Hyper-and-elliptic-curve cryptography

(which is not the same as: hyperelliptic-curve cryptography and elliptic-curve cryptography)

Daniel J. Bernstein
University of Illinois at Chicago & Technische Universiteit Eindhoven

Joint work with:
Tanja Lange
Technische Universiteit Eindhoven

But first some context. . .
ECC security vs. ECDL security

Crypto view of ECDL problem:
Fix finite $k, E/k, P \in E(k)$.

Secret key: random $a \in \mathbb{Z}/\#\mathbb{Z}P$.

Public key: $aP$.

The ECDL problem: compute secret key from public key.

ECDL solution $\Rightarrow$ ECC attack.
ECC security vs. ECDL security

Crypto view of ECDL problem:
Fix finite $k$, $E/k$, $P \in E(k)$.

**Secret key:** random $a \in \mathbb{Z}/\#\mathbb{Z}P$.

**Public key:** $aP$.

The ECDL problem: compute secret key from public key.

ECDL solution $\Rightarrow$ ECC attack.

ECC attack $\Rightarrow$ ECDL solution?
Not necessarily!

Let’s look at some examples.
Example 1: **Kummer-line ECDH** (1985 Miller). Bob has secret $b$; receives $X(A)$ from Alice; uses easy formulas to compute $X(bA)$; encrypts using $X(bA)$. 
Example 1: Kummer-line ECDH (1985 Miller). Bob has secret $b$; receives $X(A)$ from Alice; uses easy formulas to compute $X(bA)$; encrypts using $X(bA)$.

**Twist attack:** choose $A \in E(k)$, small $\#ZA$; learn $b \mod \#ZA$. 
Example 1: **Kummer-line ECDH** (1985 Miller). Bob has secret $b$; receives $X(A)$ from Alice; uses easy formulas to compute $X(bA)$; encrypts using $X(bA)$.

**Twist attack:** choose $A \in E(\bar{k})$, small $\#ZA$; learn $b \mod \#ZA$.

Typically Bob checks $X(A) \in k$ but doesn’t check $A \in E(k)$.

Formulas also work for $A \in E'(k)$ for appropriate twist $E'$ of $E$.

Typically $\#E(k)$ is large prime but $\#E'(k)$ has small factors.
Example 2: Censor scans network, terminates users who send many elements of $X(E(k))$.

2004 Möller: Fix twist-secure $E$. Send $X(aP)$ or $X(a'P')$.

Annoying: e.g., consider ECDH.

Same basic issue arises in random-number generation (see, e.g., 2006 Gjøsteen and 2006 Schoenmakers–Sidorenko), password-authenticated key exchange (e.g., 2001 Boyd–Montague–Nguyen, broken 2013), ID-based encryption, etc.
2013 Bernstein–Hamburg–Krasnova–Lange

“Elligator: Elliptic-curve points indistinguishable from uniform random strings”:

Replace $X$ with fast bijection between large $S \subseteq E(k)$ and, e.g., interval $\{0, 1, \ldots, 2^b - 1\}$.

Alice keeps generating $a$ until $aP \in S$.

Two examples given in paper, both with $\#S \approx 0.5\#E(k)$ for reasonable choices of $k$. 
“Elligator 1”,
reinterpreting and simplifying
2013 Fouque–Joux–Tibouchi:

Fix prime power $q \in 3 + 4\mathbb{Z}$;
s $\in \mathbb{F}_q^*$ with
$(s^2 - 2)(s^2 + 2) \neq 0$;
c $= 2/s^2$; $r = c + 1/c$;
d $= -(c + 1)^2/(c - 1)^2$.

Define $E : x^2 + y^2 = 1 + dx^2y^2$.
This is a complete Edwards curve.

For $\phi : \mathbb{F}_q \rightarrow E(\mathbb{F}_q)$ defined
on next slide: the only preimages of $\phi(t)$ under $\phi$ are $\{t, -t\}$. 
\[ \phi(\pm 1) = (0, 1). \]

Otherwise \( \phi(t) = (x, y) \) where
\[
\begin{align*}
  u & = \frac{(1-t)}{(1+t)}, \\
  v & = u^5 + (r^2 - 2)u^3 + u, \\
  X & = \chi(v)u, \\
  Y & = \\
    & (\chi(v)v)^{(q+1)/4} \chi(v) \chi(u^2 + 1/c^2), \\
  x & = (c - 1)sX(1 + X)/Y, \\
  y & = \frac{rX - (1 + X)^2}{rX + (1 + X)^2}.
\end{align*}
\]
“Elligator 2”, 2013 Bernstein–Hamburg–Krasnova–Lange (restricted to the easiest case):

Fix prime power $q \in 1 + 4\mathbb{Z}$; non-square $u \in \mathbb{F}_q$; $A, B \in \mathbb{F}_q^*$ with non-square $A^2 - 4B$; $\sqrt{-} : \mathbb{F}_q^2 \rightarrow \mathbb{F}_q$ with $\sqrt{\alpha^2} \in \{\alpha, -\alpha\}$.

Define $E : y^2 = x^3 + Ax^2 + Bx$.

For $\psi : \mathbb{F}_q \rightarrow E(\mathbb{F}_q)$ defined on next slide: the only preimages of $\psi(t)$ under $\psi$ are $\{t, -t\}$. 
\(\psi(0) = (0, 0).\)

Otherwise \(\psi(t) = (x, y)\) where

\[ v = -\frac{A}{1 + ut^2}, \]

\[ \epsilon = \chi(v^3 + Av^2 + Bv), \]

\[ x = \epsilon v - (1 - \epsilon)A/2, \]

\[ y = -\epsilon \sqrt{x^3 + Ax^2 + Bx}. \]

Proofs, inverse maps, etc.:

[elligator.cr.yp.to](http://elligator.cr.yp.to)
Asymptotic ECDL security

The original ECC advertising: Index calculus breaks RSA etc. in subexponential time. Scary! ECDL attack takes exp time.
Asymptotic ECDL security

The original ECC advertising: Index calculus breaks RSA etc. in subexponential time. Scary! ECDL attack takes exp time.

Reasonable conjecture \( \Rightarrow \)
2012 Petit–Quisquater using F4 solves ECDL\(_2\) in subexp time.
Do we throw away ECC\(_2\)?
Asymptotic ECDL security

The original ECC advertising: Index calculus breaks RSA etc. in subexponential time. Scary! ECDL attack takes exp time.

Reasonable conjecture $\Rightarrow$
2012 Petit–Quisquater using F4 solves ECDL$_2$ in subexp time.
Do we throw away ECC$_2$?

Replace F4 with XL?
Tung Chou is investigating.
Asymptotic ECDL security

The original ECC advertising: Index calculus breaks RSA etc. in subexponential time. Scary! ECDL attack takes exp time.

Reasonable conjecture \(\Rightarrow\) 2012 Petit–Quisquater using F4 solves ECDL_{2} in subexp time. Do we throw away ECC_{2}?

Replace F4 with XL?

Tung Chou is investigating.

Replace XL with Coppersmith to generalize ECC_{2} to ECC?
Concrete ECDL security

Typical for real-world ECC: the “NIST P-256” curve $E$: $y^2 = x^3 - 3x + a_6$ over $F_\mathbb{q}$ where $q = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, $a_6 = 41058363725152142129326129780047268409114441015993725554835256314039467401291$.

$E(F_\mathbb{q})$ has prime order $\ell$.

“NIST generator”: $P = (48439561293906451759052585252797914202762949526041747995844080717082404635286, \ldots 9)$. 
Textbook ECDL cost analysis:

\[ \approx \sqrt{\pi \ell / 2} \] group operations to compute DL in order-\( \ell \) group. Negation map gains factor \[ \approx \sqrt{2} \] for elliptic curves.

So \[ \approx 2^{128} \] group operations to compute P-256 ECDL.
Textbook ECDL cost analysis:

\[ \approx \sqrt{\pi \ell / 2} \] group operations to compute DL in order-\( \ell \) group. Negation map gains factor \( \approx \sqrt{2} \) for elliptic curves. So \( \approx 2^{128} \) group operations to compute P-256 ECDL.

This is the best algorithm that cryptanalysts have published for P-256 ECDL.
Textbook ECDL cost analysis:

\[ \approx \sqrt{\pi \ell/2} \] group operations to compute DL in order-\( \ell \) group. Negation map gains factor \( \approx \sqrt{2} \) for elliptic curves.

So \( \approx 2^{128} \) group operations to compute P-256 ECDL.

This is the best algorithm that cryptanalysts have published for P-256 ECDL.

But is it the best algorithm that exists?
Standard definition of “best”: minimize “time” (e.g., minimize total “time” over all inputs).

Many researchers have tried and failed to find better ECDL algorithms.

**Standard conjecture:**
For each \( p \in [0, 1] \), each P-256 ECDL algorithm with success probability \( \geq p \) takes “time” \( \geq 2^{128} p^{1/2} \).
Interlude regarding “time”

How much “time” does the following algorithm take?

```python
def pidigit(n0, n1, n2):
    if n0 == 0:
        if n1 == 0:
            if n2 == 0: return 3
            return 1
        if n2 == 0: return 4
        return 1
    if n1 == 0:
        if n2 == 0: return 5
        return 9
    if n2 == 0: return 2
return 6
```
Students in algorithm courses learn to count executed “steps”. Skipped branches take 0 “steps”. This algorithm uses 4 “steps”.
Students in algorithm courses learn to count executed “steps”. Skipped branches take 0 “steps”.

This algorithm uses 4 “steps”.

Generalization: There exists an algorithm that, given $n < 2^k$, prints the $n$th digit of $\pi$ using $k + 1$ “steps”.
Students in algorithm courses learn to count executed “steps”. Skipped branches take 0 “steps”.

This algorithm uses 4 “steps”.

Generalization: There exists an algorithm that, given $n < 2^k$, prints the $n$th digit of $\pi$ using $k + 1$ “steps”.

Variant: There exists a 258-“step” P-256 discrete-log attack (with 100% success probability).
Students in algorithm courses learn to count executed “steps”. Skipped branches take 0 “steps”.

This algorithm uses 4 “steps”.

Generalization: There exists an algorithm that, given $n < 2^k$, prints the $n$th digit of $\pi$ using $k + 1$ “steps”.

Variant: There exists a 258-“step” P-256 discrete-log attack (with 100% success probability). If “time” means “steps” then the standard conjectures are wrong.
1994 Bellare–Kilian–Rogaway:

“We say that $A$ is a $(t, q)$-adversary if $A$ runs in at most $t$ steps and makes at most $q$ queries to $O$.”
1994 Bellare–Kilian–Rogaway:

“We say that
A is a \((t, q)\)-adversary if
A runs in at most \(t\) steps and
makes at most \(q\) queries to \(O\).”

Oops: table-lookup attack
has very small \(t\).

Paper conjectured “useful” DES
security bounds. Any reasonable
interpretation of conjecture was
false, given paper’s definition.
Theorems in paper were vacuous.
2012 Bernstein–Lange:

There are more pathologies!

Assuming plausible heuristics, overwhelmingly verified by computer experiment:

There exists a P-256 ECDL algorithm that takes "time" \( \approx 2^{85} \) and has success probability \( \approx 1 \).

"Time" includes algorithm length.

Inescapable conclusion: **The standard conjecture is false.**
Our recommendations to fix the flawed security definitions, conjectures, proofs:

1. Switch from “time” to circuit $AT$.

(Related, online soon: Improved $AT$ exponents for batch NFS.)
Our recommendations to fix the flawed security definitions, conjectures, proofs:

1. Switch from “time” to circuit $AT$.

   (Related, online soon: Improved $AT$ exponents for batch NFS.)

2. Formalize constructivity.
Our recommendations to fix the flawed security definitions, conjectures, proofs:

1. Switch from “time” to circuit $AT$.

(Related, online soon: Improved $AT$ exponents for batch NFS.)

2. Formalize constructivity.

More details and attacks: cr.yp.to/nonuniform.html
DH speed records

Sandy Bridge cycles for high-security constant-time $a, P \rightarrow aP$ ("?" if not SUPERCOP-verified):

2012 Hamburg: 153000?
2012 Longa–Sica: 137000?
2013 Bos–Costello–Hisil–Lauter: 122728
2013 Oliveira–López–Aranha–Rodríguez-Henríquez: 114800?
2013 Faz-Hernández–Longa–Sánchez: 96000?
2014 Bernstein–Chuengsatiansup–Lange–Schwabe: 91460
Critical for 122728, 91460:

1986 Chudnovsky–Chudnovsky: traditional Kummer surface allows fast scalar mult. 
14\text{M} \text{ for } X(P) \mapsto X(2P).

2006 Gaudry: even faster. 
25\text{M} \text{ for } X(P), X(Q), X(Q - P) \mapsto X(2P), X(Q + P), \text{ including 6\text{M} by surface coefficients.}

2012 Gaudry–Schost: 1000000-CPU-hour computation found secure small-coefficient surface over $\mathbf{F}_{2^{127}-1}$. 
Hyper-and-elliptic-curve crypto

Typical example: Define \( H \):
\[
y^2 = (z - 1)(z + 1)(z + 2) \\
(z - 1/2)(z + 3/2)(z - 2/3)
\]
over \( \mathbb{F}_p \) with \( p = 2^{127} - 309 \);
\( J = \text{Jac } H \); traditional Kummer surface \( K \); traditional \( X : J \rightarrow K \).
Small \( K \) coeffs \((20 : 1 : 20 : 40)\).
Hyper-and-elliptic-curve crypto

Typical example: Define $H$:

$$y^2 = (z - 1)(z + 1)(z + 2)$$

$$\quad (z - 1/2)(z + 3/2)(z - 2/3)$$

over $\mathbb{F}_p$ with $p = 2^{127} - 309$;

$J = \text{Jac } H$; traditional Kummer surface $K$; traditional $X : J \to K$.

Small $K$ coeffs $(20 : 1 : 20 : 40)$.

Warning: There are errors in the Rosenhain/Mumford/Kummer formulas in 2007 Gaudry, 2010 Cosset, 2013 Bos–Costello–Hisil–Lauter. We have simpler, computer-verified formulas.
Define $F_{p^2} = F_p[i]/(i^2 + 1)$; 
$r = (7 + 4i)^2 = 33 + 56i$; 
$s = 159 + 56i$; $\omega = \sqrt{-384}$; 
$C : y^2 = rx^6 + sx^4 + sx^2 + r$. 
Define $F_{p^2} = F_p[i]/(i^2 + 1)$;
$r = (7 + 4i)^2 = 33 + 56i$;
$s = 159 + 56i; \omega = \sqrt{-384}$;
$C : y^2 = rx^6 + sx^4 + sx^2 + r$.

$(x, y) \mapsto (x^2, y)$ takes $C$ to $E$:
$y^2 = rx^3 + sx^2 + sx + r$. 
Define $F_{p^2} = F_p[i]/(i^2 + 1)$; 
$r = (7 + 4i)^2 = 33 + 56i$; 
s = 159 + 56i; $\omega = \sqrt{-384}$; 
$C : y^2 = rx^6 + sx^4 + sx^2 + r$. 

$(x, y) \mapsto (x^2, y)$ takes $C$ to $E :$ 
$y^2 = rx^3 + sx^2 + sx + r$. 

$(x, y) \mapsto (1/x^2, y/x^3)$ takes $C$ to 
$y^2 = \bar{r}x^3 + \bar{s}x^2 + sx + r$. 
Define $F_{p^2} = F_p[i]/(i^2 + 1)$;
$r = (7 + 4i)^2 = 33 + 56i$;
$s = 159 + 56i$; $\omega = \sqrt{-384}$;
$C : y^2 = rx^6 + sx^4 + \bar{s}x^2 + \bar{r}$.

$(x, y) \mapsto (x^2, y)$ takes $C$ to $E : y^2 = rx^3 + sx^2 + \bar{s}x + \bar{r}$.

$(x, y) \mapsto (1/x^2, y/x^3)$ takes $C$ to $y^2 = \bar{r}x^3 + \bar{s}x^2 + sx + r$.

$(z, y) \mapsto \left( \frac{1 + iz}{1 - iz}, \frac{\omega y}{(1 - iz)^3} \right)$
takes $H$ over $F_{p^2}$ to $C$. 
$J$ is isogenous to
Weil restriction $W$ of $E$, so computing $\#J(\mathbb{F}_p)$ is fast.
Here $\#J(\mathbb{F}_p) = 16 \cdot \text{prime}$; also reasonably twist-secure.
$J$ is isogenous to Weil restriction $W$ of $E$, so computing $\#J(\mathbb{F}_p)$ is fast. Here $\#J(\mathbb{F}_p) = 16 \cdot \text{prime}$; also reasonably twist-secure.

2003 Scholten: this strategy for building genus-2 curves with fast point-counting.
$J$ is isogenous to
Weil restriction $W$ of $E$, so computing $\#J(\mathbb{F}_p)$ is fast.
Here $\#J(\mathbb{F}_p) = 16 \cdot \text{prime}$; also reasonably twist-secure.

2003 Scholten:
this strategy for
building genus-2 curves
with fast point-counting.

What’s new here:
1. Small Kummer coefficients.
Requires lifting Scholten to $\mathbb{Q}$. 
2. Explicit formulas for isogenies \( \iota : W \to J \) and \( \iota' : J \to W \) with \( \iota \circ \iota' = 2 \).
2. Explicit formulas for isogenies \( \iota : W \to J \) and \( \iota' : J \to W \) with \( \iota \circ \iota' = 2 \).

We took random points in \( H(\mathbb{F}_p) \times H(\mathbb{F}_p) \); applied \( H(\mathbb{F}_p) \to C(\mathbb{F}_{p^2}) \to E(\mathbb{F}_{p^2}) = W(\mathbb{F}_p) \); interpolated formulas for \( \iota' \).

Similarly interpolated formulas for \( \iota \); verified composition.

Easy computer calculation.

“Wasting brain power is bad for the environment.”
3. Using isogenies to dynamically move computations between $E(\mathbb{F}_{p^2})$ and $J(\mathbb{F}_p)$.

e.g. Generate keys using fast formulas for $E$.
Compute shared secrets using fast formulas for $K$.

For more information:
see our talk at ANTS!
Paper coming soon.