Formalized Elliptic Curve Cryptography

Joe Hurd, Mike Gordon and Anthony Fox

Computer Laboratory
University of Cambridge

Wednesday 19 April 2006
High Confidence Software and Systems

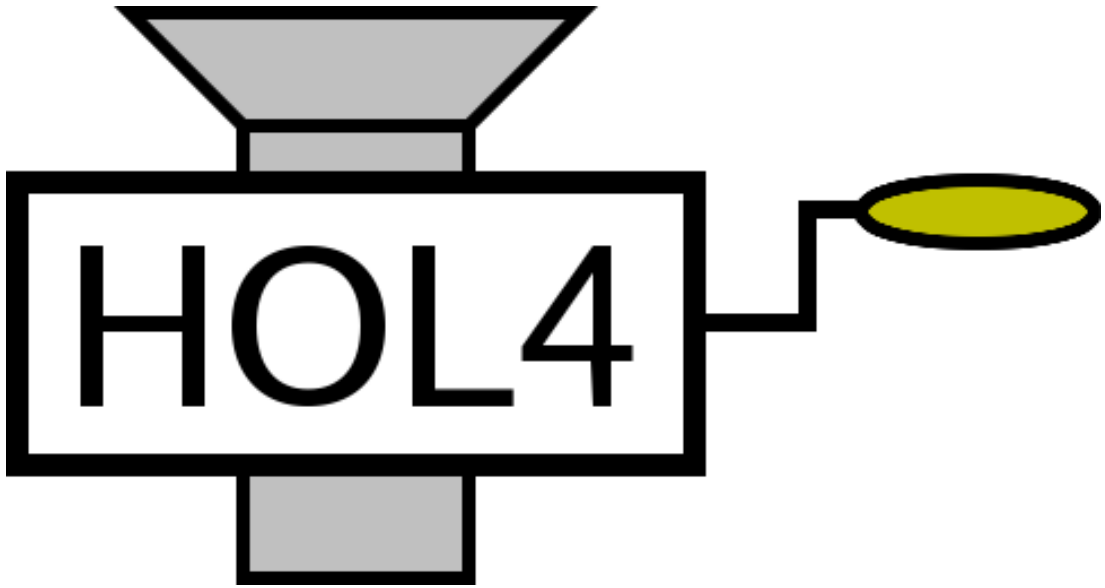
Talk Plan

Verified ARM Implementations

- **Motivation:** How to ensure that low level cryptographic software is both correct and secure?
- **Project goal:** Create formally verified ARM implementations of elliptic curve cryptographic algorithms.
- The following elements are now in place:
  - A formal specification of elliptic curve operations derived from mathematics (Hurd, Cambridge). This talk!
  - A compiler from higher order logic functions to a low level assembly language (Slind, Utah).
  - A very high fidelity model of the ARM instruction set derived from a processor model (Fox, Cambridge).

Illustrating the Verification Flow

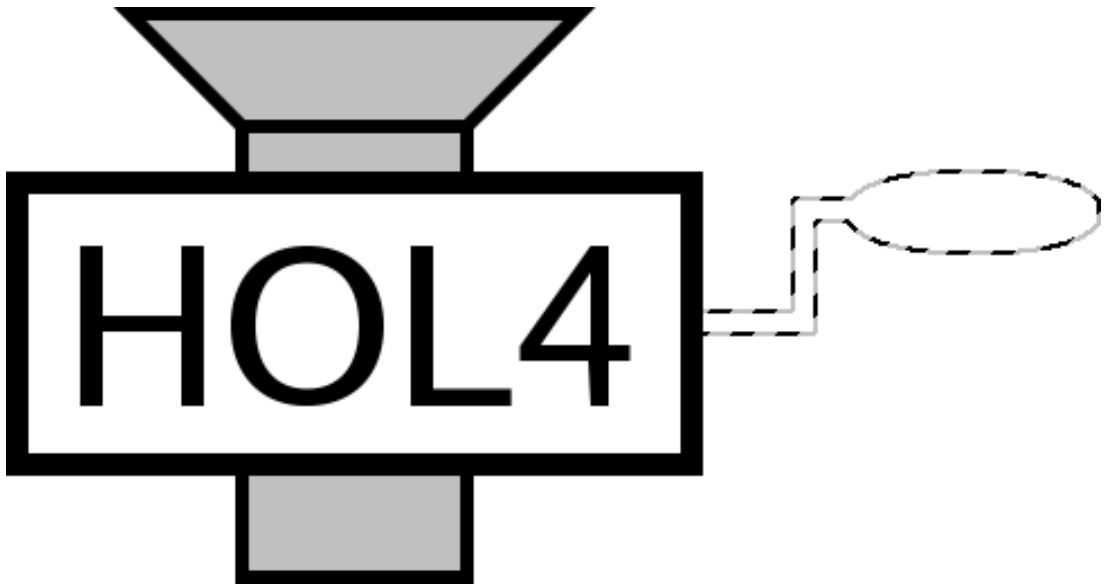
- Elliptic curve ElGamal encryption
- Key size = 320 bits



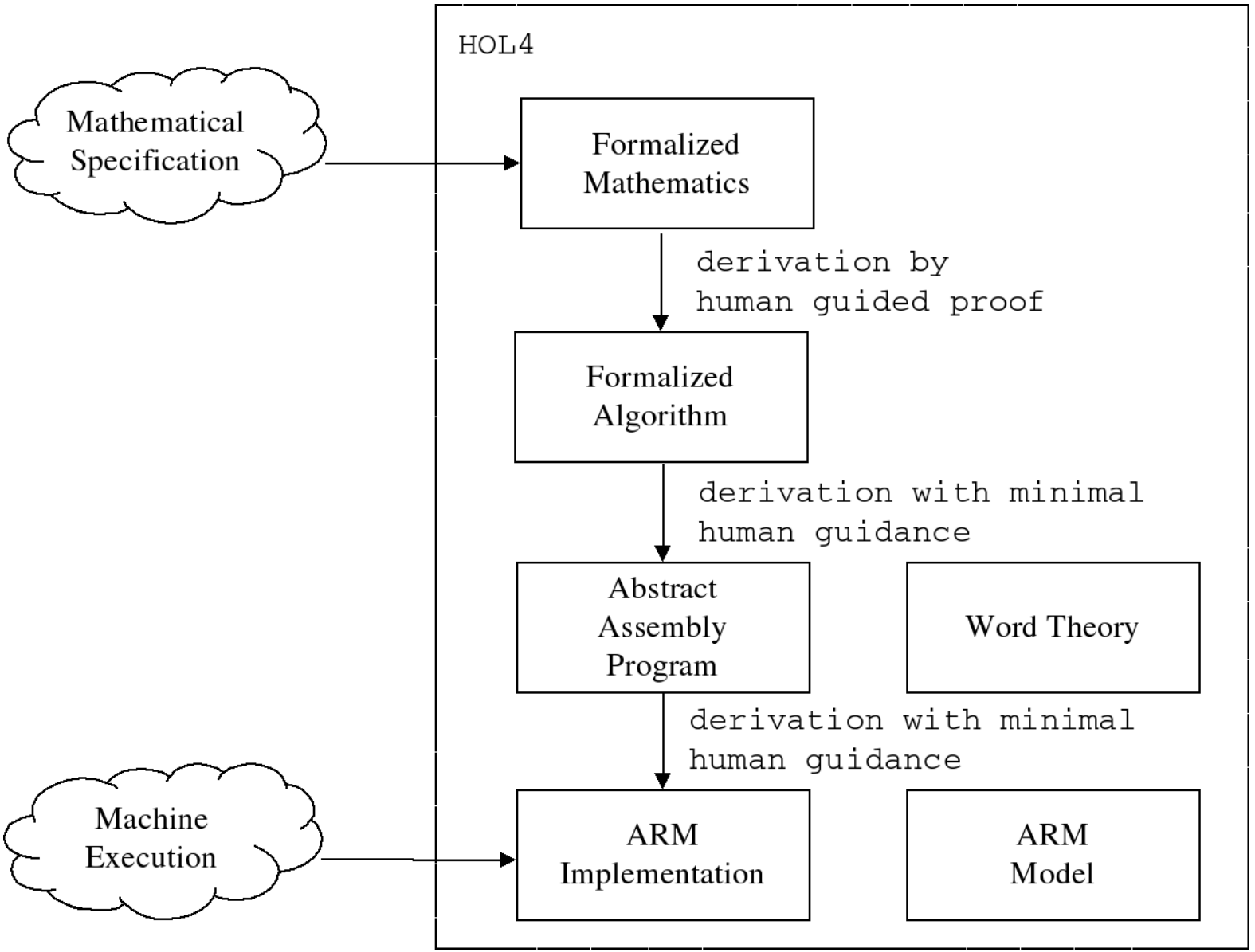
- Verified ARM machine code

Illustrating the Verification Flow

- Elliptic curve ElGamal encryption
- Key size = 320 bits



- Verified ARM machine code


HOL4
Mathematical Specification
Formalized Mathematics
derivation by human guided proof
Formalized Algorithm
derivation with minimal human guidance
Abstract Assembly Program
Word Theory
derivation with minimal human guidance
Machine Execution
ARM Implementation
ARM Model

Assumptions and Guarantees

- Assumptions that must be checked by humans:
  - **Specification:** The formalized theory of elliptic curve cryptography is faithful to standard mathematics. This talk!
  - **Model:** The formalized ARM machine code is faithful to the real world execution environment.
- Guarantee provided by formal methods:
  - The resultant block of ARM machine code faithfully implements an elliptic curve cryptographic algorithm.
  - Functional correctness + a security guarantee.
- Of course, there is also an implicit assumption that the HOL4 theorem prover is working correctly.

Assurance of the Specification

How can evidence be gathered to check whether the formal specification of elliptic curve cryptography is correct?

1. Comparing the formalized version to a standard mathematics textbook.
2. Deducing properties known to be true of elliptic curves.
3. Deriving checkable calculations for example curves.

This talk will illustrate all three methods.

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.

Elliptic Curve Cryptography: A Comparison

Standard Public Key Cryptography

- Needed: a large prime p and a number g.
- Operation: multiplication mod p.
- One-way operation: $k \mapsto g^k \bmod p$.

Elliptic Curve Cryptography

- Needed: an elliptic curve E and a point p.
- Operation: adding points on E.
- One-way operation: $k \mapsto p + \cdots + p$ (k times).

Formalization in HOL4

- Formalized theory of elliptic curves mechanized in the HOL4 theorem prover.
- Currently about 4500 lines of ML, comprising:
  - 3500 lines of definitions and theorems; and
  - 1000 lines of custom proof tools.
- Complete up to the theorem that elliptic curve arithmetic forms an Abelian group.
- Formalizing this highly abstract theorem will add evidence that the specification is correct. . .
- . . . but is anyway required for the formal verification of elliptic curve cryptographic operations.

Source Material

- The primary way to demonstrate that the specification of elliptic curve cryptography is correct is by comparing it to standard mathematics.
- The definitions of elliptic curves, rational points and elliptic curve arithmetic that we present come from the source textbook for the formalization (*Elliptic Curves in Cryptography*, by Ian Blake, Gadiel Seroussi and Nigel Smart.)
- A guiding design goal of the formalization is that it should be easy for an evaluator to see that the formalized definitions are a faithful translation of the textbook definitions.

Elliptic Curves

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

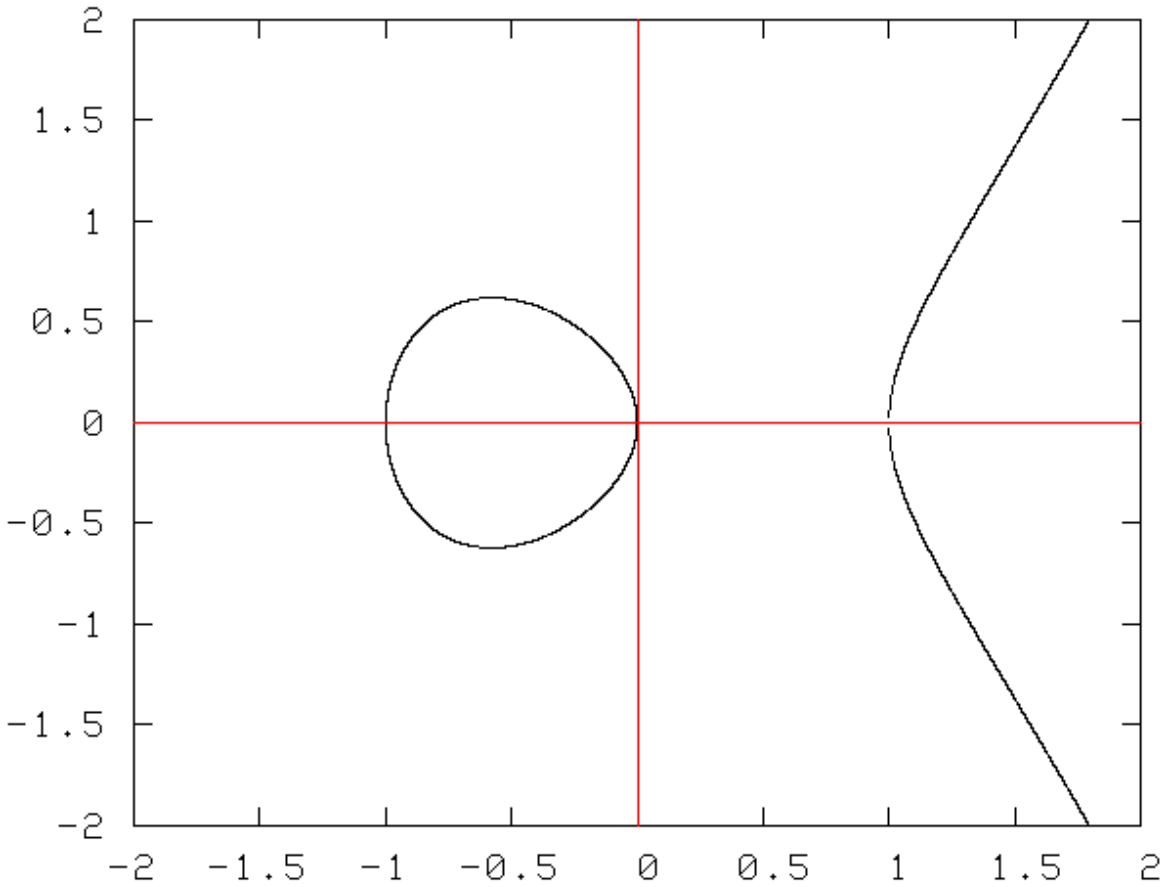
$$E : y^2 = x^3 + ax + b \ .$$

- Despite the name, they don't look like ellipses!
- It's possible to 'add' two points on an elliptic curve to get a third point on the curve.
- Elliptic curves are used in number theory; Wiles proved Fermat's Last Theorem by showing that the elliptic curve

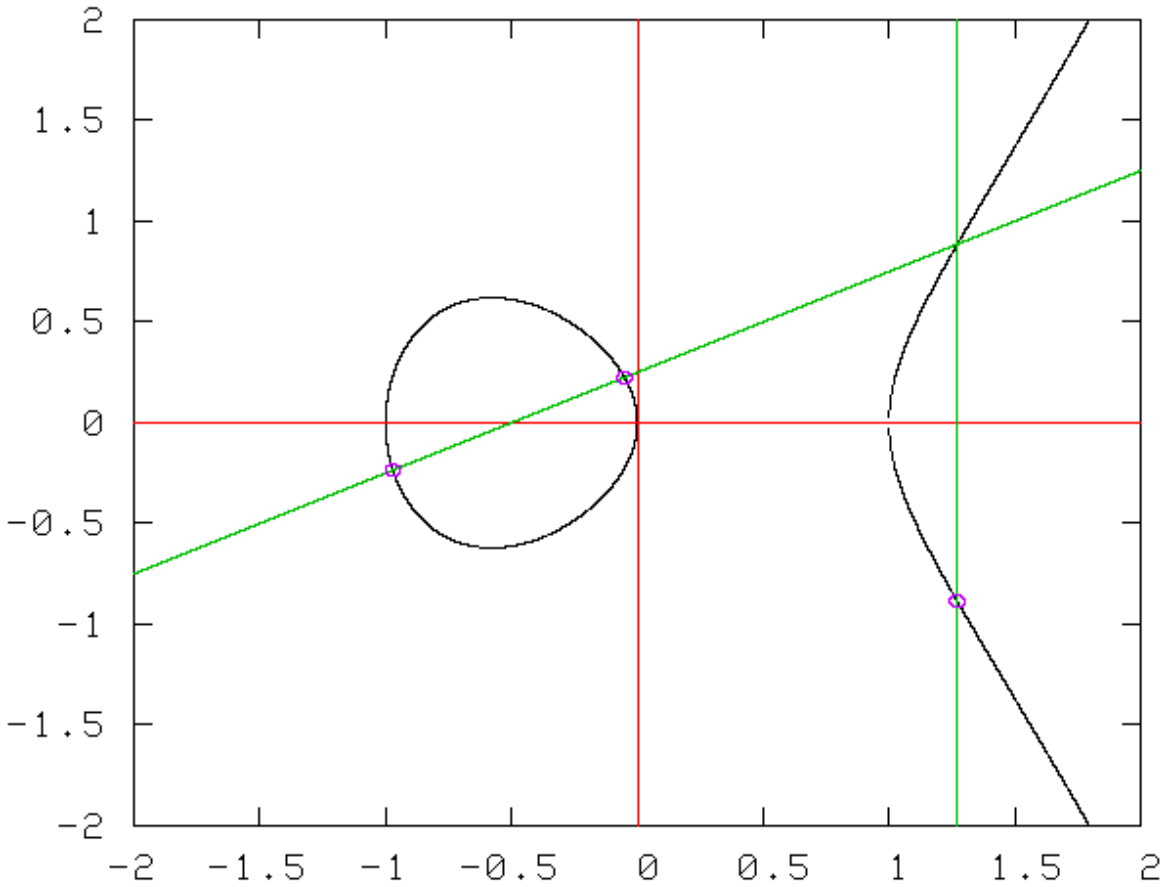
$$y^2 = x(x - a^n)(x + b^n)$$

generated by a counter-example $a^n + b^n = c^n$ cannot exist.

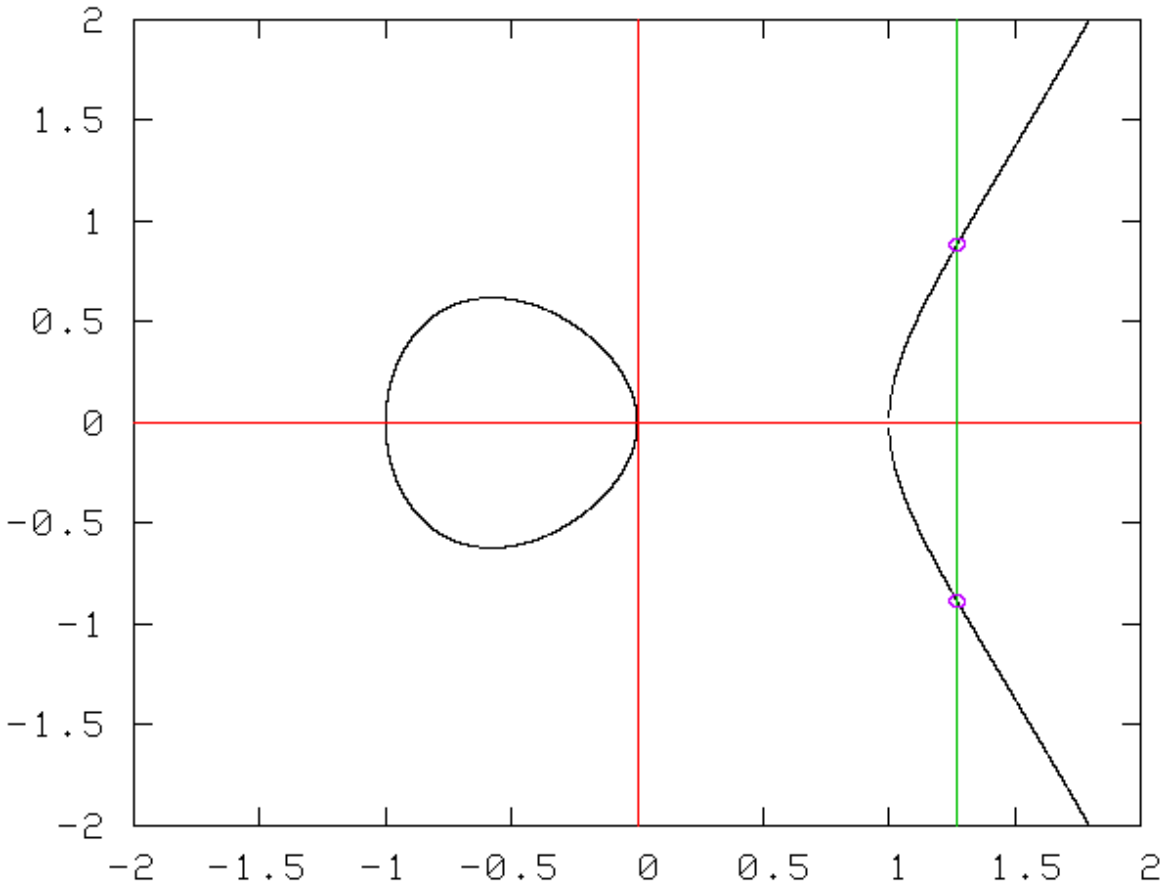
The Elliptic Curve $y^2 = x^3 - x$



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



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



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 $P_1 = (x_1, y_1)$ [denote a point] 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])
```

$$\text{“}-P_1 = (x_1, -y_1 - a_1 x_1 - a_3)\text{”}$$

Negation of Elliptic Curve Points (3)

The curve_case function makes it possible to define functions on elliptic curve points by separately treating the 'point at infinity' $\mathcal{O}$ and the other points (x, y):

Theorem

```
⊢ ∀e ∈ Curve. ∀z f.
    (curve_case e z f (curve_zero e) = z) ∧
    ∀x y. curve_case e z f (affine e.field [x; y]) = f x y
```

Negation of Elliptic Curve Points (4)

Negation maps points on the curve to points on the curve.

Theorem

$$\vdash \forall \texttt{e} \in \texttt{Curve}.\ \forall \texttt{p} \in \texttt{curve\_points e}.\ \texttt{curve\_neg e p} \in \texttt{curve\_points e}$$

Verified Elliptic Curve Calculations

- It is often desirable to derive calculations that provably follow from the definitions.
  - Can be used to sanity check the formalization,
  - or provide a 'golden' test vector.
- A custom proof tool performs these calculations.
  - The tool mainly consists of unfolding definitions in the correct order.
  - The numerous side conditions are proved with predicate subtype style reasoning.

Verified Calculations: Elliptic Curves Points

Use an example elliptic curve from a textbook exercise (Koblitz, 1987).

Example

```
ec = curve (GF 751) 0 0 1 750 0
```

Prove that the equation defines an elliptic curve and that two points given in the exercise lie on the curve.

Example

```
⊢ ec ∈ Curve
⊢ affine (GF 751) [361; 383] ∈ curve_points ec
⊢ affine (GF 751) [241; 605] ∈ curve_points ec
```

Verified Calculations: Elliptic Curve Arithmetic

Perform some elliptic curve arithmetic calculations and test that the results are points on the curve.

Example

```
⊢ curve_neg ec (affine (GF 751) [361; 383]) =
    affine (GF 751) [361; 367]

⊢ affine (GF 751) [361; 367] ∈ curve_points ec

⊢ curve_add ec (affine (GF 751) [361; 383])
               (affine (GF 751) [241; 605]) =
  affine (GF 751) [680; 469]

⊢ affine (GF 751) [680; 469] ∈ curve_points ec

⊢ curve_double ec (affine (GF 751) [361; 383]) =
    affine (GF 751) [710; 395]

⊢ affine (GF 751) [710; 395] ∈ curve_points ec
```

Doing this revealed a typo in the formalization of point doubling!

The Elliptic Curve Group

The (current) high water mark of the HOL4 formalization of elliptic curves is the ability to define the elliptic curve group.

Constant Definition

```
curve_group e =
<| carrier := curve_points e;
   id := curve_zero e;
   inv := curve_neg e;
   mult := curve_add e |>
```

Cryptography Based On Groups

- Many cryptographic algorithms make use of the Discrete Logarithm Problem over a group G:
  - Given $x, y \in G$, find a k such that $x^k = y$.
- The difficulty of this problem depends on the group G.
- For some groups, such as integer addition modulo n, the problem is easy.
- For some groups, such as multiplication modulo a large prime p, the problem is difficult.
- Warning: the number field sieve can solve this in sub-exponential time.

ElGamal Encryption (1)

The ElGamal encryption algorithm uses an instance $g^x = h$ of the Discrete Logarithm Problem.

1. Alice obtains a copy of Bob's public key (g, h).
2. Alice generates a randomly chosen natural number $k \in \{1, \ldots, \sharp G - 1\}$ and computes $a = g^k$ and $b = h^k m$.
3. Alice sends the encrypted message (a, b) to Bob.
4. Bob receives the encrypted message (a, b). To recover the message m he uses his private key x to compute

$$ba^{-x} = h^k m g^{-kx} = g^{xk-xk} m = m \ .$$

ElGamal Encryption (2)

Formalize the ElGamal encryption packet that Alice sends to Bob.

Constant Definition

```
elgamal G g h m k =
(group_exp G g k, G.mult (group_exp G h k) m)
```

This follows the algorithm precisely.

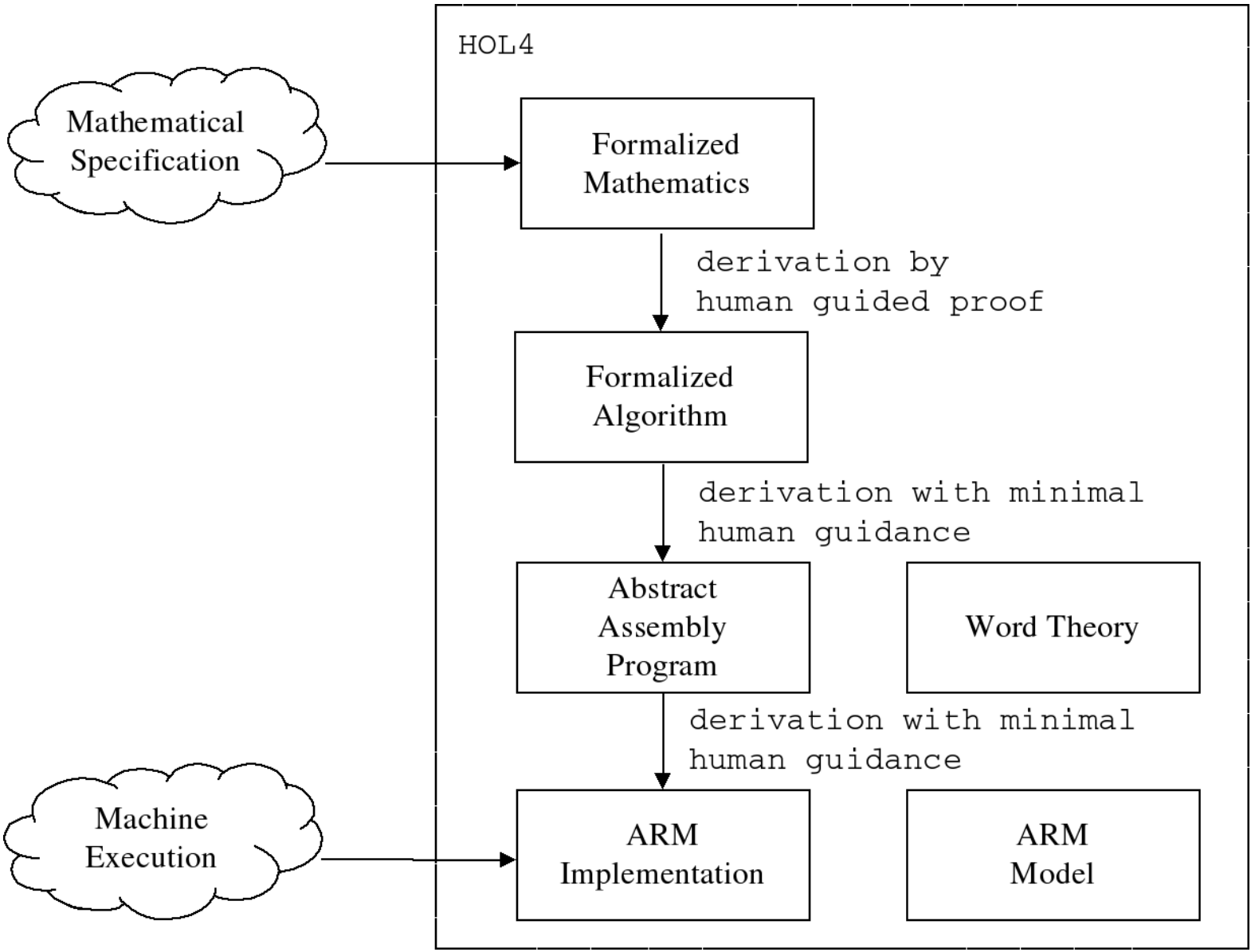
ElGamal Encryption (3)

Prove the theorem that Bob can decrypt the ElGamal encryption packet to reveal the message (assuming he knows his private key).

Theorem

$$\vdash \forall G \in \texttt{Group}.\ \forall g\ h\ m \in G.\texttt{carrier}.\ \forall k\ x.$$
$$(h = \texttt{group\_exp}\ G\ g\ x) \implies$$
$$(\texttt{let}\ (a,b) = \texttt{elgamal}\ G\ g\ h\ m\ k\ \texttt{in}$$
$$G.\texttt{mult}\ (G.\texttt{inv}\ (\texttt{group\_exp}\ G\ a\ x))\ b = m)$$

This diverges slightly from the textbook algorithm by having Bob compute $a^{-x}b$ instead of ba^{-x}, but results in a stronger theorem since the group G does not have to be Abelian.


HOL4
Mathematical Specification
Formalized Mathematics
derivation by human guided proof
Formalized Algorithm
derivation with minimal human guidance
Abstract Assembly Program
Word Theory
derivation with minimal human guidance
Machine Execution
ARM Implementation
ARM Model

Summary

- This talk has described three techniques to validate a theory of elliptic curve cryptography mechanized in the HOL4 theorem prover.
- Assurance is needed: the formalized theory will be used to write specifications for verifying ARM implementations of elliptic curve cryptography.
- In future could be 'retargeted' to verify Cryptol programs or generate verified test vectors for use outside the theorem prover.