Efficient arithmetic on elliptic curves in large characteristic

D. J. Bernstein
University of Illinois at Chicago

Fix a field and an elliptic curve.

e.g. NIST P-224: the elliptic curve $y^2 = x^3 - 3x + a_6$ over $\mathbf{Z}/p$.
Here $p = 2^{224} - 2^{96} + 1$
and $a_6 = \begin{array}{l}18958286285566608\\00040866854449392\\64155046809686793\\21075787234672564.\end{array}$

e.g. NIST P-256: the elliptic curve $y^2 = x^3 - 3x + \cdots$ over $\mathbf{Z}/p$ where $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

e.g. Curve25519: the elliptic curve $y^2 = x^3 + 486662x^2 + x$ over $\mathbf{Z}/p$ where $p = 2^{255} - 19$.

“Elliptic-curve scalar multiplication”: Given (x, y) on curve, and given integer $n \geq 0$, compute nth multiple of (x, y) in the elliptic-curve group.

This is the bottleneck in elliptic-curve Diffie-Hellman.

The big question: How quickly can we do this?

Many variations of problem: e.g. $m, n, P, Q \mapsto mP + nQ$, critical for elliptic-curve signatures.

Review of addition chains

Typical recursive formulas:

$2P = P+P$. $3P = 2P+P$.

$4P = 2P+2P$. $5P = 3P+2P$.

$6P = 3P+3P$. $7P = 5P+2P$.

$2nP = 7P+(n-7)P$ if $4 \le n < 8$.

$(2n+1)P = 2nP+P$ if $4 \le n < 8$.

$(4n+1)P = 4nP+P$ if $4 \le n < 8$.

$(4n+3)P = 4nP+3P$ if $4 \le n < 8$.

$2nP = nP+nP$ if $8 \le n$.

$(8n+1)P = 8nP+P$ if $4 \le n$.

$(8n+3)P = 8nP+3P$ if $4 \le n$.

$(8n+5)P = 8nP+5P$ if $4 \le n$.

$(8n+7)P = 8nP+7P$ if $4 \le n$.

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. ≈ 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 $\mathbf{Z}/p$.
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
$\lambda = (3(x/z^2)^2 - 3)/2(y/z^3)$
$\quad = \alpha/2yz$ with $\alpha = 3x^2 - 3z^4$;
$x'' = \lambda^2 - 2(x/z^2)$
$\quad = (\alpha^2 - 8xy^2)/(2yz)^2$;
$y'' = \lambda((x/z^2) - x'') - (y/z^3)$
$\quad = (12xy^2\alpha - \alpha^3 - 8y^4)/(2yz)^3$.

$2(x/z^2, y/z^3) = (x_2/z_2^2, y_2/z_2^3)$
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, α^2, $\alpha(\cdots)$.
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 $3x^2 - 3z^4$ as $3(x - z^2)(x + z^2)$.

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

Note:
$3x^2 - 3z^4$ 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 $P, 3P, 5P, 7P$?

Can convert $P, 3P, 5P, 7P$ out of Jacobian coordinates with one division, several mults. Then save mults in every addition of $P, 3P, 5P, 7P$. "Mixed coordinates."

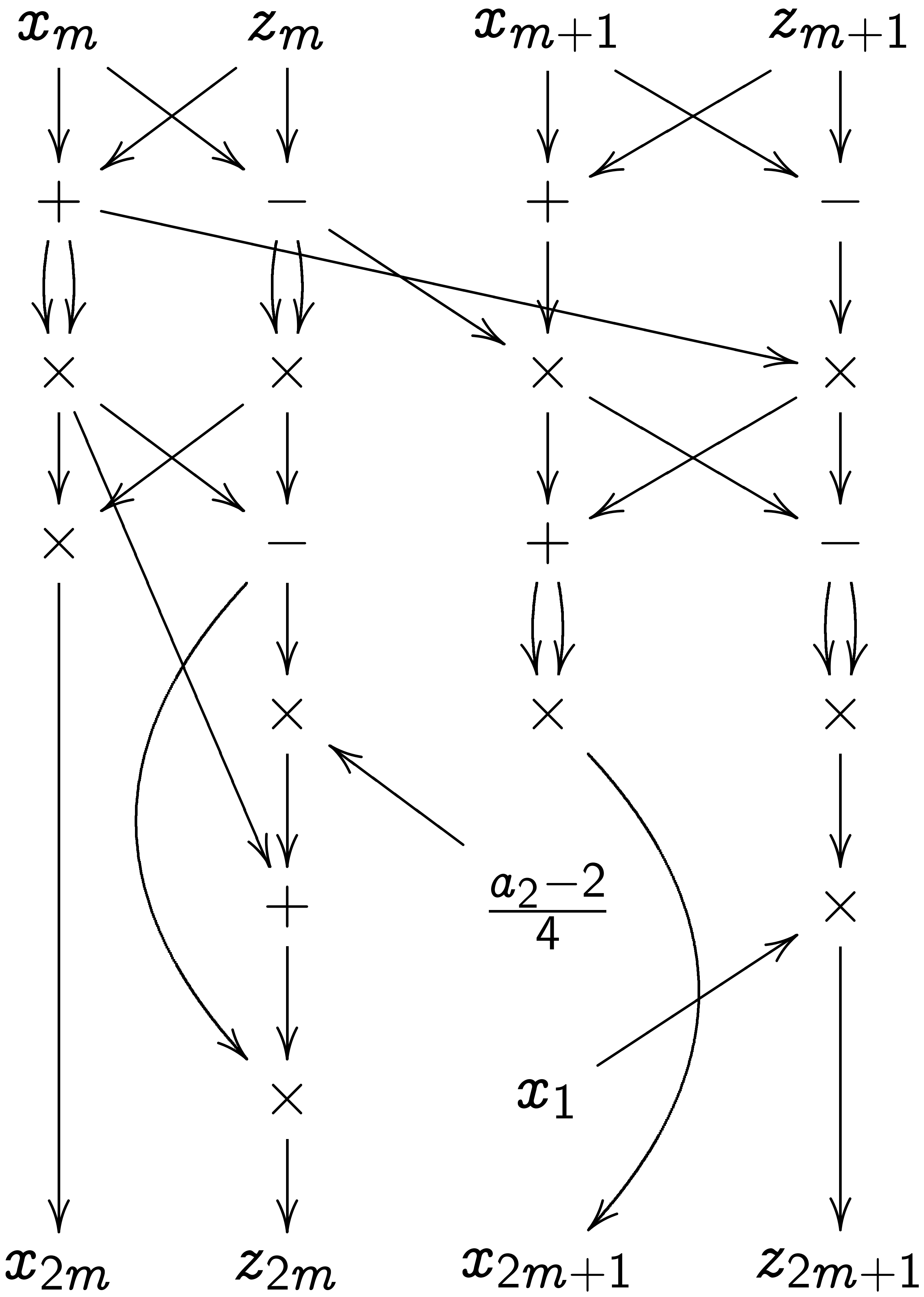
Sometimes worthwhile, depending on division speed.

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.


x_m
z_m
x_{m+1}
z_{m+1}
$+$
$-$
$+$
$-$
$\times$
$\times$
$\times$
$\times$
$\times$
$-$
$+$
$-$
$\times$
$\times$
$\times$
$+$
$\frac{a_2-2}{4}$
$\times$
$\times$
x_1
x_{2m}
z_{2m}
x_{2m+1}
z_{2m+1}

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.

What are today's speed records?

Let's focus on Pentium M.

Each Pentium M cycle does ≤ 1 floating-point operation: fp add or fp sub or fp mult.

Current scalar-multiplication software for $y^2 = x^3 + 486662x^2 + x$ over $\mathbf{Z}/(2^{255} - 19)$: 640838 Pentium M cycles. 589825 fp ops; ≈ 0.92 per cycle.

Understand cycle counts fairly well by simply counting fp ops.

Main loop: 545700 fp ops.
2140 times 255 iterations.

Reciprocal: 43821 fp ops.
$41148 = 254 \cdot 162$ for 254 squares;
$2673 = 11 \cdot 243$ for 11 more mults.

Additional work: 304 fp ops.

Inside one main-loop iteration:
$80 = 8 \cdot 10$ for 8 adds/subs;
55 for mult by 121665;
$648 = 4 \cdot 162$ for 4 squarings;
$1215 = 5 \cdot 243$ for 5 more mults;
142 for $bx[1] + (1 - b)x[0]$ etc.

An integer mod $2^{255} - 19$ is
represented in radix $2^{25.5}$
as a sum of 10 fp numbers
in specified ranges.

Add/sub: 10 fp adds/subs.
Delay reductions and carries!

Mult: poly mult using
10^2 fp mults, 9^2 fp adds;
reduce using 9 fp mults, 9 fp adds;
carry 11 times, each 4 fp adds;
overall $2 \cdot 10^2 + 4 \cdot 10 + 3$ fp ops.

Squaring: first do 9 fp doublings;
then eliminate $9^2 + 9$ fp ops;
overall $1 \cdot 10^2 + 6 \cdot 10 + 2$ fp ops.

Course advertisement

“High-speed cryptography”
at the Fields Institute, 36 hours,
starting 23 Oct, ending 17 Nov.

What are the state-of-the-art cryptographic functions for sharing secrets, expanding keys, authenticating data, signing data?
How fast are these functions in software for typical CPUs?
What’s known about security?
How were the functions chosen?

`cr.yp.to/highspeed.html`