Faster rho for elliptic curves

D. J. Bernstein
University of Illinois at Chicago

"Breaking ECC2K-130",
joint work by many authors:

ECDL attack in progress against Koblitz curve over $\mathbf{F}_{2^{131}}$ using many CPUs, GPUs, FPGAs.

Vertically integrated stack of new techniques for optimizing choice of rho iteration function, computation of iteration function, underlying binary-field arithmetic. Also new improvements in rho communication volume.

"Breaking ECC2K-130",
joint work by many authors:

ECDL attack in progress against Koblitz curve over $\mathbf{F}_{2^{131}}$ using many CPUs, GPUs, FPGAs.

Vertically integrated stack of new techniques for optimizing choice of rho iteration function, computation of iteration function, underlying binary-field arithmetic. Also new improvements in rho communication volume.

But I'm actually going to talk about something else.

Previous ECDL attack:

2009.07 Bos–Kaihara–
Kleinjung–Lenstra–Montgomery
"PlayStation 3 computing
breaks 2^{60} barrier:
112-bit prime ECDLP solved".

Successful ECDL computation
for a standard curve over $\mathbf{F}_p$
where $p = (2^{128} - 3)/(11 \cdot 6949)$.

Previous ECDL attack:

2009.07 Bos–Kaihara–
Kleinjung–Lenstra–Montgomery
"PlayStation 3 computing
breaks 2^{60} barrier:
112-bit prime ECDLP solved".

Successful ECDL computation
for a standard curve over $\mathbf{F}_p$
where $p = (2^{128} - 3)/(11 \cdot 6949)$.

"We did not use
the common negation map
since it requires branching
and results in code that runs
slower in a SIMD environment."

2009.07 Bos–Kaihara–Kleinjung–Lenstra–Montgomery “On the security of 1024-bit RSA and 160-bit elliptic curve cryptography”:

Group order $q \approx p$;
“expected number of iterations” is “$\sqrt{\frac{\pi \cdot q}{2}} \approx 8.4 \cdot 10^{16}$”; “we do not use the negation map”; “456 clock cycles per iteration per SPU”; “24-bit distinguishing property” ⇒ “260 gigabytes”.

“The overall calculation can be expected to take approximately 60 PS3 years.”

2009.09 Bos–Kaihara–Montgomery "Pollard rho on the PlayStation 3":

"Our software implementation is optimized for the SPE . . . the computational overhead for [the negation map], due to the conditional branches required to check for fruitless cycles [13], results (in our implementation on this architecture) in an overall performance degradation."

"[13]" is 2000 Gallant–Lambert–Vanstone.

2010.07 Bos–Kleinjung–Lenstra "On the use of the negation map in the Pollard rho method":

"If the Pollard rho method is parallelized in SIMD fashion, it is a challenge to achieve any speedup at all. . . . Dealing with cycles entails administrative overhead and branching, which cause a non-negligible slowdown when running multiple walks in SIMD-parallel fashion. . . . [This] is a major obstacle to the negation map in SIMD environments."

2010 Bernstein–Lange–Schwabe,
first announcement today:
Our software solves
random ECDL on the same curve
(with no precomputation)
in 35.6 PS3 years on average.

For comparison:
Bos–Kaihara–Kleinjung–Lenstra–
Montgomery code
uses 65 PS3 years on average.

2010 Bernstein–Lange–Schwabe,
first announcement today:
Our software solves
random ECDL on the same curve
(with no precomputation)
in 35.6 PS3 years on average.

For comparison:
Bos–Kaihara–Kleinjung–Lenstra–Montgomery code
uses 65 PS3 years on average.
Computation used 158000 kWh
(if PS3 ran at only 300W),
wasting $>$70000 kWh,
unnecessarily generating $>$10000
kilograms of carbon dioxide.

Several levels of speedups,
starting with fast arithmetic
and continuing up through rho.

Most important speedup:
We use the negation map.

Several levels of speedups,
starting with fast arithmetic
and continuing up through rho.

Most important speedup:
We use the negation map.

Extra cost in each iteration:
extract bit of "s"
(normalized y, needed anyway);
expand bit into mask;
use mask to conditionally
replace (s, y) by $(-s, -y)$.
5.5 SPU cycles ($\approx 1.5\%$ of total).
No conditional branches.

Bos–Kleinjung–Lenstra say
that "on average more elliptic
curve group operations are
required per step of each walk.
This is unavoidable" etc.

Specifically: If the precomputed
adding-walk table has r points,
need 1 extra doubling to escape
a cycle after $\approx 2r$ additions.
And more: "cycle reduction" etc.

Bos–Kleinjung–Lenstra say
that the benefit of large r
is "wiped out by
cache inefficiencies."

There's really no problem here!

We use $r = 2048$.
$1/2r = 1/4096$; negligible.

Recall: p has 112 bits.
28 bytes for table entry (x, y).

We expand to 36 bytes
to accelerate arithmetic.
We compress to 32 bytes
by insisting on small x, y;
very fast initial computation.

Only 64KB for table.
Our Cell table-load cost: 0,
overlapping loads with arithmetic.
No "cache inefficiencies."

What about fruitless cycles?

We run 45 iterations.
We then save s;
run 2 slightly slower iterations
tracking minimum (s, x, y);
then double tracked (x, y)
if new s equals saved s.
Credits: 1999 GLV, 1999 DGM.

(Occasionally replace 2 by 12
to detect 4-cycles, 6-cycles.
Such cycles are almost
too rare to worry about,
but detecting them has a
completely negligible cost.)

Maybe fruitless cycles waste
some of the 47 iterations.
. . . but this is infrequent.
Lose $\approx 0.6\%$ of all iterations.

Tracking minimum isn't free,
but most iterations skip it!
Same for final s comparison.
Still no conditional branches.
Overall cost $\approx 1.3\%$.

Doubling occurs for only
$\approx 1/4096$ of all iterations.
We use SIMD quite lazily here;
overall cost $\approx 0.6\%$.
Can reduce this cost further.

To confirm iteration effectiveness
we have run many experiments
on $y^2 = x^3 - 3x + 9$
over the same $\mathbf{F}_p$,
using smaller-order P.
Matched DL cost predictions.

Final conclusions:
Sensible use of negation,
with or without SIMD,
has negligible impact
on cost of each iteration.
Impact on number of iterations
is almost exactly $\sqrt{2}$.
Overall benefit is
extremely close to $\sqrt{2}$.