Generic attacks and index calculus

D. J. Bernstein
University of Illinois at Chicago

The discrete-logarithm problem

Define $p = 1000003$.

Easy to prove: p is prime.

Can we find an integer
$n \in \{1, 2, 3, \ldots, p-1\}$
such that $5^n \bmod p = 262682$?

Easy to prove: $n \mapsto 5^n \bmod p$
permutes $\{1, 2, 3, \ldots, p-1\}$.
So there *exists* an n
such that $5^n \bmod p = 262682$.

Could find n by brute force.
Is there a faster way?

Typical cryptanalytic application:

Imagine standard $p = 1000003$ in the Diffie-Hellman protocol.

User chooses secret key n, publishes $5^n \bmod p = 262682$.

Can attacker quickly solve the discrete-logarithm problem? Given public key $5^n \bmod p$, quickly find secret key n?

(Warning: This is *one* way to attack the protocol. Maybe there are better ways.)

Relations to ECC:

1. Some DL techniques also apply to elliptic-curve DL problems. Use in evaluating security of an elliptic curve.

2. Some techniques don't apply. Use in evaluating advantages of elliptic curves compared to multiplication.

3. Tricky: Some techniques have extra applications to some curves. See Tanja Lange's talk on Weil descent etc.

Understanding brute force

Can compute successively
$5^1 \bmod p = 5$,
$5^2 \bmod p = 25$,
$5^3 \bmod p = 125, \ldots,$
$5^8 \bmod p = 390625$,
$5^9 \bmod p = 953122, \ldots,$
$5^{1000002} \bmod p = 1$.

At some point we'll find n
with $5^n \bmod p = 262682$.

Maximum cost of computation:
$\le p - 1$ mults by 5 mod p;
$\le p - 1$ nanoseconds on a CPU
that does 1 mult/nanosecond.

This is negligible work
for $p \approx 2^{20}$.

But users can
standardize a larger p,
making the attack slower.

Attack cost scales linearly:
$\approx 2^{50}$ mults for $p \approx 2^{50}$,
$\approx 2^{100}$ mults for $p \approx 2^{100}$, etc.

(Not exactly linearly:
cost of mults grows with p.
But this is a minor effect.)

Computation has a good chance of finishing earlier.
Chance scales linearly:
1/2 chance of 1/2 cost;
1/10 chance of 1/10 cost; etc.

"So users should choose large n."

That's pointless. We can apply "random self-reduction":
choose random r, say 726379;
compute $5^r \bmod p = 515040$;
compute $5^r 5^n \bmod p$ as
$(515040 \cdot (5^n \bmod p)) \bmod p$;
compute discrete log;
subtract $r \bmod p-1$; obtain n.

Computation can be parallelized.

One low-cost chip can run many parallel searches.
Example, 2^6 €: one chip, 2^{10} cores on the chip, each 2^{30} mults/second?
Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.
Example, 2^{30} €: 2^{24} chips, so 2^{34} cores, so 2^{64} mults/second, so 2^{89} mults/year.

Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets $5^{n_1} \bmod p$, $5^{n_2} \bmod p$, ..., $5^{n_{100}} \bmod p$: Can find *all* of $n_1, n_2, \ldots, n_{100}$ with $\le p - 1$ mults mod p.

Simplest approach: First build a sorted table containing $5^{n_1} \bmod p$, ..., $5^{n_{100}} \bmod p$. Then check table for $5^1 \bmod p$, $5^2 \bmod p$, etc.

Interesting consequence #1:
Solving all 100 DL problems
isn't much harder than
solving one DL problem.

Interesting consequence #2:
Solving *at least one*
out of 100 DL problems
is much easier than
solving one DL problem.

When did this computation
find its *first* n_i?
Typically $\approx (p-1)/100$ mults.

Can use random self-reduction
to turn a single target
into multiple targets.

Given $5^n \bmod p$:
Choose random $r_1, r_2, \ldots, r_{100}$.
Compute $5^{r_1}5^n \bmod p$,
$5^{r_2}5^n \bmod p$, etc.

Solve these 100 DL problems.
Typically $\approx (p-1)/100$ mults
to find *at least one*
$r_i + n \bmod p - 1$,
immediately revealing n.

Also spent some mults
to compute each $5^{r_i} \bmod p$:
$\approx \lg p$ mults for each i.

Faster: Choose $r_i = ir_1$
with $r_1 \approx (p-1)/100$.
Compute $5^{r_1} \bmod p$;
$5^{r_1} 5^n \bmod p$;
$5^{2r_1} 5^n \bmod p$;
$5^{3r_1} 5^n \bmod p$; etc.
Just 1 mult for each new i.

$\approx 100 + \lg p + (p-1)/100$ mults
to find n given $5^n \bmod p$.

Faster: Increase 100 to $\approx \sqrt{p}$.
Only $\approx 2\sqrt{p}$ mults
to solve one DL problem!

"Shanks baby-step-giant-step
discrete-logarithm algorithm."

Example: $p = 1000003$,
$5^n \bmod p = 262682$.

Compute $5^{1024} \bmod p = 58588$.
Then compute 1000 targets:
$5^{1024}5^n \bmod p = 966849$,
$5^{2\cdot 1024}5^n \bmod p = 579277$,
$5^{3\cdot 1024}5^n \bmod p = 579062, \ldots,$
$5^{1000\cdot 1024}5^n \bmod p = 321705$.

Build a sorted table of targets:

$2573 = 5^{430 \cdot 1024} 5^n \bmod p$,

$3371 = 5^{192 \cdot 1024} 5^n \bmod p$,

$3593 = 5^{626 \cdot 1024} 5^n \bmod p$,

$4960 = 5^{663 \cdot 1024} 5^n \bmod p$,

$5218 = 5^{376 \cdot 1024} 5^n \bmod p$, ...,

$999675 = 5^{344 \cdot 1024} 5^n \bmod p$.

Look up $5^1 \bmod p$, $5^2 \bmod p$, $5^3 \bmod p$, etc. in this table.

$5^{755} \bmod p = 966603$; find $966603 = 5^{332 \cdot 1024} 5^n \bmod p$ in the table of targets; so $755 = 332 \cdot 1024 + n \bmod p - 1$; deduce $n = 660789$.

Eliminating storage

Improved method: Define $x_0 = 1$;
$x_{i+1} = 5x_i \bmod p$ if $x_i \in 3\mathbf{Z}$;
$x_{i+1} = x_i^2 \bmod p$ if $x_i \in 2 + 3\mathbf{Z}$;
$x_{i+1} = 5^n x_i \bmod p$ otherwise.

Then $x_i = 5^{a_i n + b_i} \bmod p$
where $(a_0, b_0) = (0, 0)$ and
$(a_{i+1}, b_{i+1}) = (a_i, b_i + 1)$, or
$(a_{i+1}, b_{i+1}) = (2a_i, 2b_i)$, or
$(a_{i+1}, b_{i+1}) = (a_i + 1, b_i)$.

Search for a collision in x_i:
$x_1 = x_2$? $x_2 = x_4$? $x_3 = x_6$?
$x_4 = x_8$? $x_5 = x_{10}$? etc.
Deduce linear equation for n.

The x_i's enter a cycle, typically within $\approx \sqrt{p}$ steps.

Example: 1000003, 262682.

Modulo 1000003:

$x_1 = 5^n = 262682.$

$x_2 = 5^{2n} = 262682^2 = 626121.$

$x_3 = 5^{2n+1} = 5 \cdot 626121 = 130596.$

$x_4 = 5^{2n+2} = 5 \cdot 130596 = 652980.$

$x_5 = 5^{2n+3} = 5 \cdot 652980 = 264891.$

$x_6 = 5^{2n+4} = 5 \cdot 264891 = 324452.$

$x_7 = 5^{4n+8} = 324452^2 = 784500.$

$x_8 = 5^{4n+9} = 5 \cdot 784500 = 922491.$

etc.

$x_{1785} = 5^{249847n+759123} = 555013.$

$x_{3570} = 5^{388795n+632781} = 555013.$

(Cycle length is 357.)

Conclude that

$249847n + 759123 \equiv$

$388795n + 632781 \pmod{p-1}$,

so $n \equiv 160788 \pmod{(p-1)/6}$.

Only 6 possible n's.

Try each of them.

Find that $5^n \bmod p = 262682$

for $n = 160788 + 3(p-1)/6$, i.e.,

for $n = 660789$.

This is “Pollard’s rho method.”
Optimized: $\approx \sqrt{p}$ mults.
Another method, similar speed:
“Pollard’s kangaroo method.”

Can parallelize both methods.
“van Oorschot/Wiener parallel
DL using distinguished points.”

Bottom line: With c mults,
distributed across many cores,
have chance $\approx c^2/p$
of finding n from $5^n \bmod p$.

With 2^{90} mults (a few years?),
have chance $\approx 2^{180}/p$.
Negligible if, e.g., $p \approx 2^{256}$.

Factors of the group order

Assume 5 has order ab.

Given x, a power of 5:

5^a has order b, and
x^a is a power of 5^a.
Compute $\ell = \log_{5^a} x^a$.

5^b has order a, and
$x/5^\ell$ is a power of 5^b.
Compute $m = \log_{5^b}(x/5^\ell)$.

Then $x = 5^{\ell+mb}$.

This “Pohlig-Hellman method” converts an order-ab DL into an order-a DL, an order-b DL, and a few exponentiations.

e.g. $p = 1000003$, $x = 262682$:
$p - 1 = 6b$ where $b = 166667$.
Compute $\log_{5^6}(x^6) = 160788$.
Compute $x/5^{160788} = 1000002$.
Compute $\log_{5^b} 1000002 = 3$.
Then $x = 5^{160788+3b} = 5^{660789}$.

Use rho: $\approx \sqrt{a} + \sqrt{b}$ mults.
Better if ab factors further: apply Pohlig-Hellman recursively.

All of the techniques so far
apply to elliptic curves.

An elliptic curve over $\mathbf{F}_q$
has $\approx q+1$ points
so can compute ECDL using
$\approx \sqrt{q}$ elliptic-curve adds.
Need quite large q.

If largest prime divisor
of number of points
is much smaller than q
then Pohlig-Hellman method
computes ECDL more quickly.
Need larger q;
or change choice of curve.

Index calculus

Have generated many group elements $5^{an+b} \bmod p$. Deduced equations for n from random collisions.

Index calculus obtains discrete-logarithm equations in a different way.

Example for $p = 1000003$: Can completely factor $-3/(p-3)$ as $-3^1/2^6 5^6$ in $\mathbf{Q}$ so $-3^1 \equiv 2^6 5^6 \pmod{p}$ so $\log_5(-1) + \log_5 3 \equiv 6 \log_5 2 + 6 \log_5 5 \pmod{p-1}$.

Can completely factor $62/(p+62)$
as $2^1 31^1/3^1 5^1 11^2 19^1 29^1$
so $\log_5 2 + \log_5 31 \equiv$
$\log_5 3 + \log_5 5 + 2\log_5 11 +$
$\log_5 19 + \log_5 29 \pmod{p-1}$.

Try to completely factor
$1/(p+1)$, $2/(p+2)$, etc.
Find factorization of $a/(p+a)$
as product of powers of -1,
$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31$
for each of the following a's:
-5100, -4675, -3128,
-403, -368, -147, -3,
62, 957, 2912, 3857, 6877.

Each complete factorization produces a log equation.

Now have 12 linear equations for $\log_5 2, \log_5 3, \ldots, \log_5 31$.
Free equations: $\log_5 5 = 1$, $\log_5(-1) = (p-1)/2$.

By linear algebra compute $\log_5 2, \log_5 3, \ldots, \log_5 31$.

(If this hadn't been enough, could have searched more a's.)

By similar technique obtain discrete log of any target.

For $p \to \infty$, index calculus scales surprisingly well: cost p^ϵ where $\epsilon \to 0$.

Compare to rho: $\approx p^{1/2}$.

Specifically: searching $a \in \{1, 2, \ldots, y^2\}$, with $\lg y \in O(\sqrt{\lg p \lg \lg p})$, finds y complete factorizations into primes $\le y$, and computes discrete logs.

(Assuming standard conjectures. Have extensive evidence.)

Latest index-calculus variants use the "number-field sieve" and the "function-field sieve."

To compute discrete logs in $\mathbf{F}_q$:
$\lg \text{cost} \in$
$O((\lg q)^{1/3}(\lg \lg q)^{2/3})$.

For security:
$q \approx 2^{256}$ to stop rho;
$q \approx 2^{2048}$ to stop NFS.

We don't know any index-calculus methods for ECDL!
. . . except for some curves.