Hyperelliptic-curve cryptography

D. J. Bernstein
University of Illinois at Chicago

Thanks to:
NSF DMS–0140542
NSF ITR–0716498
Alfred P. Sloan Foundation
Two parts to this talk:

1. Elliptic curves; “modern cryptography.”
2. Genus-2 hyperelliptic curves; “future cryptography.”

Will cryptography eventually move to genus 3, 4, 5, ...? Maybe, but current guess is that genus 2 is optimal.
Elliptic-curve computations

Write $p = 2^{255} - 19$; $p$ is a prime.

Costs of arithmetic in $\mathbb{F}_p$ with state-of-the-art software:
10 “ops” for $f, g \mapsto f + g$.
55 “ops” for $f \mapsto 121665 f$.
162 “ops” for $f \mapsto f^2$.
243 “ops” for $f, g \mapsto fg$.

1.3GHz Pentium M: 1.3 cycles/ns; typically $\approx 1$ “op”/cycle.

Newer chips than the Pentium M: more cycles/ns; more “ops”/cycle.
“Curve25519” is the elliptic curve
\[ y^2 = x^3 + 486662x^2 + x \] over \( \mathbb{F}_p \).

“Curve25519(\( \mathbb{F}_p \))” is
the commutative group
\( \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : \ y^2 = x^3 + 486662x^2 + x \} \cup \{\infty\} \)
with chord-and-tangent addition.

Neutral element of the group: \( \infty \).

Negation in the group:
\( \infty \mapsto \infty; (x, y) \mapsto (x, -y) \).

Chord-and-tangent idea:
points on a line add to 0,
when counted with multiplicity.
Chord-and-tangent definition:

• \(\infty + \infty = \infty;\)
• \((x_1, y_1) + \infty = (x_1, y_1);\)
• \(\infty + (x_2, y_2) = (x_2, y_2);\)
• \((x_1, y_1) + (x_1, -y_1) = \infty;\)
• for \(y_1 \neq 0, (x_1, y_1) + (x_1, y_1) = (x_3, y_3)\) where
  \[x_3 = \lambda^2 - 486662 - x_1 - x_1,\]
  \[y_3 = \lambda(x_1 - x_3) - y_1,\]
  \[\lambda = \frac{(3x_1^2 + 973324x_1 + 1)}{2y_1};\]
• for \(x_1 \neq x_2, (x_1, y_1) + (x_2, y_2) = (x_3, y_3)\) where
  \[x_3 = \lambda^2 - 486662 - x_1 - x_2,\]
  \[y_3 = \lambda(x_1 - x_3) - y_1,\]
  \[\lambda = \frac{(y_2 - y_1)}{(x_2 - x_1)}.\]
Profusion of cases is annoying for mathematicians and programmers. Do we need so many cases?

Can cover $E(k) \times E(k)$ with 3 open addition laws. (1985 H. Lange–Ruppert)

How about just one law that covers $E(k) \times E(k)$?

One complete addition law?

Can avoid expensive divisions using projective coordinates

\[(X : Y : Z) \mapsto (X/Z, Y/Z)\].

\[12M + 2S\] for \(Q, R \mapsto Q + R\).
\[7M + 3S\] for \(Q \mapsto 2Q\)
on \(y^2 = x^3 - 3x + a_6\);
slightly slower without \(a_4 = -3\).

(1986 Chudnovsky–Chudnovsky)

Here \(M\) is mult in \(F_p\),
\(S\) is squaring in \(F_p\).

For full performance picture
also have to count adds in \(F_p\).
Or “Jacobian” coordinates 
\((X : Y : Z) \mapsto (X/Z^2, Y/Z^3)\).

12M + 4S for \(Q, R \mapsto Q + R\).
4M + 4S for \(Q \mapsto 2Q\)
on \(y^2 = x^3 - 3x + \text{const.}\)
(1986 Chudnovsky–Chudnovsky)

11M + 5S; 3M + 5S.
(2001 Bernstein)

Many more coordinate systems.
Survey and various improvements:
“Explicit-Formulas Database,”
http://hyperelliptic.org/EFD
(joint work with Tanja Lange)
From \( n \in \mathbb{Z}, \; Q \in \text{Curve25519}(\mathbb{F}_p) \) compute \( nQ \in \text{Curve25519}(\mathbb{F}_p) \) using \( O(\lg n) \) curve additions.

Recursion: \( 0Q = \infty; \; 1Q = Q; \)
\(-1)Q = -Q; \; 2nQ = 2(nQ); \)
\((2n + 1)Q = 2nQ + Q.\)

Faster: “Sliding windows.”
e.g. \((8n + 7)Q = 8nQ + 7Q\)
after precomputing \(3Q, 5Q, 7Q\).
Asymptotics: \( \approx \lg n \) doublings,
\( \approx (\lg n)/\lg \lg n \) more additions.

For average \( n \approx 2^{255}: \)
\( \approx 252 \) doublings, \( \approx 50 \) additions;
\( \approx 2400 \) “ops” per bit of \( n.\)
Or (1987 Montgomery): Compute $x(Q), x(2nQ), x((2n + 1)Q)$ or $x(Q), x((2n + 1)Q), x((2n + 2)Q)$, given $x(Q), x(nQ), x((n + 1)Q)$, using $5M + 4S + 1D + 8\text{add}$, where $D$ is mult by 121665.

Only 1998 “ops” per bit of $n$.

$n, x(Q) \mapsto x(nQ)$ for $n \approx 2^{255}$ in $< 500\mu s$ on 1.3GHz Pentium M. (2005 Bernstein)

$n, x(Q) \mapsto x(nQ)$ for $n \approx 2^{255}$ in $< 170\mu s$ on 2.4GHz Core 2; $> 24000 \; nQ/\text{sec}$ using four cores. (2007 Gaudry–Thomé)
Elliptic-curve Diffie–Hellman
(1986 Miller; 1987 Koblitz)

$\bar{g} = (9, \ldots)$ is a standard element of Curve25519($\mathbb{F}_p$) with order $p_1$.

I have a “secret key”: an integer $n \in \{0, 1, \ldots, 2^{256} - 1\}$.

I compute a “public key” $x(n\bar{g})$ and publish it. 32 bytes.

You have a secret key $m$.
You publish $x(m\bar{g})$.

We compute secret $x(mn\bar{g})$.
Then “ciphers” such as “AES” encrypt and authenticate data.
\#Curve25519(F_p) \approx 2^{255};
in fact \#Curve25519(F_p) = 8p_1
for a known prime \( p_1 \approx 2^{252} \).

Attacker can compute \( x(mn\bar{9}) \)
using \( \approx \sqrt{p_1} \approx 2^{126} \) adds.
No faster attacks known.

Side notes to cryptographers:
\( p \) has large order mod \( p_1 \);
\( 2p + 2 - \#Curve25519(F_p) = 4p_2 \)
for a known prime \( p_2 \approx 2^{253} \);
\( p \) has large order mod \( p_2 \);
\((p + 1 - 8p_1)^2 - 4p\) is not
a small multiple of a square.
Elliptic-curve signatures

I sign a message $m$ by generating another secret $s$, computing $R = s\bar{9}$, computing $t = H(R, m)s + n \mod p_1$. Here $H$ is a standard “hash function” such as “SHA-256.”

Signature is $(R, t)$. Anyone can verify $t\bar{9} = H(R, m)R + n\bar{9}$. No fast attacks known.

(first similar idea: 1985 ElGamal; many generalizations, variations; these choices: 2006 van Duin)
Compute $t\bar{g} - H(R, m)R$ using $\approx 252$ doublings, $\approx 100$ additions.

Even better: To verify a batch
\[ t_1\bar{g} - h_1R_1 = K_1, \]
\[ t_2\bar{g} - h_2R_2 = K_2, \]
\[ \vdots \]
\[ t_{100}\bar{g} - h_{100}R_{100} = K_{100}: \]
Verify linear combination
\[ (v_1t_1 + \cdots + v_{100}t_{100})\bar{g} - v_1h_1R_1 - \cdots - v_{100}h_{100}R_{100} - v_1K_1 - \cdots - v_{100}K_{100} = 0 \]
for random 128-bit $v_1, \ldots, v_{100}$.
(1994 Naccache et al.; 1998 Bellare et al.)
Use subtractive multi-scalar multiplication algorithm:

if \( n_1 \geq n_2 \geq \cdots \) then

\[
R_1 + R_2 + \cdots = (n_1 - qn_2)R_1 + n_2(qR_1 + R_2) + n_3R_3 + \cdots
\]

where \( q = \lfloor n_1/n_2 \rfloor \).

(credited to Bos and Coster by 1994 de Rooij;

see also tweaks by 2007 Wei Dai)

Only \( \approx 25.2 \) curve adds/bit
to verify 100 signatures.

Doublings are negligible here;
want fast \( Q, R \mapsto Q + R \).

Projective is better than Jacobian.
More curves

Same cryptographic protocols work with any “fast” group. Let’s try another group.

\[ \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : x^2 + y^2 = 1\} \]

is a commutative group with

\[(x_1, y_1) + (x_2, y_2) = (x_3, y_3)\]

where \( x_3 = x_1y_2 + x_2y_1 \)

and \( y_3 = y_1y_2 - x_1x_2 \).

Addition law is complete and fast! Only 3\( \mathbf{M} \) for \( Q, R \mapsto Q + R \).

But this curve is vulnerable to “index calculus.” Security requires larger \( p \), outweighing speedup.
If $d$ is not a square in $\mathbb{F}_p$
then \[\{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : \quad x^2 + y^2 = 1 + dx^2y^2\}\]
is a commutative group with
\[(x_1, y_1) + (x_2, y_2) = (x_3, y_3)\]
defined by Edwards addition law:

\[x_3 = \frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2},\]
\[y_3 = \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2}.\]

\[d = -1: \ 1761 \text{ Euler, 1866 Gauss};\]
any $d = c^4$: 2007 Edwards;
addition is complete for $d \neq \Box$: 2007 Bernstein–Lange)
Outline of completeness proof: use curve equation to see that
\((dx_1x_2y_1y_2)^2 = 1\)
\(\Rightarrow (x_1 + dx_1x_2y_1y_2y_1)^2 = dx_1^2y_1^2(x_2 + y_2)^2\)
\(\Rightarrow d\) is a square. □

This curve has genus 1!
Equivalent to an elliptic curve.
e.g. Curve25519 is equivalent to the complete Edwards curve
\(x^2 + y^2 = 1 + (1 - 1/121666)x^2y^2\).

Edwards addition law is complete despite Bosma–Lenstra theorem.
Edwards curves are fast!
(2007 Bernstein–Lange)

Can use projective coordinates.
10\textit{M} + 1\textit{S} for \( Q, R \mapsto Q + R \).
3\textit{M} + 4\textit{S} for \( Q \mapsto 2Q \),
assuming \( d \) is small.

Can sacrifice completeness
and use “inverted” coordinates
\((X : Y : Z) \mapsto (Z/X, Z/Y)\).
9\textit{M} + 1\textit{S} for \( Q, R \mapsto Q + R \).
3\textit{M} + 4\textit{S} for \( Q \mapsto 2Q \),
assuming \( d \) is small.
Why do we use $\mathbb{F}_p$?
Why not, e.g., $\mathbb{F}_{2^{251}}$?

“Binary Edwards curves”

$$d_1(x + y) + d_2(x^2 + y^2)$$

$$= xy(1 + x)(1 + y)$$

have complete addition law
if $x^2 + x + d_2$ is irreducible;
also fast doublings etc.
(2008 Bernstein–Lange–Reza Rezaeian Farashahi)

2008.03.31 news: Intel announces support for $\mathbb{F}_2$ poly mult
in next year’s chips.
What about genus 2?

Choose much smaller prime $q$, say $q = 2^{127} - 1$.

Costs of arithmetic in $\mathbb{F}_q$:
5 “ops” for $f, g \mapsto f + g$.
57 “ops” for $f \mapsto f^2$.
73 “ops” for $f, g \mapsto fg$.

Recall 10, 162, 243 for arithmetic in $\mathbb{F}_{2^{255}-19}$. $\mathbb{F}_q$ is much faster.
$2 \times$ faster for $f, g \mapsto f + g$.
$2.842 \times$ faster for $f \mapsto f^2$.
$3.329 \times$ faster for $f, g \mapsto fg$. 
Choose genus-2 hyperelliptic curve $C$ over $\mathbb{F}_q$ with unique $\infty$.

How fast is arithmetic in the group $(\text{Jac } C)(\mathbb{F}_q)$?

Is $\text{Jac } C$ faster than Curve25519?

Similar group size, $\approx 2^{254}$.

Conjecturally similar security for these cryptographic protocols.

Basic disadvantage of genus 2: $\#\mathcal{M}$ for addition is much larger for $\text{Jac } C$ than for Curve25519.

Basic advantage of genus 2: $\mathbb{F}_q$ is much faster than $\mathbb{F}_p$. Does this outweigh the disadvantage?
Can use Gauss-style algorithm (Cantor; Koblitz) to multiply in ideal-class group.

Many genus-2 speedups:
2000 Harley; 2001 Lange;
2001 Matsuo–Chao–Tsujii;
2002 Miyamoto–Doi–Matsuo–Chao–Tsujii; 2002 Takahashi;
culminating in 2002 Lange,
\[34M + 7S\text{ for } P \mapsto 2P.\]
Still not as fast as genus 1.

More speedups for binary genus 2.
Faster than binary genus 1!
Still not as fast as non-binary.
Alternative: compute
$x(P), x(2nP), x((2n + 1)P)$ or
$x(P), x((2n + 1)P), x((2n + 2)P)$,
given $x(P), x(nP), x((n + 1)P)$,
where $x : (\text{Jac } C)/\{\pm 1\} \hookrightarrow K$
is a standard rational map
to Kummer surface $K \subset \mathbb{P}^3$.

Can do this computation
in just $16\mathbf{M} + 9\mathbf{S}$.
(2005 Gaudry, improving
1986 Chudnovsky–Chudnovsky)

Analogous to Montgomery’s
$x : E/\{\pm 1\} \hookrightarrow \mathbb{P}^1$. 
Gaudry’s formulas use 1841 “ops” for each bit of $n$.
New software speed records.
(2006 Bernstein)

But wait, there’s more!
“A few multiplications can be saved” by small choices of $C$.
(2005 Gaudry)

$7M + 12S$ for small $C$.
1659 “ops,” and as few as
1355 “ops” for extremely small $C$.
(2006 Bernstein)
Problem: For security, need large prime in \( \#(\text{Jac } C)(\mathbb{F}_q) \), like \( p_1 = \#\text{Curve25519}(\mathbb{F}_p)/8 \). Also, signers need to know prime.

How do we compute \( \# \text{ Jac } C \)?

Strategy 1: Build \( C \) by CM. Trivially write down \( \# \text{ Jac } C \).

Problem: \( C \) isn’t small!

We want better speeds.

Strategy 2: Choose a small \( C \). Compute \( \# \text{ Jac } C \mod \ell \) for several small primes \( \ell \).
Strategy 2 is “polynomial time” (1985 Schoof; 1990 Pila) . . . but much, much, much slower for genus 2 than for genus 1.

$q \approx 2^{64}$: 2000 Gaudry–Harley.

$q \approx 2^{80}$: 2004 Gaudry–Schost.

$q \approx 2^{100}$: 2008 Gaudry–Schost.

For one candidate curve $C$,

$\approx 1.3 \cdot 2^{51}$ CPU cycles.

$\approx 1.2 \cdot 2^{33}$ bytes RAM.
How does strategy 2 work?

Write down generic point $P \in \text{Jac } C$ with $\ell P = 0$.

Specifically: express $\ell P = 0$ as system of equations on coordinates of $P$;
extend $F_q$ to ring $R = F_q[\text{coords}]/\text{equations}$;
note that $\ell P = 0$ in $(\text{Jac } C)(R)$.

Genus 1: $\# R \approx q^{\ell^2}$.

Genus 2: $\# R \approx q^{\ell^4}$.

Much larger computations.
Define $q$th-power Frobenius map $\varphi : (\text{Jac } C)(\mathbb{F}_q) \rightarrow (\text{Jac } C)(\mathbb{F}_q)$.

**Genus 1:** Find linear equation

$$\varphi^2(P) - s_1 \varphi(P) + qP = 0$$

with $s_1 \in \{0, 1, \ldots, \ell - 1\}$. Then

$$1 - s_1 + q - \# \text{Jac}(C)(\mathbb{F}_q) \in \ell \mathbb{Z}.$$ 

**Genus 2:** Find linear equation

$$\varphi^4(P) - s_1 \varphi^3(P) + s_2 \varphi^2(P) - qs_1 \varphi(P) + q^2P = 0$$

with $s_1, s_2 \in \{0, 1, \ldots, \ell - 1\}$. Then

$$1 - s_1 + s_2 - qs_1 + q^2$$

$$- \# \text{Jac}(C)(\mathbb{F}_q) \in \ell \mathbb{Z}.$$
Typical papers replace $R$ by a field quotient, allegedly saving time.

Bad idea for large $q$.

Finding field quotients loses more time than it saves. “Factorization is slow.”

Can save time in genus 1 by building a smaller $R$ that defines a Frob-stable subgroup of $\ell$-torsion. (1991 Elkies; 1992 Atkin)

But analogous techniques seem to lose time in genus 2.
Which coords to choose?

Gaudry et al. write \( P = P_1 - P_2 \) with \( P_i = (x_i, y_i) \in C \to \text{Jac } C \). Equation \( \ell P_i = \ell P_j \) gives two equations in \( x_1, x_2 \). Eliminate \( x_2 \), obtaining equation in \( x_1 \).

Elimination time \((\ell^6 \log q)^{1+o(1)}\) using fast-arithmetic techniques.

Several constant-factor speedups: symmetrize; \# Jac \( C \mod 2^2 \) etc.; reduce \# Jac \( C \) range; et al.
With my student Nikki Pitcher: various improvements, including log-factor speedup ("faster poly multiplication"), log-factor space reduction ("low-memory interpolation").

Clearly \( q \approx 2^{128} \) is reachable. Moderate computation will find small secure genus-2 curves, new leaders for Diffie–Hellman.

But what about signatures? Addition speed is paramount. Open: genus-2 Edwards?