-point projection and $P = p'((\mu, \nu, \zeta))$ then $P \in E(K)$.

Rationalizing equations by projecting points

In the projective space, points that lie on the same line, represent the same plane point and are therefore considered to be equivalent. Here we want a similar property with our projection mapping:

Definition (Equivalent points in $\tau(V)^{3*}$)

Let $P = (\mu, \nu, \zeta'), Q = (\mu', \nu', \zeta') \in \tau(V)^{3*}$ then:

$$P \sim Q \iff \text{reduce}(\sigma) = \text{reduce}(\sigma') \wedge \text{reduce}(\tau) = \text{reduce}(\tau')$$

Where $\sigma = \mu\zeta'^2, \sigma' = \mu'\zeta'^2, \tau = \nu\zeta'^3, \tau' = \nu'\zeta'^3$.

As in the projective plane, equivalent lines represent the same point:

Lemma (Equivalence implies equality of affine points)

Let $\mathcal{P}, \mathcal{Q} \in \tau(V)^{3*}$ and set $P = p(\mathcal{P}), Q = p(\mathcal{Q})$. Assume $P, Q \in E(K) \setminus \{\infty\}$ and that $\mathcal{P} \sim \mathcal{Q}$ then $P = Q$.

Verifying the associativity property

- Finally, we can verify the associativity property.
- Some results are purely computational: the unrolling procedure of Inox should be able to calculate them
- Then we have a list of derived results.

In next slides, set

$$\Sigma = \Pi_0 \oplus \Pi_1 = (\mu, \nu, \zeta)$$

$$\Sigma' = \Pi_0 \oplus \Pi_0 = (\mu', \nu', \zeta')$$

$$\phi = ((\mu - X_1 \zeta^2) + 2Y_1 Y_2)^2 - (2Y_1 Y_2)^2$$

$$\psi = (\mu' - X_2 \zeta'^2) \zeta^2$$

Verifying the associativity property

Name	Property
Prop-1	$-(\Pi_0 \oplus \Pi_0) \sim (-\Pi_0) \oplus (-\Pi_0)$
Prop-2	$-(\Pi_0 \oplus \Pi_1) \sim (-\Pi_0) \oplus (-\Pi_1)$
Prop-3	$(-\Pi_0) \oplus (\Pi_0 \oplus \Pi_1) \sim \Pi_0$
Prop-4	$(-\Pi_0) \oplus (\Pi_0 \oplus \Pi_1) \sim \Pi_1$
Prop-5	$\Pi_2 \oplus (\Pi_0 \oplus \Pi_1) \sim \Pi_1 \oplus (\Pi_0 \oplus \Pi_2)$
Prop-6	$\Pi_1 \oplus (\Pi_0 \oplus \Pi_0) \sim \Pi_0 \oplus (\Pi_0 \oplus \Pi_1)$
Prop-7	$(\Pi_0 \oplus \Pi_0) \oplus (\Pi_0 \oplus \Pi_0) \sim \Pi_0 \oplus (\Pi_0 \oplus (\Pi_0 \oplus \Pi_0))$
Prop-8	$(\Pi_0 \oplus \Pi_1) \oplus (\Pi_0 \oplus \Pi_1) \sim \Pi_0 \oplus (\Pi_1 \oplus (\Pi_0 \oplus \Pi_1))$
Prop-9	$reduce(\phi) = reduce(\psi)$
Prop-10	$(0, 0, 1) \oplus (\Pi_0 \oplus (0, 0, 1)) \sim \Pi_0$
Prop-11	$(\Pi_0 \oplus (0, 0, 1)) \oplus (\Pi_0 \oplus (0, 0, 1)) \sim \Pi_0 \oplus \Pi_0$

Table : Computational results derived from the definitions

Verifying the associativity property

Then the set of lemmas that drive towards associativity is shown below. This finishes the proof of the associative law.






Name	Property
Lemma-1	$-(P_0 + P_1) = (-P_0) + (-P_1)$
Lemma-2	If $P_0 + P_1 \neq \ominus P_0$ then $(\ominus P_0) \oplus (P_0 \oplus P_1) = P_1$
Lemma-3	If $P_0 + P_1 \neq P_2, -P_2 \wedge P_0 + P_2 \neq P_1, -P_1$ then $P_2 + (P_0 + P_1) = P_1 + (P_0 + P_2)$
Lemma-4	If $P_0 + P_1 \neq -(P_0 + P_1), P_0 + P_1 \neq -P_1, P_1 + (P_0 + P_1) \neq P_0, -P_0$ then $(P_0 + P_1) + (P_0 + P_1) = P_0 + (P_1 + (P_0 + P_1))$
Lemma-5	If $P_0 + P_1 = -P_0$ then $P_1 = -(P_0 + P_0)$
Lemma-6	If $P_0 + P_1 = -P_2$ then $(P_0 + P_1) + P_2 = P_0 + (P_1 + P_2)$
Lemma-7	$(P_0 + P_0) + P_1 = P_0 + (P_0 + P_1)$
Lemma-8	$(P_0 + (0, 0)) + (P_0 + (0, 0)) = P_0 + ((0, 0) + (P_0 + (0, 0)))$
Lemma-9	$(P_0 + P_1) + P_2 = P_0 + (P_1 + P_2)$

Table : Results derived from the above propositions

- 1 Introduction
- 2 Welder fundamentals
- 3 Elliptic curves notions
- 4 The naive approach
- 5 The tactic
- 6 Towards the proof of associativity
- 7 Future work**

- Converting these to the Montgomery form or the Edwards form.
- Instantiate the field and perform the multiplications algorithms for each case.
- The *HACL** library is to our knowledge the first library that has verified the byte level scalar multiplication.

- Ensure soundness.
- Adding interactivity.
- Incorporate automation of induction.
- Maybe using unification to automatically derive code for theorems.

-  B. Gregoire. “Proving equalities in a commutative ring done right in Coq”. In: *International Conference on Theorem Proving in Higher Order Logics*. Springer Berlin Heidelberg (2005), pp. 98–113.
-  L.C. Paulson. *Logic and computation: interactive proof with Cambridge LCF*. Cambridge University Press, 1987.
-  David M. Russsinoff. “A Computationally Surveyable Proof of the Curve25519 Group Axioms”. In: *Unpublished work* (2015).
-  David M. Russsinoff. “Polynomial Terms and Sparse Horner Normal Form”. In: *Unpublished work* (2015).
-  L. Théry. *Proving the group law for elliptic curves formally*. Tech. rep. INRIA, 2007.



L.C. Washington. *Elliptic curves: number theory and cryptography*. CRC press, 2008.



J. K. Zinzindohoué et al. “HACL*: A Verified Modern Cryptographic Library.” In: (2017).