Mechanizing Elliptic Curve Associativity

Why a Formalized Mathematics Challenge is Useful for Verification of Crypto ARM Machine Code

Joe Hurd

Computer Laboratory
University of Cambridge

Galois Connections
Friday 15 December 2006

Talk Plan

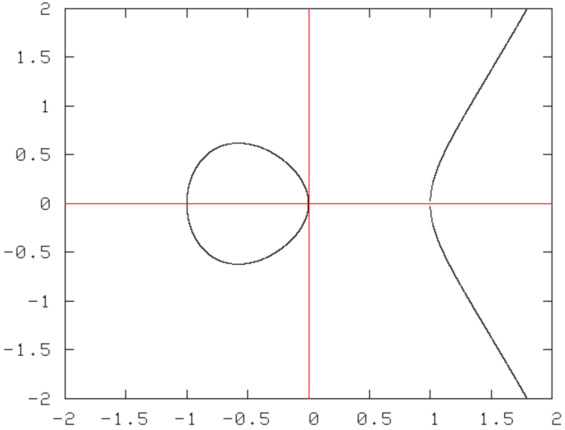
Elliptic Curves

- An elliptic curve over the reals is the set of points (x,y) satisfying an equation of the form

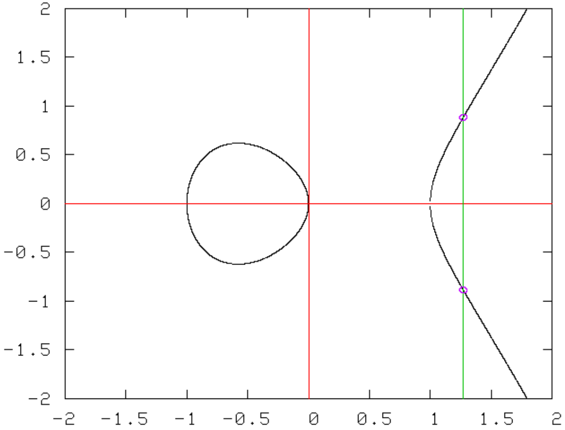
$$E : y^2 = x^3 + ax + b \ .$$

- There is also a 'point at infinity' considered to lie on the elliptic curve, called $\mathcal{O}$.
- It's possible to 'add' two points on an elliptic curve to get a third point on the curve.

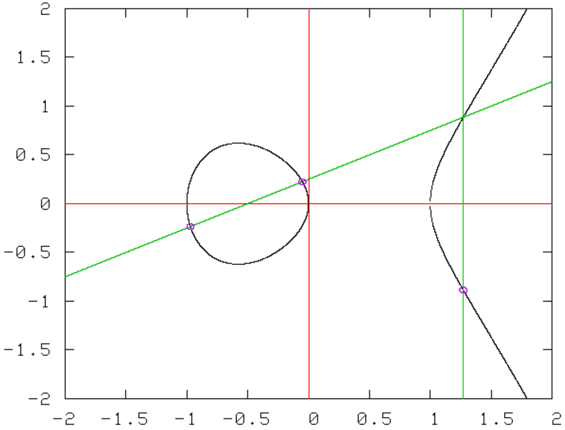
The Elliptic Curve $y^2 = x^3 - x$



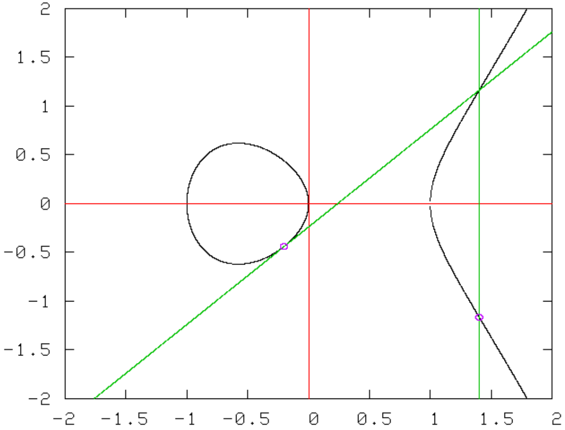
Elliptic Curve Arithmetic: Negation



The Elliptic Curve $y^2 = x^3 - x$: Addition



The Elliptic Curve $y^2 = x^3 - x$: Doubling



The Elliptic Curve Group

Theorem

Elliptic curve addition forms a group, i.e.,

1. $\mathcal{O} + P = P$
2. $-P + P = \mathcal{O}$
3. $(P + Q) + R = P + (Q + R)$

plus the closure conditions $P, Q \in E \implies \mathcal{O}, -P, P + Q \in E$.

Cryptography Based on Groups

- The Discrete Logarithm Problem over a group G tests the difficulty of inverting the power operation:
  - Given $x, y \in G$, find a k such that $x^k = y$.
- Cryptographic operations can be built using this primitive.
  - Elgamal encryption
  - Digital Signature Algorithm
- The level of security depends entirely on the group G.
  - The group of addition modulo n is easily broken.
  - A 'black-box group' requires $\sqrt{|G|}$ group operations to break.
- Standard public key cryptography uses the group of multiplication modulo a large prime.

Elliptic Curve Cryptography

- First proposed in 1985 by Koblitz and Miller.
- Part of the 2005 NSA Suite B set of cryptographic algorithms.
- Certicom the most prominent vendor, but there are many implementations.
- Advantages over standard public key cryptography:
  - Known theoretical attacks much less effective,
  - so requires much shorter keys for the same security,
  - leading to reduced bandwidth and greater efficiency.
- However, there are also disadvantages:
  - Patent uncertainty surrounding many implementation techniques.
  - The algorithms are more complex, so it's harder to implement them correctly.

Elliptic Curve Cryptography: More Secure?

- This table shows equal security key sizes:

standard	elliptic curve
1024 bits	173 bits
4096 bits	313 bits

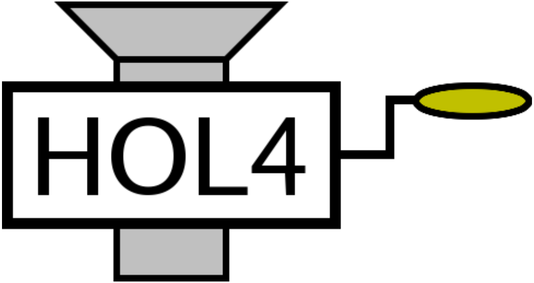
- But. . . there has been less theoretical effort made to attack elliptic curve cryptosystems.

Verified ARM Implementations

- **Motivation:** How to ensure that low level cryptographic software is both correct and secure?
    - Critical application, so need to go beyond bug finding to assurance of correctness.
- **Project goal:** Create formally verified ARM implementations of elliptic curve cryptographic algorithms.
    - Joint project between Cambridge University and the University of Utah, managed by Mike Gordon.

Illustrating the Verification Flow

- Elliptic curve ElGamal encryption
- Key size = 320 bits



- Verified ARM machine code

Verification Guarantee

The verified ARM code should be correct and secure.

1. Functional correctness:
   - Encryption followed by decryption is the identity.
2. A security guarantee:
   - The code correctly implements elliptic curve Elgamal.

Note: Functional correctness relies on the elliptic curve group.

Negation of Elliptic Curve Points (1)

Blake, Seroussi and Smart define negation of elliptic curve points using affine coordinates:

> *"Let E denote an elliptic curve given by*
>
> $$E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$$
>
> *and let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ denote points on the curve. Then*
>
> $$-P_1 = (x_1, -y_1 - a_1x_1 - a_3)\ ."$$

Negation of Elliptic Curve Points (2)

Negation is formalized by cases on the input point, which smoothly handles the special case of $\mathcal{O}$:

Constant Definition

```
curve_neg e =
let f = e.field in
...
let a3 = e.a3 in
curve_case e (curve_zero e)
  (λx1 y1.
     let x = x1 in
     let y = ~y1 - a1 * x1 - a3 in
     affine f [x; y])
```

Elliptic Curve Addition

And now Blake, Seroussi and Smart's definition of point addition:

"Set

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}, \quad \mu = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1}$$

when $x_1 \neq x_2$*, and set*

$$\lambda = \frac{3x_1^2 + 2a_2 x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3},$$
$$\mu = \frac{-x_1^3 + a_4 x_1 + 2a_6 - a_3 y_1}{2y_1 + a_1 x_1 + a_3}$$

when $x_1 = x_2$ *and* $P_2 \neq -P_1$*."*

Elliptic Curve Addition (2)

"If

$$P_3 = (x_3, y_3) = P_1 + P_2 \neq \mathcal{O}$$

then x_3 *and* y_3 *are given by the formulae*

$$\begin{aligned} x_3 &= \lambda^2 + a_1\lambda - a_2 - x_1 - x_2 \ , \\ y_3 &= -(\lambda + a_1)x_3 - \mu - a_3 \ .\text{''} \end{aligned}$$

Elliptic Curve Addition (3)

Constant Definition

```
curve_double e =
let f = e.field in
...
let a6 = e.a6 in
curve_case e (curve_zero e)
(λx1 y1.
  let d = & 2 * y1 + a1 * x1 + a3 in
  if d = field_zero f then curve_zero e
  else
   let l = (& 3 * x1 ** 2 + & 2 * a2 * x1 + a4 - a1 * y1) / d in
   let m = (~(x1 ** 3) + a4 * x1 + & 2 * a6 - a3 * y1) / d in
   let x = l ** 2 + a1 * l - a2 - &2 * x1 in
   let y = ~(l + a1) * x - m - a3 in
   affine e.field [x; y])
```

The special case of $P_1 = -P_1$ is handled by the test for $d = 0$.

Elliptic Curve Addition (4)

Constant Definition

```
curve_add e p1 p2 =
if p1 = p2 then curve_double e p1
else
  let f = e.field in
  ...
  let a6 = e.a6 in
  curve_case e p2
    (λx1 y1.
       curve_case e p1
         (λx2 y2.
            if x1 = x2 then curve_zero e
            else
              let d = x2 - x1 in
              let l = (y2 - y1) / d in
              let m = (y1 * x2 - y2 * x1) / d in
              let x = l ** 2 + a1 * l - a2 - x1 - x2 in
              let y = ~(l + a1) * x - m - a3 in
              affine e.field [x; y]) p2) p1
```

Associativity Challenge Problem

Goal (Associativity of Point Addition)

```
∀e ∈ Curve. ∀p q r ∈ curve_points e.

  curve_add e p (curve_add e q r)
    =
  curve_add e (curve_add e p q) r
```

Challenge 1: General Fields

- Elliptic curve addition is a group for *any* underlying field.
  - Can't simply specialize to the real or complex numbers, because cryptography application uses finite fields.
  - Can't simply assume the field elements form an entire type, because that makes it difficult to reason about subfields (the original Galois connection).
- So all the field operations must be partial functions.
- The assisting tool must be able to handle partial functions.

Challenge 2: Case Splitting

- A naive approach expands all the cases in the associativity goal (doubling vs. adding distinct points, etc.).
- This generates about 100 subgoals, each with a slightly different logical context.
- The assisting tool must be able to keep track of all the assumptions in each case.

Challenge 3: Polynomial Normalization

- After splitting into cases the naive approach expands all the polynomials and tries to show them equal.
- The intermediate expressions can become large (millions of symbols).
  - An instance of proof state space explosion?
- The assisting tool must be able to handle large terms.

Computer Algebra Techniques in Theorem Proving

1. Use a computer algebra system as an oracle.
    - Needs careful handling to avoid unsoundness.
2. Use the computer algebra system to compute a witness for the problem, and then verify it in the theorem prover.
    - Sound, but not all problems fit into the model.
3. Implement computer algebra techniques as derived rules.
    - Sound, covers all problems, but might be inefficient.
4. Implement computer algebra algorithms and data structures as object logic functions, prove them correct and execute them in the theorem prover.
    - Sound and efficient (same complexity), but can be difficult.

Simple Polynomial Normalization

- Proving that doubling a point on an elliptic curve results in another point on the curve can be solved naively by multiplying out.
- The normalized polynomials reach 300,000 symbols before cancelling out, which is too big for the HOL theorem prover.
- However, this isn't a big problem for a computer algebra system.
- Would like a simple polynomial normalization algorithm.
  - Today will be used as an ML oracle.
  - One day could be formalized in HOL and proved correct.

Simple Polynomial Normalization

- Consider the following data structure for polynomials:

Type Definition

```
poly = Var of string
     | Sum of (poly,int) finiteMap
     | Prod of (poly,int) finiteMap
```

- Example: $(2x + 3)^6$ is represented as

$$\mathrm{Prod}\,\{\mathrm{Sum}\,\{\mathrm{Var}\;x \mapsto 2,\;\mathrm{Prod}\,\{\} \mapsto 3\} \mapsto 6\}$$

- Note that numbers don't need a special constructor.

Simple Polynomial Normalization

- Simple normalization rules:

$$\begin{array}{rcl} \text{Sum } (\{p \mapsto 0\} \cup M) & \longrightarrow & \text{Sum } M \\ \text{Prod } (\{p \mapsto 0\} \cup M) & \longrightarrow & \text{Prod } M \\ \text{Sum } (\{\text{Sum } M' \mapsto n\} \cup M) & \longrightarrow & \text{Sum } ((n * M') \cup M) \\ \text{Prod } (\{\text{Prod } M' \mapsto n\} \cup M) & \longrightarrow & \text{Prod } ((n * M') \cup M) \end{array}$$

- One complicated normalization rule:

$$\text{Prod } (\{\text{Sum } S \mapsto n\} \cup P) \longrightarrow \text{Sum } (S^n * P)$$

where S^n is the multinomial

$$(x_1 + \cdots + x_m)^n = \sum_{k_1,\ldots,k_m} \binom{n}{k_1, \ldots, k_m} x_1^{k_1} \cdots x_m^{k_m}$$

Simple Polynomial Normalization

- These rules are sufficient to normalize polynomials.
- Though simple, they are efficient enough to prove the closure of point doubling in just a few seconds.
- Notice that the bulk of the work is being done by the data structure, not the algorithm.

> *Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious. [Brooks, 1975]*

Summary

- This talk has proposed the elliptic curve associativity law as a challenge problem for automated reasoning.
- It is a rare instance of a deep mathematical theorem that is needed for a practical low-level verification.
- One way to meet the challenge avoiding case splitting and large expressions would be to mechanize all the abstract algebra used in a mathematical proof.
  - This would be just as impressive!