ECDLP course

ECC2K-130

Daniel J. Bernstein
University of Illinois at Chicago

Tanja Lange
Technische Universiteit Eindhoven

The target: ECC2K-130

1997: Certicom announces several elliptic-curve challenges.

"The Challenge is to compute the ECC private keys from the given list of ECC public keys and associated system parameters. This is the type of problem facing an adversary who wishes to completely defeat an elliptic curve cryptosystem."

Goals: help users select key sizes; compare random and Koblitz; compare $\mathbf{F}_{2^m}$ and $\mathbf{F}_p$; etc.

1997: ECCp-79 broken by Baisley and Harley.

1997: ECC2-79 broken by Harley et al.

1998: ECCp-89, ECC2-89 broken by Harley et al.

1998: ECCp-97 broken by Harley et al. (1288 computers).

1998: ECC2K-95 broken by Harley et al. (200 computers).

1999: ECC2-97 broken by Harley et al. (740 computers).

2000: ECC2K-108 broken by Harley et al. (9500 computers).

Certicom: "The 109-bit Level I challenges are feasible using a very large network of computers. The 131-bit Level I challenges are expected to be infeasible against realistic software and hardware attacks, unless of course, a new algorithm for the ECDLP is discovered."

2002: ECCp-109 broken by Monico et al. (10000 computers).

2004: ECC2-109 broken by Monico et al. (2600 computers).

Next challenge: ECC2K-130.

The attacker: ECRYPT

European Union has funded ECRYPT I network (2004–2008), ECRYPT II network (2008–2012).

ECRYPT II: KU Leuven; ENS; EPFL; RU Bochum; RHUL; TU Eindhoven; TU Graz; U Bristol; U Salerno; France Télécom; IBM Research; 22 adjoint members.

Work is handled by "virtual labs":

- SymLab: secret-key crypto;
- MAYA: public-key crypto;
- VAMPIRE: implementations.

Working groups in VAMPIRE:

- VAM1: "Efficient Implementation of Security Systems".
- VAM2: "Physical Security".

2009.02: VAMPIRE (VAM1)
sets its sights on ECC2K-130.
Exactly how difficult
is breaking ECC2K-130?
Also ECC2-131 etc.

Sensible topic for implementors.
Optimizing ECC attacks
isn't far from optimizing ECC.

With our latest implementations,
ECC2K-130 is breakable
in two years on average by

- 1595 Phenom II x4 955 CPUs,

With our latest implementations,
ECC2K-130 is breakable
in two years on average by

- 1595 Phenom II x4 955 CPUs,
- or 1231 Playstation 3s,

With our latest implementations,
ECC2K-130 is breakable
in two years on average by

- 1595 Phenom II x4 955 CPUs,
- or 1231 Playstation 3s,
- or 631 GTX 295 cards,

With our latest implementations,
ECC2K-130 is breakable
in two years on average by

- 1595 Phenom II x4 955 CPUs,
- or 1231 Playstation 3s,
- or 631 GTX 295 cards,
- or 308 XC3S5000 FPGAs,

With our latest implementations,
ECC2K-130 is breakable
in two years on average by

- 1595 Phenom II x4 955 CPUs,
- or 1231 Playstation 3s,
- or 631 GTX 295 cards,
- or 308 XC3S5000 FPGAs,
- or any combination thereof.

With our latest implementations,
ECC2K-130 is breakable
in two years on average by

- 1595 Phenom II x4 955 CPUs,
- or 1231 Playstation 3s,
- or 631 GTX 295 cards,
- or 308 XC3S5000 FPGAs,
- or any combination thereof.

This is a computation that
Certicom called “infeasible”?

With our latest implementations,
ECC2K-130 is breakable
in two years on average by

- 1595 Phenom II x4 955 CPUs,
- or 1231 Playstation 3s,
- or 631 GTX 295 cards,
- or 308 XC3S5000 FPGAs,
- or any combination thereof.

This is a computation that
Certicom called “infeasible”?

Certicom has now backpedaled,
saying that ECC2K-130
“may be within reach”.

The target: ECC2K-130

The Koblitz curve
$y^2 + xy = x^3 + 1$
over $\mathbf{F}_{2^{131}}$ has 4ℓ points,
where ℓ is prime.
Field representation uses
irreducible polynomial
$f = z^{131} + z^{13} + z^2 + z + 1$.

Certicom generated their
challenge points as two random
points in order-ℓ subgroup by
taking two random points on the
curve and multiplying them by 4.

This produced the following points P, Q:

```
x(P) = 05 1C99BFA6 F18DE467 C80C23B9 8C7994AA
y(P) = 04 2EA2D112 ECEC71FC F7E000D7 EFC978BD
x(Q) = 06 C997F3E7 F2C66A4A 5D2FDA13 756A37B1
y(Q) = 04 A38D1182 9D32D347 BD0C0F58 4D546E9A
```

(unique encoding of $\mathbf{F}_{2^{131}}$ in hex).

The challenge:

Find an integer

$k \in \{0, 1, \ldots, \ell - 1\}$

such that $[k]P = Q$.

Bigger picture:

128-bit curves have been proposed for real (RFID, TinyTate).

Equivalence classes for Koblitz curves

P and $-P$ have same x-coordinate.

Search for x-coordinate collision.

Search space is only $\ell/2$; this gives factor $\sqrt{2}$ speedup ...

provided that $f(P_i) = f(-P_i)$.

Equivalence classes for Koblitz curves

P and $-P$ have same x-coordinate.
Search for x-coordinate collision.
Search space is only $\ell/2$; this
gives factor $\sqrt{2}$ speedup ...
provided that $f(P_i) = f(-P_i)$.

More savings: P and $\sigma^i(P)$ have
$x(\sigma^j(P)) = x(P)^{2^j}$.

Consider equivalence classes under
Frobenius and $\pm$; gain
additional factor $\sqrt{n} = \sqrt{131}$.

Need to ensure that the iteration
function satisfies
$f(P_i) = f(\pm\sigma^j(P_i))$ for any j.

Could again define adding walk starting from $|P_i|$.

Redefine $|P_i|$ as canonical representative of class containing P_i: e.g., lexicographic minimum of P_i, $-P_i$, $\sigma(P_i)$, etc.

Iterations now involve many squarings, but squarings are not so expensive in characteristic 2.

Iteration function for Koblitz curves

Normal basis of finite field

$\mathbf{F}_{2^n}$ has elements
$\{\zeta, \zeta^2, \zeta^{2^2}, \zeta^{2^3}, \ldots, \zeta^{2^{n-1}}\}$.

Representation for x and x^2
$\sum_{i=0}^{n-1} x_i \zeta^{2^i} = (x_0, x_1, x_2, \ldots, x_{n-1})$
$\sum_{i=1}^{n} x_i \zeta^{2^i} = (x_{n-1}, x_0, \ldots, x_{n-2})$
using $(\zeta^{2^{n-1}})^2 = \zeta^{2^n} = \zeta$.

Harley and Gallant-Lambert-Vanstone observe that in normal basis, $x(P)$ and $x(P)^{2^j}$ have same Hamming weight

$$\mathrm{HW}(x(P)) = \sum_{i=0}^{n-1} x_i$$

(addition over $\mathbf{Z}$).

Suggestion:

$$P_{i+1} = P_i + \sigma^j(P_i),$$

as iteration function.

Choice of j depends on HW($x(P)$).

This ensures that the walk is well defined on classes since

$$\begin{aligned} f(\pm\sigma^m(P_i)) = \\ \pm\sigma^m(P_i) + \sigma^j(\pm\sigma^m(P_i)) = \\ \pm(\sigma^m(P_i) + \sigma^m(\sigma^j(P_i))) = \\ \pm\sigma^m(P_i + \sigma^j(P_i)) = \\ \pm\sigma^m(P_{i+1}). \end{aligned}$$

GLV suggest using
$j = \text{hash}(\text{HW}(x(P)))$,
where the hash function
maps to $[1, n]$.

Harley uses a smaller set of
exponents; for his attack on
ECC2K-108 he takes
$j \in \{1, 2, 4, 5, 6, 7, 8\}$;
computed as
$j = (\text{HW}(x(P)) \bmod 7) + 2$
and replacing 3 by 1.

Our choice of iteration function

Restricting size of j matters – squarings are cheap but:

- in bitslicing need to compute all powers (no branches allowed);
- code size matters (in particular for Cell CPU);
- logic costs area for FPGA;
- having a large set doesn't actually gain much randomness.

Analysis of the loss in randomness similar to Wednesday's.

Having few coefficients lets us exclude short fruitless cycles.

To do so, compute
the shortest vector in the lattice
$\left\{v : \prod_j (1+\sigma^j)^{v_j} = 1\right\}$.
Usually the shortest vector has negative coefficients (which cannot happen with the iteration); shortest vector with positive coefficients is somewhat longer.

For implementation it is better to have a continuous interval of exponents, so shift the interval if shortest vector is short.

Our iteration function:
$P_{i+1} = P_i + \sigma^j(P_i)$ where
$j = (\mathrm{HW}(x(P))/2 \bmod 8) + 3$,
so $j \in \{3, 4, 5, 6, 7, 8, 9, 10\}$.
Shortest combination of these powers is long.
Note that $\mathrm{HW}(x(P))$ is even.

Iteration consists of

- computing the Hamming weight $\mathrm{HW}(x(P))$ of the normal-basis representation of $x(P)$;
- checking for distinguished points (is $\mathrm{HW}(x(P)) \leq 34$?);
- computing j and $P + \sigma^j(P)$.

Analysis of our iteration function

For a perfectly random walk
$\approx \sqrt{\pi\ell/2}$ iterations
are expected on average.
Have $\ell \approx 2^{131}/4$ for ECC2K-130.

A perfectly random walk
on classes under $\pm$ and Frobenius
would reduce number of iterations
by $\sqrt{2 \cdot 131}$.

Analysis of our iteration function

For a perfectly random walk
$\approx \sqrt{\pi \ell / 2}$ iterations
are expected on average.
Have $\ell \approx 2^{131}/4$ for ECC2K-130.

A perfectly random walk
on classes under $\pm$ and Frobenius
would reduce number of iterations
by $\sqrt{2 \cdot 131}$.

Loss of randomness
from having only 8 choices of j.
Further loss from non-randomness
of Hamming weights:

Hamming weights around 66
are much more likely than at the
edges; effect still noticeable
after reduction to 8 choices.

Hamming weights around 66
are much more likely than at the
edges; effect still noticeable
after reduction to 8 choices.

Our $\sqrt{1 - \sum_i p_i^2}$ heuristic says
that the total loss is 6.9993%.
(Higher-order anti-collision
analysis: actually above 7%.)
This loss is justified by
the very fast iteration function.

Hamming weights around 66
are much more likely than at the
edges; effect still noticeable
after reduction to 8 choices.

Our $\sqrt{1 - \sum_i p_i^2}$ heuristic says
that the total loss is 6.9993%.
(Higher-order anti-collision
analysis: actually above 7%.)
This loss is justified by
the very fast iteration function.

Average number of iterations for
our attack against ECC2K-130:
$\sqrt{\pi \ell / (2 \cdot 2 \cdot 131)} \cdot 1.069993$
$\approx 2^{60.9}$.

Some highlights:

Detailed analysis of randomness of iteration function.

Could increase randomness of the walk but then iteration function gets slower. Optimized for time per iteration $\times$ # iterations.

Do not remember multiset of j's; instead recompute this from seed when collision is found (cheaper, less storage).

Comparative study of normal basis and polynomial basis representation;
new: optimal polynomial bases.

Cost of iteration function

1 normal-basis Hamming-weight computation;
1 application of σ^j for some $j \in \{3, 4, \ldots, 10\}$: $\leq 20\mathbf{S}$ if computed as a series of squarings;
1 elliptic-curve addition:
$1\mathbf{I}+2\mathbf{M}+1\mathbf{S}+7\mathbf{a}$
in affine coordinates.

With Montgomery inversion, each iteration costs
$\leq (1/N)(\mathbf{I}-3\mathbf{M})+5\mathbf{M}+21\mathbf{S}+7\mathbf{a}$
plus a Hamming-weight computation in normal basis.

Bit operations

We can compute an iteration using a straight-line (branchless) sequence of $70467 + 70263/N$ two-input bit operations.
e.g. 71880 bit operations/iteration for $N = 51$.

Bit operations:
"AND" and "XOR"; i.e., multiplication and addition in $\mathbf{F}_2$.

Bit operations

We can compute an iteration
using a straight-line (branchless)
sequence of $70467 + 70263/N$
two-input bit operations.
e.g. 71880 bit operations/iteration
for $N = 51$.

Bit operations:
"AND" and "XOR"; i.e.,
multiplication and addition in $\mathbf{F}_2$.

Compare to 34061 bit operations
(131^2 ANDs + 130^2 XORs)
for one schoolbook multiplication
of two 131-bit polynomials.

Details on the multiplication

Define $M(n)$ as minimum # bit operations for multiplying n-bit polys.

e.g. $M(131) \leq 34061$ from schoolbook multiplication: 131^2 ANDs + 130^2 XORs.

Details on the multiplication

Define $M(n)$ as minimum
bit operations for
multiplying n-bit polys.

e.g. $M(131) \leq 34061$ from
schoolbook multiplication:
131^2 ANDs + 130^2 XORs.

Much lower costs are known
from Karatsuba, Toom, etc.

Current record (CRYPTO 2009):
$M(131) \leq 11961$.

“Your metric is too simple!
Hardware has area-time tradeoffs!
Software does not work on bits!”

"Your metric is too simple!
Hardware has area-time tradeoffs!
Software does not work on bits!"

Response: Optimizing
bit operations is very close to
optimizing the *throughput*
of unrolled, pipelined hardware.
See, e.g., ECC2K-130 FPGA
paper at FPL 2010.

"Your metric is too simple!
Hardware has area-time tradeoffs!
Software does not work on bits!"

Response: Optimizing
bit operations is very close to
optimizing the *throughput*
of unrolled, pipelined hardware.
See, e.g., ECC2K-130 FPGA
paper at FPL 2010.

Also very close to optimizing
the speed of software using
vectorized bit operations.
More on this tomorrow.

All of the implementations of the ECC2K-130 attack started with the standard pentanomial basis $1, z, z^2, \ldots, z^{130}$ of $\mathbf{F}_{2^{131}} = \mathbf{F}_2[z]/(z^{131} + z^{13} + z^2 + z + 1)$.

Cost $M(131) + 455$
for multiplication.

Cost 203 for squaring.

Our final attack iteration has
5 mults, 21 squarings,
1 normal-basis Hamming weight.

Question at start of project:
Work entirely in normal basis?
Critical issue: mult speed.

Type-I normal basis of $\mathbf{F}_{2^n}$
is a permutation of
$\zeta, \zeta^2, \ldots, \zeta^n$
in $\mathbf{F}_2[\zeta]/(\zeta^n + \cdots + \zeta + 1)$.

Cost $M(n)$ to multiply,
obtaining coefficients of
$\zeta^2, \zeta^3, \ldots, \zeta^{2n}$.

Cost $2n - 2$ to reduce
$\zeta^2, \zeta^3, \ldots, \zeta^{2n}$
to $\zeta, \zeta^2, \ldots, \zeta^n$.

Alternative (1989 Itoh–Tsujii),
slightly faster when n is large:
redundant $1, \zeta, \ldots, \zeta^n$;
cost $M(n+1) + n$.

But $\mathbf{F}_{2^{131}}$ doesn't have
a type-I normal basis.

$\mathbf{F}_{2^{131}}$ has a type-II
normal basis $\zeta + \zeta^{-1}$,
$\zeta^2 + \zeta^{-2}$, $\zeta^4 + \zeta^{-4}$,
$\ldots$, $\zeta^{2^{130}} + \zeta^{-2^{130}}$ where
ζ is a primitive 263rd root of 1.

1995 Gao–von zur Gathen–
Panario: Can multiply on
type-II normal basis of $\mathbf{F}_{2^n}$
by multiplying in
$\mathbf{F}_2[\zeta]/(\zeta^{2n} + \cdots + 1)$.
Cost $> 2M(n)$.

2001 Bolotov–Gashkov:

Can quickly convert
from type-II normal basis
$c, c^2, c^4, \ldots, c^{2^{n-1}}$
to "standard basis"
$1, c, c^2, \ldots, c^{n-1}$
where $c = \zeta + \zeta^{-1}$.

Cost $\leq (n/2) \lg n + 3n$.
e.g. ≤ 853 for $n = 131$.

Same cost for inverse.

(Analysis is too pessimistic;
actual cost is lower.)

Bolotov–Gashkov multiply
in this "standard basis"
with a poly mult, cost $M(n)$,
and a reduction modulo
the minimal polynomial of c.

e.g. $c^{131} + c^{130} + c^{128} + c^{124} + c^{123} + c^{122} + c^{120} + c^{115} + c^{114} + c^{112} + c^{99} + c^{98} + c^{96} + c^{67} + c^{66} + c^{64} + c^3 + c^2 + 1 = 0$ for $n = 131$.

Bolotov–Gashkov reduction
uses sparsity; cost ≤ 2340.
Overall cost $\leq M(131) + 4899$
for type-II normal-basis mult.
Still too slow to be useful.

2007 Shokrollahi
(first published in Ph.D. thesis,
then in WAIFI 2007 paper
by von zur Gathen, Shokrollahi,
and Shokrollahi):

Convert from type-II normal basis
to redundant $1, c, c^2, \ldots, c^n$.

Multiply polynomials, producing
redundant $1, c, c^2, \ldots, c^{2n}$.

Convert to redundant
$1, \zeta + \zeta^{-1}, \zeta^2 + \zeta^{-2},$
$\zeta^3 + \zeta^{-3}, \ldots\ \zeta^{2n} + \zeta^{-2n}$.

Use $\zeta^{2n+1} = 1$
to eliminate redundancy.

For $n = 131$:

Shokrollahi's analysis says
$\leq M(132) + 3462$.

Our analysis of Shokrollahi's
algorithm says $M(132) + 1559$.

Easy speedup: $M(131) + 1559$.

5 mults, other iteration overhead:
$5M(131) + 12249$.

Compare to pentanomial basis:
$5M(131) + 14372$.

Do even better by mixing
permuted type-II optimal normal
basis $\zeta + \zeta^{-1}$, $\zeta^2 + \zeta^{-2}$,
$\zeta^3 + \zeta^{-3}$, ..., $\zeta^n + \zeta^{-n}$
with "type-II optimal polynomial
basis" $\zeta + \zeta^{-1}$, $(\zeta + \zeta^{-1})^2$,
$(\zeta + \zeta^{-1})^3$, ..., $(\zeta + \zeta^{-1})^n$.

Use normal basis for outputs
that will be provided to squaring,
poly basis for outputs
that will be provided to mult.

Use a new reduction algorithm
for poly-basis output.
See paper for details.

Current iteration cost:
$5M(131) + 10305$.

Practical impact:
All of the ECC2K-130
implementations have upgraded
from pentanomial basis
to type-II bases, saving time.

Addendum

In general, an efficiently computable endomorphism ϕ of order r speeds up Pollard rho method by factor $\sqrt{r}$.

Can define walk on classes by inspecting all $2r$ points $\pm P, \pm\phi(P), \ldots, \pm\phi^{r-1}(P)$ to choose unique representative for class and then doing an adding walk.

So $y^2 = x^3 + ax$ and $y^2 = x^3 + b$ come at a security loss of $\sqrt{2}$.

GLS curves also have endomorphisms of order 2.
As in the case of GLV curves, loss of factor $\sqrt{2}$ is fully made up for by the faster arithmetic.

Security of DLP might not be sufficient for your protocol; some are based on hardness of static Diffie-Hellman problem.

Recent observation (Granger 2010): Oracle assisted DHP is easier on GLS curves than on curves over prime fields.