Elliptic-curve cryptography

Tanja Lange
Technische Univesiteit Eindhoven
The Netherlands

July 23, 2014

Elliptic-curve cryptography: signatures and key exchange

Given: some elliptic curve E, point P on E of known order ℓ.

Key exchange:
Alice picks random a, computes aP, sends it to Bob,
receives bP, computes $a(bP)$,
uses KDF(abP) for authenticated encryption with Bob.

Fairly standard; differences in key-derivation function.

Signatures:
Alice has public key aP, secret key a. Wants to sign message m.
Computes $h(m)$, h a hash function.
Picks random k, computes $R = kP$, $s = k^{-1}(h(m) + ar) \bmod \ell$,
where r is derived from R, signature is (r, s).

This is close to ECDSA. Standards vary in definition of s,
e.g. in German ECGDSA and Korean EC-KCDSA.

EdDSA also changes what is hashed and how k is chosen.

Elliptic curves in math

Definition

An elliptic curve is a smooth, projective algebraic curve of genus one with at least one point.

Elliptic curves in math

Definition

An elliptic curve is a smooth, projective algebraic curve of genus one with at least one point.

Definition

Let K be a field and let $a_1, a_2, a_3, a_4, a_6 \in K$. The equation

$$E : y^2 + (a_1x + a_3)y = x^3 + a_2x^2 + a_4x + a_6$$

defines an elliptic curve if there is no point (x_1, y_1) on $E(\bar{k})$ satisfying $2y_1 + a_1x_1 + a_3 = 0$ and $a_1y_1 = 3x_1^2 + 2a_2x_1 + a_4$.

This equation form is called a *Weierstrass curve*.

Elliptic curves in Weierstrass form

For crypto want $K = \mathbf{F}_p$, p an odd prime[1]. This simplifies the curve equation to $y^2 = x^3 + ax + b$; RHS has no double roots.

Most standards use $y^2 = x^3 - 3x + b$, this gives some speed up and is generic.

The group used for crypto is the set

$$E(\mathbf{F}_p) = \{(x_1, y_1) \in \mathbf{F}_p \times \mathbf{F}_p | y_1^2 = x_1^3 + ax_1 + b\} \cup \{\infty\}.$$

[1] Some hardware systems benefit from $K = \mathbf{F}_{2^p}$.

Elliptic curves in Weierstrass form

For crypto want $K = \mathbf{F}_p$, p an odd prime[1]. This simplifies the curve equation to $y^2 = x^3 + ax + b$; RHS has no double roots.

Most standards use $y^2 = x^3 - 3x + b$, this gives some speed up and is generic.

The group used for crypto is the set

$$E(\mathbf{F}_p) = \{(x_1, y_1) \in \mathbf{F}_p \times \mathbf{F}_p | y_1^2 = x_1^3 + ax_1 + b\} \cup \{\infty\}.$$

Computations usually done without inversions.
Represent $P_1 = (x_1, y_1)$ as $(X_1 : Y_1 : Z_1)$,
with $x_1 = X_1/Z_1$ and $y_1 = Y_1/Z_1$ for $Z_1 \neq 0$.
This also gives a representation to ∞, namely $(0 : 1 : 0)$.

The Z coordinate collects all the divisions; working with fractions requires cross-multiplications. Doublings $(P_1 + P_1)$ become faster than general additions $(P_1 + P_2)$.

[1] Some hardware systems benefit from $K = \mathbf{F}_{2^p}$.

Coordinate systems for Weierstrass curves

Old ($\leq$ 1985) systems:

- Affine: $P_1 = (x_1, y_1)$.
- Projective: $P_1 = (X_1 : Y_1 : Z_1)$ with $x_1 = X_1/Z_1$ and $y_1 = Y_1/Z_1$ for $Z_1 \neq 0$.
- Jacobian: $P_1 = (X_1 : Y_1 : Z_1)$ with $x_1 = X_1/Z_1^2$ and $y_1 = Y_1/Z_1^3$ for $Z_1 \neq 0$.
- x-coordinate only: $P_1 = (X_1 : Z_1)$ with $x_1 = X_1/Z_1$; does not distinguish between P_1 and $-P_1 = (x_1, -y_1)$. Can compute $x(kP_1)$ from $x(P_1)$.

Somewhat newer systems

- Extended Jacobian: $P_1 = (X_1 : Y_1 : Z_1 : Z_1^2 : Z_1^3)$ with $x_1 = X_1/Z_1^2$ and $y_1 = Y_1/Z_1^3$ for $Z_1 \neq 0$.
- Modified Jacobian: $P_1 = (X_1 : Y_1 : Z_1 : aZ_1^4)$ with $x_1 = X_1/Z_1^2$ and $y_1 = Y_1/Z_1^3$ for $Z_1 \neq 0$.

Addition law on Weierstrass curves

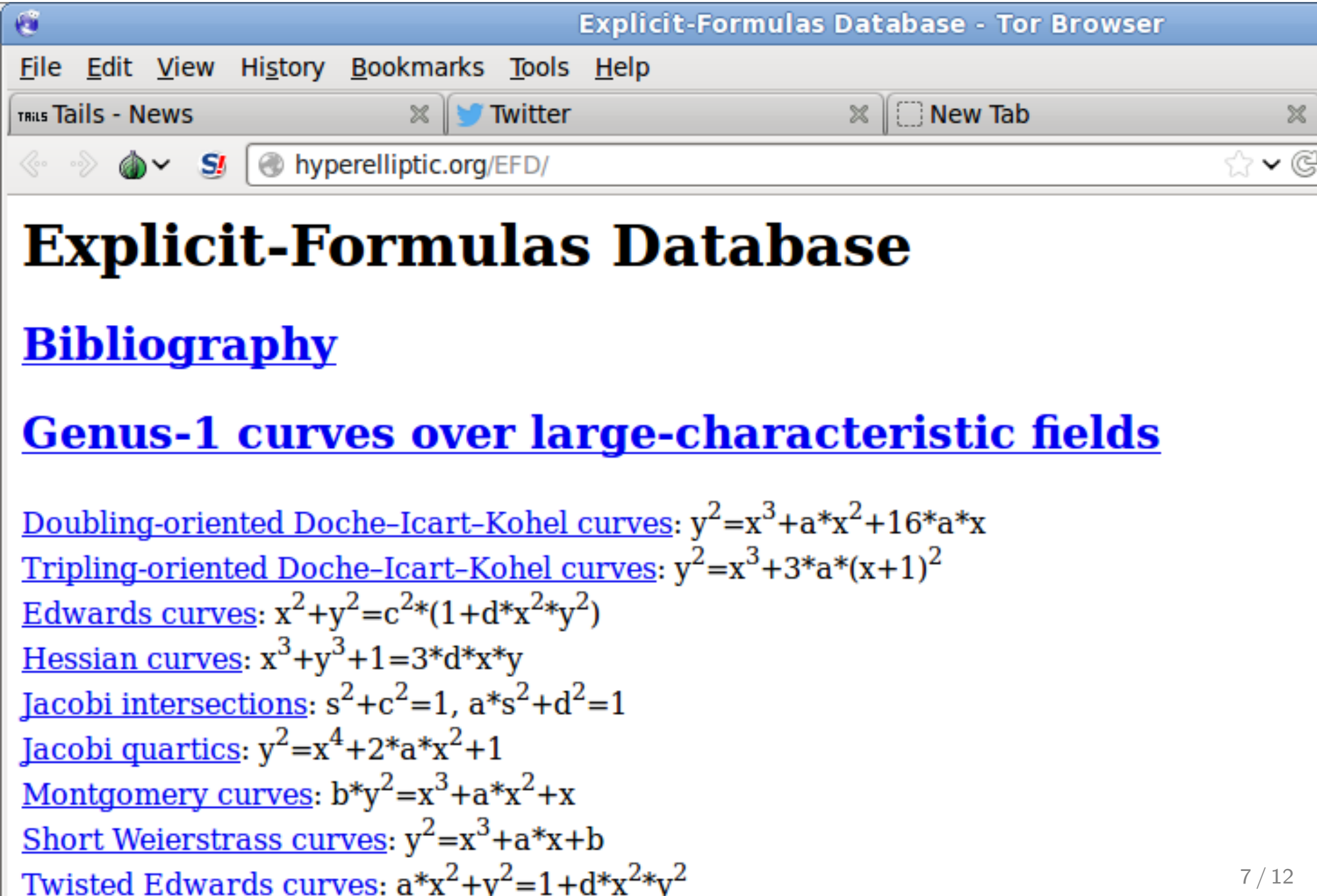
Compute $P_3 = P_1 + P_2$:

1. if $P_1 = \infty$ then $P_3 = P_2$;
2. elseif $P_2 = \infty$ then $P_3 = P_1$;
3. elseif $P_1 = -P_2$ then $P_3 = \infty$;
4. elseif $P_1 = P_2$ then $\lambda = (3x_1^2 + a)/(2y_1)$, $x_3 = \lambda^2 - 2x_1$, $y_3 = \lambda(x_1 - x_3) - y_1$;
5. else $\lambda = (y_1 - y_2)/(x_1 - x_2)$, $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$.

Projective coordinates do not magically take care of ∞ or $P_1 + (-P_1)$; so most cases remain.

Do **not** do the case distinction by if/else, this gives a timing attack.

Other curve shapes https://hyperelliptic.org/EFD/



Security analysis of curve shapes

Any elliptic curve, e.g. twisted Edwards curve $ax^2 + y^2 = 1 + dx^2y^2$, is birationally equivalent to a Weierstrass curve $y^2 = x^3 + a_4x + a_6$ over the same field.

Such a map ϕ respects addition, so the discrete logarithm problem has equal strength on both curves: given $Q = kP$ on E_E we get $\phi(Q) = k\phi(P)$ on E_W. No new analysis of DLP hardness needed.

Not every Weierstrass curve is birationally equivalent to a twisted Edwards, Hessian, Montgomery, . . . curve over **the same field**. They are birationally equivalent over a small extension of the field $\mathbf{F}_{p^m}$, with $m \leq 9$. This gives assurance for the security of the ECDLP.

Security analysis of curve shapes

Any elliptic curve, e.g. twisted Edwards curve $ax^2 + y^2 = 1 + dx^2y^2$, is birationally equivalent to a Weierstrass curve $y^2 = x^3 + a_4x + a_6$ over the same field.

Such a map ϕ respects addition, so the discrete logarithm problem has equal strength on both curves: given $Q = kP$ on E_E we get $\phi(Q) = k\phi(P)$ on E_W. No new analysis of DLP hardness needed.

Not every Weierstrass curve is birationally equivalent to a twisted Edwards, Hessian, Montgomery, . . . curve over **the same field**. They are birationally equivalent over a small extension of the field $\mathbf{F}_{p^m}$, with $m \leq 9$. This gives assurance for the security of the ECDLP.

But there are differences in how ECC is implemented on these. DJB and I, 2013-05-31 "If you implement the NIST curves, chances are you're doing it wrong".

Attacks on DLP

Area of extensive research over past > 25 years.
For curves of almost prime order over $\mathbf{F}_p$ the strongest attack is parallel Pollard rho with negation ...

Attacks on DLP

Area of extensive research over past > 25 years.
For curves of almost prime order over $\mathbf{F}_p$ the strongest attack is parallel Pollard rho with negation . . .
. . . unless the curve has some very special properties.
These are easy to check for:
http://safecurves.cr.yp.to/verify.html has Sage scripts to verify that the curve does not

- have too small group order;
- allow additive transfers;
- have small embedding degree;
- have small discriminant.

The scripts also check that all primes are indeed primes etc.

The criterion on the discriminant appears only in the Brainpool-curves description, but is not a restriction in practice.

ECC security – protocol/side-channel attacks

Implementations need to ensure that received point is valid:

- Point is on the curve.
- Point has the correct group order.

Can skip first check

- if point is reconstructed from compressed form;
- if curve is twist-secure and only x is used.

Can skip second check

- if group order is prime or protocol takes care of it.

Most protocols include handling of cofactors.

Implementations need to ensure that (cache) timing information does not leak information on the used secrets:

- No data-dependent execution time.
- No data-dependent branches.
- No data-dependent array indices.

Montgomery curves

Peter L. Montgomery, 1987 (submitted 1985)

$$E_M : By^2 = x^3 + Ax^2 + x$$

Curve shape perfectly suited for x-coordinate-only addition, used in scalar multiplication.

Keep two points $P_1 = mP$ and $P_2 = mP + P$ at distance P.
Per bit of scalar add these and double one of them to compute nP.

$11 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$

$(0P, 1P) \overset{1\cdot 2^3}{\mapsto} (1P, 2P) \overset{0\cdot 2^2}{\mapsto} (2P, 3P) \overset{1\cdot 2^1}{\mapsto} (5P, 6P) \overset{1\cdot 2^0}{\mapsto} (11P, 12P)$

This gives a very regular execution pattern, leading to side-channel secured software implementations.

Twisted Edwards curves

Harold M. Edwards, 2007

$$ax^2 + y^2 = 1 + dx^2y^2$$

$-(x_1, y_1) = (-x_1, y_1)$, very different addition formulas.

If a is a square and d is a non-square in $\mathbf{F}_p$ (exactly half of all elements are squares) then addition is **complete**.
This means $P_1 + P_2$ does not need any case distinction.

Can get a very regular execution pattern by using DBL, DBL, ADD.
$11 = 2^4 - 2^2 - 2^0$, thus $11P = 2(2(2(2P) - P)) - P$
(use care in choosing P and $-P$ to add).

In general can get a very regular execution pattern by using signed fixed windowing. Constant-time methods with more precomputation exist for fixed basepoint.

Big benefit: can compute $Q + 0P$ and $P + P$ with regular ADD.