Efficient arithmetic
on elliptic curves

D. J. Bernstein
University of Illinois at Chicago
Classic question about the Diffie-Hellman system:
How quickly can we compute $n$th powers mod $p$?

Assume that someone gives you $p$; e.g. $p = 2^{262} - 5081$.

This talk asks the analogous question for elliptic-curve Diffie-Hellman:
How quickly can we compute $n$th multiples in an elliptic-curve group?

“Elliptic-curve scalar multiplication.”
Assume that someone gives you a field and an elliptic curve.

E.g. NIST P-224: the elliptic curve
\[ y^2 = x^3 - 3x + a_6 \]
over \( \mathbb{Z}/p \).
Here \( p = 2^{224} - 2^{96} + 1 \)
and \( a_6 = 189582862855666608 \\
00040866854449392 \\
64155046809686793 \\
21075787234672564. \)

E.g. NIST P-256.

E.g. Curve25519.
Your task: Given \((x, y)\) on curve, and given integer \(n \geq 0\), compute \(n\)th multiple of \((x, y)\) in the elliptic-curve group.

Warning: Answer is not \((nx, ny)\) unless you’re extremely lucky. Elliptic-curve point addition is not vector addition; \((x, y) + (x', y')\) is almost never \((x + x', y + y')\).

Can emphasize this by changing notation: \(\oplus\), \(\Theta\), \([n]\), etc. But this talk uses simplified notation.
Multiples via additions

Typical recursive formulas:

\[ 2P = P + P. \]
\[ 3P = 2P + P. \]
\[ 4P = 2P + 2P. \]
\[ 5P = 3P + 2P. \]
\[ 6P = 3P + 3P. \]
\[ 7P = 5P + 2P. \]
\[ 2nP = 7P + (n - 7)P \text{ if } 4 \leq n < 8. \]
\[ (2n + 1)P = 2nP + P \text{ if } 4 \leq n < 8. \]
\[ (4n + 1)P = 4nP + P \text{ if } 4 \leq n < 8. \]
\[ (4n + 3)P = 4nP + 3P \text{ if } 4 \leq n < 8. \]
\[ 2nP = nP + nP \text{ if } 8 \leq n. \]
\[ (8n + 1)P = 8nP + P \text{ if } 4 \leq n. \]
\[ (8n + 3)P = 8nP + 3P \text{ if } 4 \leq n. \]
\[ (8n + 5)P = 8nP + 5P \text{ if } 4 \leq n. \]
\[ (8n + 7)P = 8nP + 7P \text{ if } 4 \leq n. \]
This “addition chain”
(“length-3 sliding windows”) uses $\approx \lg n$ doublings and $\approx 0.25 \lg n$ more additions to compute $nP$ for average $n$.

e.g. $\approx 320$ additions for average $n \in \{0, 1, \ldots, 2^{256} - 1\}$.

Some easy improvements from fast negation on elliptic curves: $(16n - 7)P = 16nP - 7P$, etc.
Also use “endomorphisms” for “Koblitz curves,” “GLV curves.”

More complicated methods replace 0.25 by $\approx 1/\lg \lg n$. 
Explicit doubling formulas

On curve $y^2 = x^3 - 3x + a_6$:

$2(x, y) = (x'', y'')$ where

$\lambda = (3x^2 - 3)/2y,$

$x'' = \lambda^2 - 2x,$

$y'' = \lambda(x - x'') - y.$

7 subs etc., 2 squarings,
1 more mult, 1 division.

How do we divide efficiently in a finite field?
\( f/g = fg^{p-2} \) in prime field \( \mathbb{Z}/p \).
Can compute \( g^{p-2} \) with
\[ \approx \lg p \] squarings and \[ \approx (\lg p)/\lg \lg p \] more mults.

e.g. \( p = 2^{224} - 2^{96} + 1: \)
223 squarings, 11 more mults.

More generally, \( f/g = fg^{q-2} \)
in any field of size \( q \).

There are faster division methods (e.g. “Euclid”—beware timing attacks!); smaller “I/M ratio.”
Special methods for some fields.
Speedup: delay divisions

Division costs many mults even with fastest division methods.

Save time by delaying divisions.

Naive division-delay method: Store field elements as fractions until end of computation. Divide once before output.

Mult fractions with 2 field mults. Divide fractions with 2 field mults. Add fractions with 3 field mults.
Speedup: unify denominators

For elliptic-curve doubling, have denominator $2y$
in $\lambda = (3x^2 - 3)/2y$;
denominator $(2y)^2$
in $x'' = \lambda^2 - 2x$;
denominator $(2y)^3$
in $y'' = \lambda(x - x'') - y$.

Subsequent computations will perform separate computations on the denominators $(2y)^2, (2y)^3$ of $x'', y''$.

Save time by manipulating denominators together.
“Jacobian coordinates”:
Store \((x, y, z)\) to represent elliptic-curve point \((x/z^2, y/z^3)\).

\[2(x/z^2, y/z^3) = (x'', y'')\] where
\[
\lambda = (3(x/z^2)^2 - 3)/2(y/z^3)
\]
\[= \alpha/2yz \text{ with } \alpha = 3x^2 - 3z^4;\]
\[x'' = \lambda^2 - 2(x/z^2)\]
\[= (\alpha^2 - 8xy^2)/(2yz)^2;\]
\[y'' = \lambda((x/z^2) - x'') - (y/z^3)\]
\[= (12xy^2\alpha - \alpha^3 - 8y^4)/(2yz)^3.\]
\[2(x/z^2, y/z^3) = (x_2/z_2^2, y_2/z_2^3)\]
where \(z_2 = 2yz\),
\[\alpha = 3x^2 - 3z^4,\]
\[x_2 = \alpha^2 - 8xy^2,\]
\[y_2 = \alpha(4xy^2 - x_2) - 8y^4.\]

Easily compute with 6 squarings, 3 more mults: \(x^2, z^2, z^4, y^2, y^4, yz, xy^2, \alpha^2, \alpha(\ldots)\).

Also some subs, doublings, etc.

Use fast field arithmetic:
e.g., can delay carries and reductions in computing \(y_2\).
Speedup: difference of squares

Can compute $3x^2 - 3x^4$ as $3(x - x^2)(x + x^2)$.

Replace 3 squarings by 1 mult, 1 squaring. Revised total: 4 squarings, 4 more mults.

Note:
$3x^2 - 3x^4$ came from $3x^2 - 3$, derivative of $x^3 - 3x + a_6$. Wouldn’t have same speedup for, e.g., $x^3 - 5x + a_6$. 
Speedup: $f^2, g^2, 2fg$

After computing $f^2$ and $g^2$ can compute $2fg$
as $(f + g)^2 - f^2 - g^2$.

In particular:
After computing $y^2$ and $z^2$
can compute $2yz$
as $(y + z)^2 - y^2 - z^2$.

Replace 1 mult with 1 squaring.
Revised total: 5 squarings, 3 more mults.
Explicit addition formulas

Similar speedups in formulas for adding distinct points.

5 squarings, 11 more mults.

Again some opportunities to delay carries, etc.
**Speedup: cache results**

In adding \((x_1/z_1^2, y_1/z_1^3)\) to \((x_2/z_2^2, y_2/z_2^3)\), compute many intermediates, including \(z_1^2, z_1^3\).

Often add same point again to a different point; can reuse \(z_1^2, z_1^3\).

“Chudnovsky coordinates.”
Speedup: delay fewer divisions?

Faster divisions sometimes justify delaying fewer divisions.

e.g. Do we really need fractions for $P, 3P, 5P, 7P$?


“Mixed coordinates.”

Sometimes worthwhile, depending on division speed.
Montgomery coordinates

On elliptic curves with “Montgomery form”
\[ y^2 = x^3 + a_2 x^2 + x, \]
preferably with small \((a_2 - 2)/4:\)
\[ n(x_1, \ldots) = (x_n/z_n, \ldots) \]
where \(z_1 = 1; \]
\[ x_{2m} = (x_m^2 - z_m^2)^2; \]
\[ z_{2m} = 4x_m z_m (x_m^2 + a_2 x_m z_m + z_m^2); \]
\[ x_{2m+1} = 4(x_m x_{m+1} - z_m z_{m+1})^2; \]
\[ z_{2m+1} = 4(x_m z_{m+1} - z_m x_{m+1})^2 x_1. \]

Can also figure out \(y,\)
or use cryptographic protocols that ignore \(y.\)
Assuming \((a_2 - 2)/4\) small, main operations are
4 squarings, 5 more mults
for each bit of \(n\).

Compare to Jacobian coordinates:
each bit of \(n\) has
5 squarings, 3 more mults,
and on occasion
5 more squarings, 11 more mults.

Montgomery form is better
if \(n\) is not gigantic.