High-speed Diffie-Hellman, part 2

D. J. Bernstein
University of Illinois at Chicago

Classic question about the Diffie-Hellman system:
How quickly can we compute nth powers mod p?

"Modular exponentiation."

Assume standard prime p;
e.g. $p = 2^{262} - 5081$.
How quickly can we compute
$g^n \bmod 2^{262} - 5081$,
given integers g, n?

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

“Elliptic-curve
scalar multiplication.”

Assume standard field
and standard 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 = $ 18958286285566608
000408668544493926
415504680968679321
075787234672564.

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 $p = 2^{255} - 19$.

Your task: Given (x, y) on curve, and given integer $n \geq 0$, compute nth 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: $+$, $\oplus$, $[n]$, etc. But this talk uses simplified notation.

Similar tasks are critical for elliptic-curve signatures.

e.g. Schnorr signatures, unfortunately patented:

Signer has secret key n, public key nB.

To sign m: choose random z, uniform in $\{0, 1, \ldots, \#\langle B\rangle - 1\}$; compute $r = \text{SHA-256}(zB, m)$; compute $s = z + rn \bmod \#\langle B\rangle$; send (m, r, s).

To verify (m, r, s): Check $r = \text{SHA-256}(sB - rnB, m)$.

Multiples via additions

Typical recursive formulas:

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

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

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

$2nP = 7P+(n-7)P$ 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" $y^2 = x^3 + a_2x^2 + x$, preferably with small $(a_2 - 2)/4$:

$n(x_1, \ldots) = (x_n/z_n, \ldots)$ where
$z_1 = 1$; $x_{2m} = (x_m^2 - z_m^2)^2$;
$z_{2m}{=}4x_mz_m(x_m^2{+}a_2x_mz_m{+}z_m^2)$;
$x_{2m+1}{=}4(x_mx_{m+1}{-}z_mz_{m+1})^2$;
$z_{2m+1}{=}4(x_mz_{m+1}{-}z_mx_{m+1})^2x_1$.

Can also figure out y, or use cryptographic protocols that ignore y.

$$\begin{tikzcd}
x_m \arrow[d] \arrow[dr] & z_m \arrow[d] \arrow[dl] & x_{m+1} \arrow[d] \arrow[dr] & z_{m+1} \arrow[d] \arrow[dl] \\
+ \arrow[d, shift left] \arrow[d, shift right] \arrow[drrr] & - \arrow[d, shift left] \arrow[d, shift right] \arrow[dr] & + \arrow[d] & - \arrow[d] \\
\times \arrow[d] \arrow[dr] \arrow[dddr, bend right] & \times \arrow[d] \arrow[dl] & \times \arrow[d] \arrow[dr] & \times \arrow[d] \arrow[dl] \\
\times \arrow[dddd] & - \arrow[d] \arrow[ddd, bend left] & + \arrow[d, shift left] \arrow[d, shift right] & - \arrow[d, shift left] \arrow[d, shift right] \\
 & \cdot\frac{a_2-2}{4} \arrow[d] & \times \arrow[ddd] & \times \arrow[d] \\
 & + \arrow[d] & & \cdot x_1 \arrow[dd] \\
 & \times \arrow[d] & & \\
x_{2m} & z_{2m} & x_{2m+1} & z_{2m+1}
\end{tikzcd}$$

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.

Choosing curves

Traditional algorithm design:
Have a function f.
Want fastest algorithm
that computes f.

Cryptographic algorithm design:
Have gigantic collection of
apparently-safe functions f.
Want fastest algorithm
that computes some f.

Elliptic-curve Diffie-Hellman could use any elliptic curve E over any finite field $\mathbf{F}_q$.

Some choices of $E, \mathbf{F}_q$ are better than others.
Higher speed: easier to compute nth multiples in $E(\mathbf{F}_q)$.
Higher security: harder to find n given an nth multiple, i.e., to solve ECDLP.
Lower bandwidth. Etc.

How do we choose $E, \mathbf{F}_q$?
Which curves are best?

Occasionally an application has different criteria for $E, \mathbf{F}_q$.
e.g. Some cryptographic protocols use “pairings” and need specific “embedding degrees.”

For simplicity I’ll focus on traditional protocols:
Diffie-Hellman, ECDSA, etc.

Can also consider, e.g., genus-2 hyperelliptic curves.
2006.09: New speed records, faster than elliptic curves.

For simplicity I’ll focus on the elliptic-curve case.

Field size?

The group $E(\mathbf{F}_q)$ has
$\approx q$ elements.

"Generic" algorithms such as
"Pollard's rho method"
solve ECDLP using
$\approx q^{1/2}$ simple operations.
Highly parallelizable.

e.g. $\approx 2^{40}$ simple operations
to solve ECDLP if $q \approx 2^{80}$.
Reject q: too small.

$q \approx 2^{256}$ is clearly safe against these ECDLP algorithms. $\approx 2^{128}$ simple operations would need massive advances in computer technology.

These algorithms can finish early, but almost never do: e.g., chance $\approx 2^{-56}$ of finishing after 2^{100} simple operations. No serious risk.

Popular today: $q \approx 2^{160}$. Somewhat faster arithmetic. I don't recommend this; I can imagine 2^{80} simple operations.

Field degree?

Field size q is a power of field characteristic p. Many possibilities for field degree $(\lg q)/(\lg p)$.

e.g. $q = 2^{255} - 19$; prime;
$p = 2^{255} - 19$; degree 1.

e.g. $q = (2^{61} - 1)^5$;
$p = 2^{61} - 1$; degree 5.

e.g. $q = 2^{255}$;
$p = 2$; degree 255.

What's the best degree?

Degree > 1 has a possible security problem: “Weil descent.”

e.g. Degree divisible by 4 allows ECDLP to be solved with only about $q^{0.375}$ simple operations. Need to increase q, outweighing all known benefits.

Other degrees are at risk too. Exactly which curves are broken by Weil descent? Very complicated answer; active research area.

Maybe we can be comfortable with degree > 1 despite Weil descent.

Standard argument for using
small characteristic, large degree:

Arithmetic on polynomials mod 2
is just like integer arithmetic
but faster: skip the carries.

Also have fast squarings.
Use fast curve endomorphisms.

Fewer bit operations
for scalar multiplication
in characteristic 2,
compared to large characteristic.
Speculation: ≈ 4 times fewer?

Counterargument:

Typical CPU includes circuits
for integer multiplication,
not for poly mult mod 2.

Large char is slower in hardware
than char 2, but
char 2 is slower in software
than large char.
Hard for char-2 standards
to survive.

For simplicity I'll assume
that the counterargument wins:
we won't use char 2.

Medium char? Similar problems.

e.g. $q = (2^{31} - 1)^8$, $p = 2^{31} - 1$, degree 8, polys with coefficients in $\{0, 1, \ldots, 2^{31} - 2\}$:

Coefficient products fit comfortably into 64 bits.
Also have fast inversion.

But hard to take advantage of 128-bit products; and hard to fit into 53-bit floating-point products.
Big speed loss on many CPUs, outweighing all known benefits.

Prime shape?

Assume prime field from now on; $\mathbf{F}_q = \mathbf{F}_p = \mathbf{Z}/p$.

How to choose prime p? Three common choices in literature.

"Binomial":
e.g., $2^{255} - 19$.

"Radix 2^{32}":
e.g., NIST prime $2^{224} - 2^{96} + 1$.

"Random":
no special shape for p.

Classic Diffie-Hellman had an argument for random primes.

Here's the argument:
Best attack so far, namely modern "NFS" index calculus, is faster for special primes, requiring larger primes, outweighing any possible speedup.

Argument disappears for elliptic curves over prime fields. Attacker doesn't seem to benefit from special primes; don't have anything like NFS.

So choose prime
very close to power of 2,
saving time in field operations.

Binomial primes allow very fast
reduction, as we've seen.

Radix-2^{32} primes also allow
very fast reduction *if*
integer arithmetic uses radix 2^{32}.
Otherwise not quite as fast.
Different CPUs want
different choices of radix,
so binomial primes are better.

Which power of 2?

Primes not far below 2^{32w} allow field elements to fit in $4w$ bytes, minimal waste.

Comfortable security, $w = 8$:
$2^{253} + 39$, $2^{253} + 51$, $2^{254} + 79$,
$2^{255} - 31$, $2^{255} - 19$, $2^{255} + 95$.
I recommend $2^{255} - 19$.

Subgroup shape?

Elliptic-curve Diffie-Hellman uses standard base point B. Bob's secret key is n; Bob's public key is nB.

Order of B in group should be a prime $\ell \approx q$. Otherwise ECDLP is accelerated by "Pohlig-Hellman algorithm."

This constrains curve choice: number of elements of $E(\mathbf{F}_q)$ must have large prime divisor ℓ.

Quickly compute $\#E(\mathbf{F}_q)$,
number of elements of $E(\mathbf{F}_q)$,
using "Schoof's algorithm."
Then can check for ℓ.

Also enforce other constraints:
$\gcd\{\#E(\mathbf{F}_q), q\} = 1$ to stop
"anomalous curve attack";
large prime divisor of
"twist order" $2q + 2 - \#E(\mathbf{F}_q)$
to stop "twist attacks";
large embedding degree
to eliminate "pairings."

Curve shape?

How to choose a_1, a_2, a_3, a_4, a_6 defining elliptic curve $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$?

See some coefficients in explicit formulas for curve operations.

e.g. Derivative $3x^2 + 2a_2x + a_4$ usually creates mult by a_2.

But formulas vary: e.g., mult by $(a_2 - 2)/4$ in Montgomery's formulas.

Save time in these formulas by specializing coefficients.

e.g. $y^2 = x^3 - 3x + a_6$.

e.g. $y^2 = x^3 + a_2x^2 + x$.

Many other interesting choices.

Warning: some specializations can force low embedding degree or otherwise create security problems. Remember to check all the security conditions.

Note on comparing curves
and comparing explicit formulas:
Count CPU cycles, not field ops!
Otherwise you make bad choices.

Reality: mult by small constant
is as expensive as several adds.

Reality: square-to-multiply ratio
is 2/3 for a typical field,
not the often-presumed 4/5.

Reality: $a^2 + b^2 + c^2$ is
faster than (a^2, b^2, c^2).

Current speed records use curve $y^2 = x^3 + a_2x^2 + x$ with small $(a_2 - 2)/4$.
Additional advantages:
easily resist timing attacks;
easily eliminate y.

$a_2 = 486662$ has near-prime curve order and twist order.

"Curve25519":
`http://cr.yp.to/ecdh.html`

How fast is this curve?

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 Curve25519:
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.