Elliptic curves for applications

Tanja Lange
Technische Universiteit Eindhoven

part of this material is joint work
with Daniel J. Bernstein
University of Illinois at Chicago
and
Peter Schwabe
Academia Sinica, Taiwan

Why ECC?

January 2010 news:
An academic team announces successful RSA-768 factorization.
Used ≈ 2 years of computation on ≈ 1000 CPU cores.
"Factoring a 1024-bit RSA modulus would be about a thousand times harder."

Why ECC?

January 2010 news:
An academic team announces successful RSA-768 factorization.
Used ≈ 2 years of computation on ≈ 1000 CPU cores.
“Factoring a 1024-bit RSA modulus would be about a thousand times harder.”

Many users of 1024-bit RSA:
EFF SSL observatory study
http://eff.org/observatory
shows that more than 50% of all certificates are $\leq$ 1024-bit RSA.

1000 cores in perspective:

Typical laptop has 2 cores.

1000 cores in perspective:

Typical laptop has 2 cores.

A GTX 295 graphics card has 60 cores (“MPs”).

1000 cores in perspective:

Typical laptop has 2 cores.

A GTX 295 graphics card has 60 cores ("MPs").

EPFL's 200-Playstation cluster has 1200 cores.

1000 cores in perspective:

Typical laptop has 2 cores.

A GTX 295 graphics card
has 60 cores (“MPs”).

EPFL’s 200-Playstation
cluster has 1200 cores.

Dan had an account on the
TACC Ranger supercomputer,
which has 62976 cores.

1000 cores in perspective:

Typical laptop has 2 cores.

A GTX 295 graphics card has 60 cores ("MPs").

EPFL's 200-Playstation cluster has 1200 cores.

Dan had an account on the TACC Ranger supercomputer, which has 62976 cores.

The Conficker/Downadup criminal-controlled botnet has $\approx 10\,000\,000$ cores.

2003 Shamir et al.:
An attacker building ASICs for 10 million USD can break RSA-1024 in a year.

2003 RSA company:
Move to 2048 bits "over the remainder of this decade."

2003 Shamir et al.:
An attacker building ASICs for 10 million USD can break RSA-1024 in a year.

2003 RSA company:
Move to 2048 bits "over the remainder of this decade."

2007 NIST: Same.

2003 Shamir et al.:
An attacker building ASICs for 10 million USD can break RSA-1024 in a year.

2003 RSA company:
Move to 2048 bits "over the remainder of this decade."

2007 NIST: Same.

These recommendations don't even take into account *batch* algorithms that save time in breaking many keys together.

A 1024-bit RSA key is built from two secret 512-bit primes.

There are $\approx 2^{503}$ possible 512-bit primes. Can't imagine trying them all. But the attacks are much faster: only $\approx 2^{80}$ calculations.

A 1024-bit RSA key is built from two secret 512-bit primes.

There are $\approx 2^{503}$ possible 512-bit primes.
Can't imagine trying them all.
But the attacks are much faster: only $\approx 2^{80}$ calculations.

2048-bit key: 1024-bit primes; $\approx 2^{1014}$ possible primes.
Still below modern standards!
Attacks: $\approx 2^{112}$ calculations.

A 1024-bit RSA key is built from two secret 512-bit primes.

There are $\approx 2^{503}$ possible 512-bit primes.
Can't imagine trying them all.
But the attacks are much faster: only $\approx 2^{80}$ calculations.

2048-bit key: 1024-bit primes; $\approx 2^{1014}$ possible primes.
Still below modern standards!
Attacks: $\approx 2^{112}$ calculations.

3072-bit key: 1536-bit primes; $\approx 2^{1526}$ possible primes.
Attacks: $\approx 2^{128}$ calculations.

Attacks use "index calculus"
= "combining congruences."
Long history, including
many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS).
Also many smaller improvements.

Costs of these algorithms for
breaking RSA-1024, RSA-2048:
$\approx 2^{120}$, 2^{170}, CFRAC;
$\approx 2^{110}$, 2^{160}, LS;
$\approx 2^{100}$, 2^{150}, QS;
$\approx 2^{80}$, 2^{112}, NFS.

1977: RSA is introduced.

1985: Miller proposes ECC.
Explains several obstacles
to congruence-combination
attacks on elliptic curves.

Subsequent ECC history:
Improved algorithms, but
still take exponential time.

Subsequent RSA history:
Continued security losses
from improved algorithms
for combining congruences.
Major loss in 1990 (NFS);
many smaller losses since then.

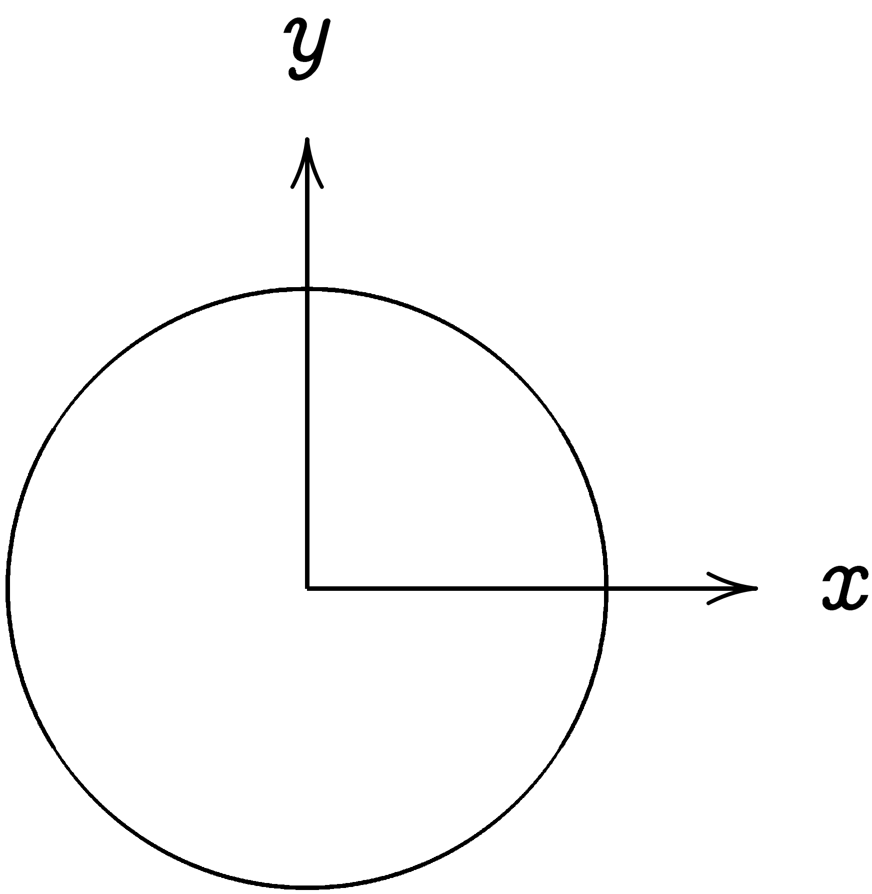
256-bit ECC keys match security of 3072-bit RSA keys.

When properly implemented, 256-bit ECC is much faster than 3072-bit RSA for almost all real-world applications.

ANSI, IEEE, NIST issued ECC standards ten years ago. US government “Suite B” now prohibits RSA, requires ECC.

Electronic ID or travel documents in many European countries based on ECC. Some internet protocols (DNSCurve, CurveCP) use ECC.

<u>The clock</u>



This is the curve $x^2 + y^2 = 1$.

Warning:
This is *not* an elliptic curve.
“Elliptic curve” $\neq$ “ellipse.”

Examples of points on this curve:

Examples of points on this curve:

$(0, 1) =$ "12:00".

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

Examples of points on this curve:

$(0, 1) =$ “12:00”.

$(0, -1) =$ “6:00”.

$(1, 0) =$ “3:00”.

$(-1, 0) =$ “9:00”.

$(\sqrt{3/4}, 1/2) =$

Examples of points on this curve:

$(0, 1) =$ “12:00”.

$(0, -1) =$ “6:00”.

$(1, 0) =$ “3:00”.

$(-1, 0) =$ “9:00”.

$(\sqrt{3/4}, 1/2) =$ “2:00”.

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$

Examples of points on this curve:

$(0,1) =$ "12:00".

$(0,-1) =$ "6:00".

$(1,0) =$ "3:00".

$(-1,0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

$(\sqrt{1/2}, \sqrt{1/2}) =$ "1:30".

$(3/5, 4/5)$. $(-3/5, 4/5)$.

Examples of points on this curve:

$(0, 1) =$ “12:00”.

$(0, -1) =$ “6:00”.

$(1, 0) =$ “3:00”.

$(-1, 0) =$ “9:00”.

$(\sqrt{3/4}, 1/2) =$ “2:00”.

$(1/2, -\sqrt{3/4}) =$ “5:00”.

$(-1/2, -\sqrt{3/4}) =$ “7:00”.

$(\sqrt{1/2}, \sqrt{1/2}) =$ “1:30”.

$(3/5, 4/5)$. $(-3/5, 4/5)$.

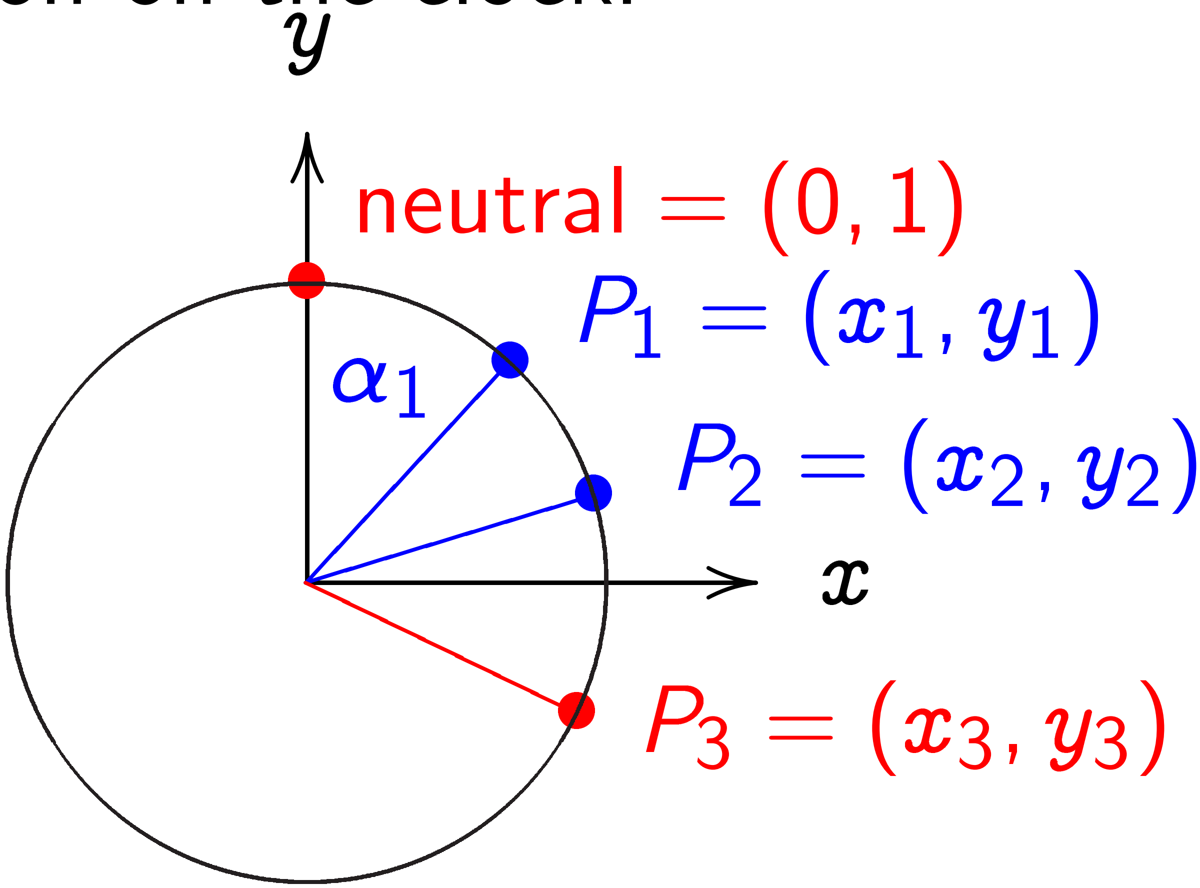
$(3/5, -4/5)$. $(-3/5, -4/5)$.

$(4/5, 3/5)$. $(-4/5, 3/5)$.

$(4/5, -3/5)$. $(-4/5, -3/5)$.

Many more.

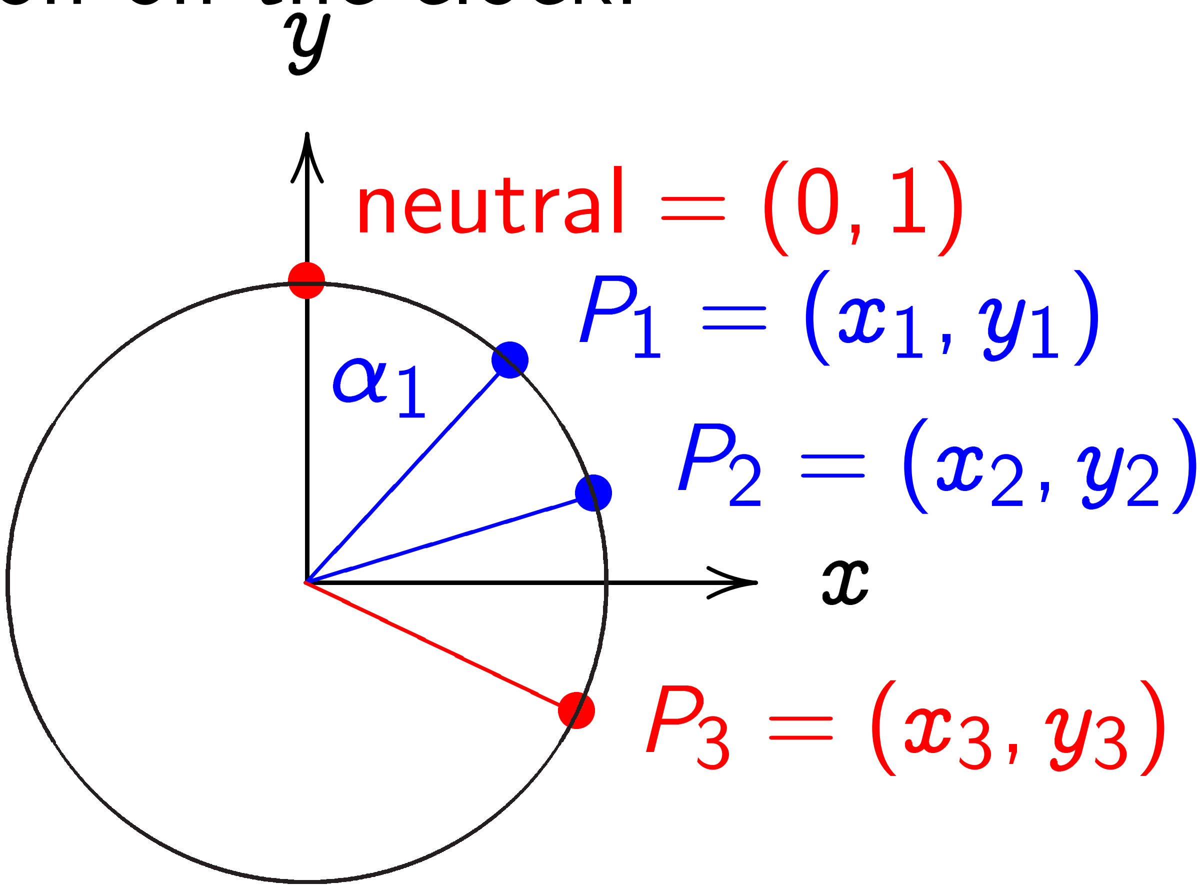
Addition on the clock:



$x^2 + y^2 = 1$, parametrized by

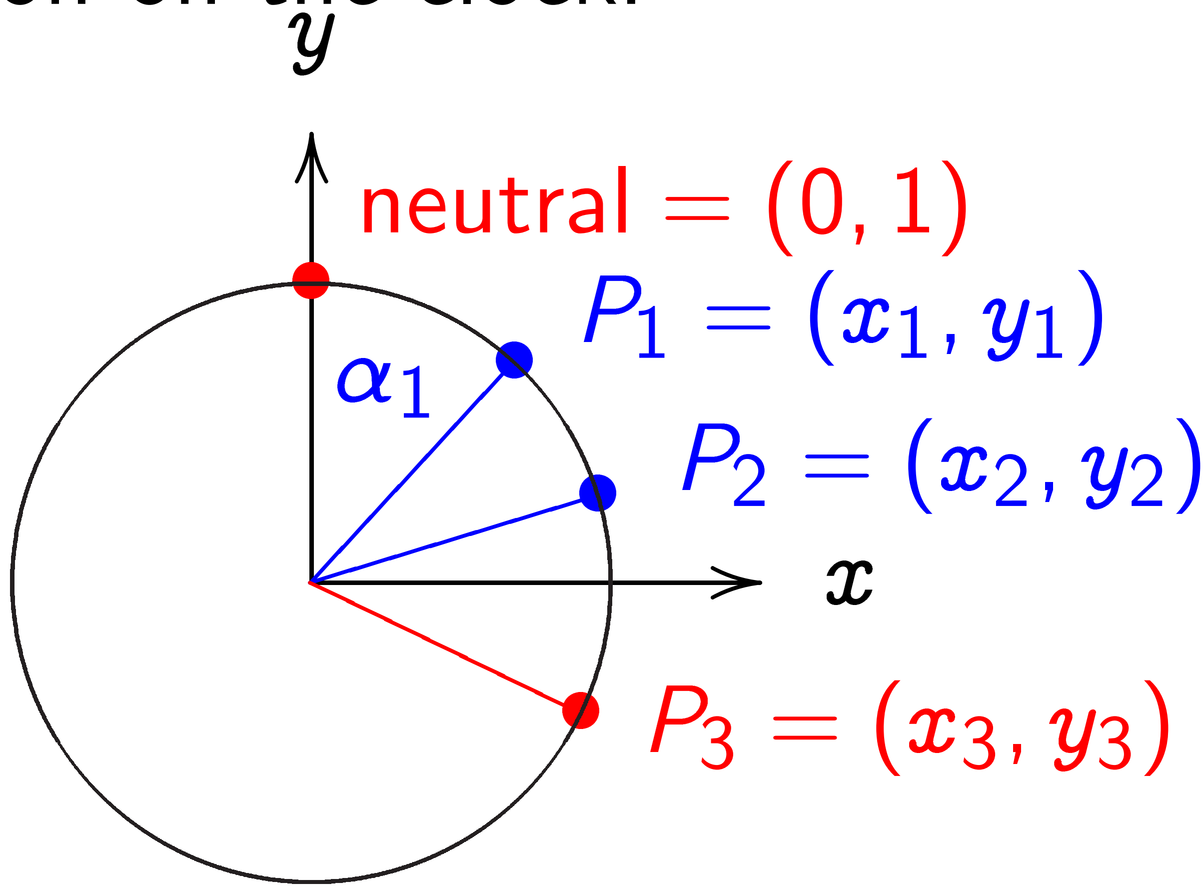
$x = \sin\alpha, \quad y = \cos\alpha.$

Addition on the clock:



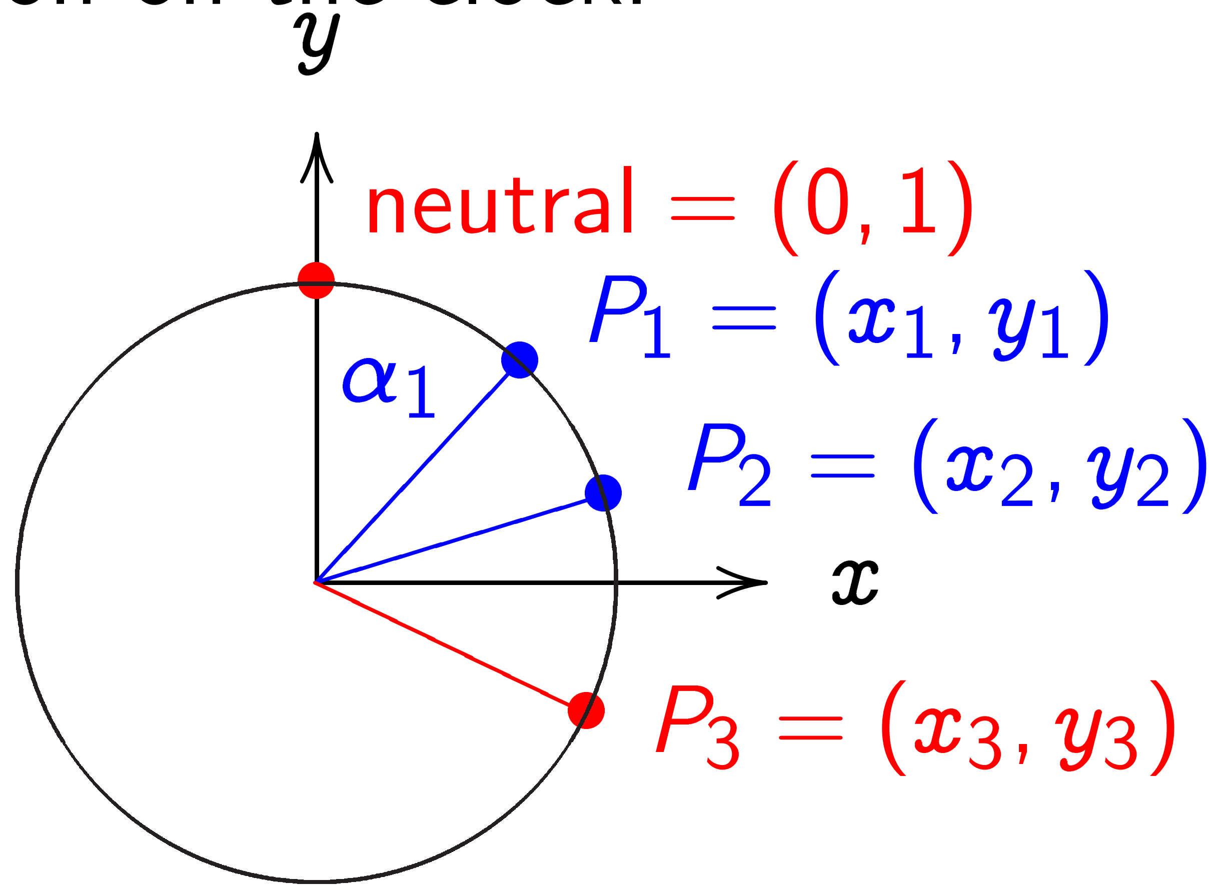
$x^2 + y^2 = 1$, parametrized by $x = \sin\alpha$, $\quad y = \cos\alpha$. Recall $(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$

Addition on the clock:



$x^2 + y^2 = 1$, parametrized by
$x = \sin\alpha, \quad y = \cos\alpha$. Recall
$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$

Addition on the clock:



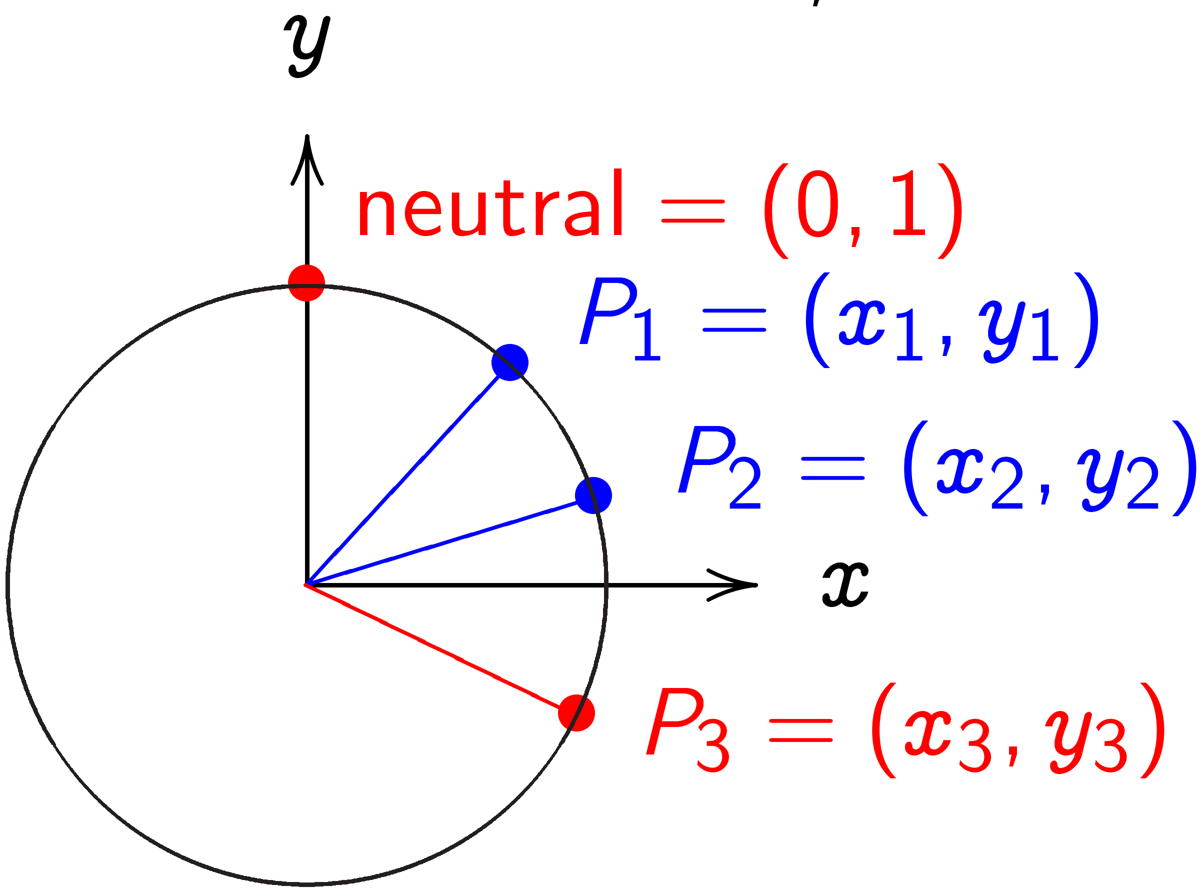
$x^2 + y^2 = 1$, parametrized by
$x = \sin\alpha, \quad y = \cos\alpha$. Recall

$$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$$
$$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$$
$$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2).$$

Adding two points corresponds to adding the angles α_1 and α_2. Angles modulo 360° are a group, so points on clock are a group.

Neutral element: angle $\alpha = 0$; point $(0, 1)$; "12:00".
The point with $\alpha = 180^\circ$ has order 2 and equals 6:00.
3:00 and 9:00 have order 4.
Inverse of point with α is point with $-\alpha$ since $\alpha + (-\alpha) = 0$.
There are many more points where angle α is not "nice."

Clock addition without sin, cos:



Use Cartesian coordinates for addition. Addition formula for the clock $x^2 + y^2 = 1$:

sum $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$

$= (x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)$.

Note $(x_1, y_1) + (-x_1, y_1) = (0, 1)$.

$kP = \underbrace{P + P + \cdots + P}_{k \text{ copies}}$ for $k \geq 0$.

Examples of clock addition:

"2:00" + "5:00"

$$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$$
$$= (-1/2, -\sqrt{3/4}) = \text{"7:00"}.$$

"5:00" + "9:00"

$$= (1/2, -\sqrt{3/4}) + (-1, 0)$$
$$= (\sqrt{3/4}, 1/2) = \text{"2:00"}.$$

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

Examples of clock addition:

"2:00" + "5:00"

$$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$$

$$= (-1/2, -\sqrt{3/4}) = \text{"7:00"}.$$

"5:00" + "9:00"

$$= (1/2, -\sqrt{3/4}) + (-1, 0)$$

$$= (\sqrt{3/4}, 1/2) = \text{"2:00"}.$$

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

Examples of clock addition:

$$\text{“2:00”} + \text{“5:00”}$$
$$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$$
$$= (-1/2, -\sqrt{3/4}) = \text{“7:00”}.$$

$$\text{“5:00”} + \text{“9:00”}$$
$$= (1/2, -\sqrt{3/4}) + (-1, 0)$$
$$= (\sqrt{3/4}, 1/2) = \text{“2:00”}.$$

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

Examples of clock addition:

$$\text{“2:00”} + \text{“5:00”}$$
$$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$$
$$= (-1/2, -\sqrt{3/4}) = \text{“7:00”}.$$

$$\text{“5:00”} + \text{“9:00”}$$
$$= (1/2, -\sqrt{3/4}) + (-1, 0)$$
$$= (\sqrt{3/4}, 1/2) = \text{“2:00”}.$$

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$$(x_1, y_1) + (0, 1) =$$

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$$(x_1, y_1) + (0, 1) = (x_1, y_1).$$

Examples of clock addition:

"2:00" + "5:00"
$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$
$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"
$= (1/2, -\sqrt{3/4}) + (-1, 0)$
$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$(x_1, y_1) + (0, 1) = (x_1, y_1)$.

$(x_1, y_1) + (-x_1, y_1) =$

Examples of clock addition:

$$\text{“2:00”} + \text{“5:00”}$$
$$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$$
$$= (-1/2, -\sqrt{3/4}) = \text{“7:00”}.$$

$$\text{“5:00”} + \text{“9:00”}$$
$$= (1/2, -\sqrt{3/4}) + (-1, 0)$$
$$= (\sqrt{3/4}, 1/2) = \text{“2:00”}.$$

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

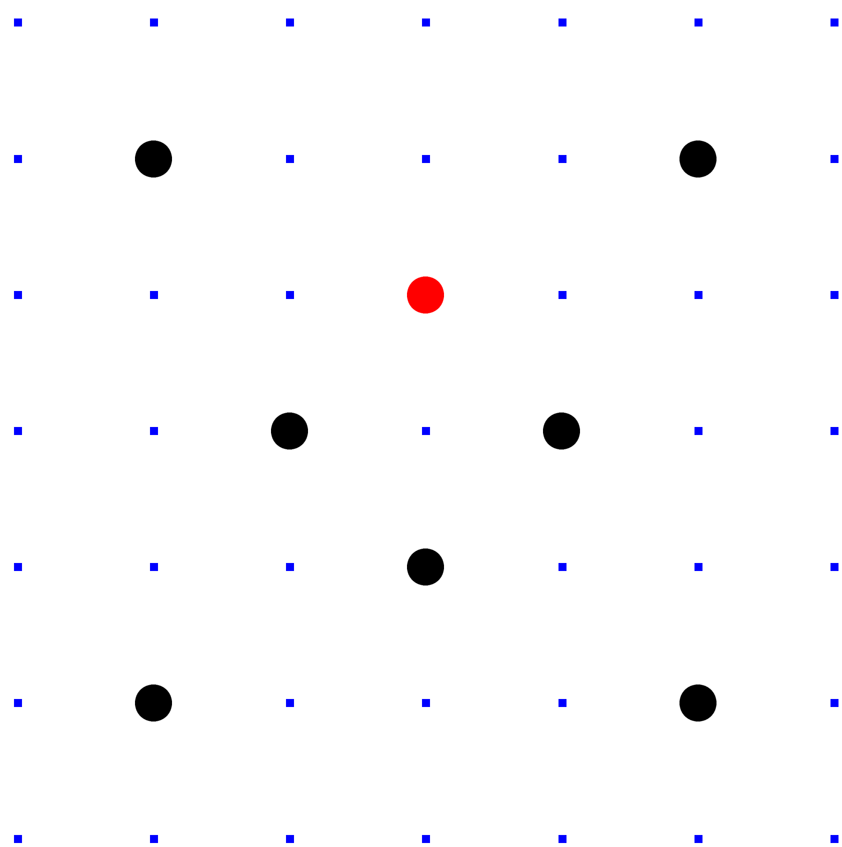
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$$(x_1, y_1) + (0, 1) = (x_1, y_1).$$

$$(x_1, y_1) + (-x_1, y_1) = (0, 1).$$

Clocks over finite fields



$\text{Clock}(\mathbf{F}_7) =$
$\{(x, y) \in \mathbf{F}_7 \times \mathbf{F}_7 : x^2 + y^2 = 1\}$.
Here $\mathbf{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$
$= \{0, 1, 2, 3, -3, -2, -1\}$
with $+, -, \times$ modulo 7.

Larger example: Clock($\mathbf{F}_{1000003}$).

Examples of clock addition:

$2(1000, 2) = (4000, 7)$.

$4(1000, 2) = (56000, 97)$.

$8(1000, 2) = (863970, 18817)$.

$16(1000, 2) = (549438, 156853)$.

$17(1000, 2) = (951405, 877356)$.

With 30 clock additions

we computed

$n(1000, 2) = (947472, 736284)$

for some 6-digit n.

Can you figure out n?

Clock cryptography

Standardize a large prime p
and some $(X, Y) \in \text{Clock}(\mathbf{F}_p)$.

Alice chooses big secret a.
Computes her public key $a(X, Y)$.

Bob chooses big secret b.
Computes his public key $b(X, Y)$.

Alice computes $a(b(X, Y))$.
Bob computes $b(a(X, Y))$.
I.e., both obtain $(ab)(X, Y)$.
They use this shared value
to encrypt with AES-GCM etc.

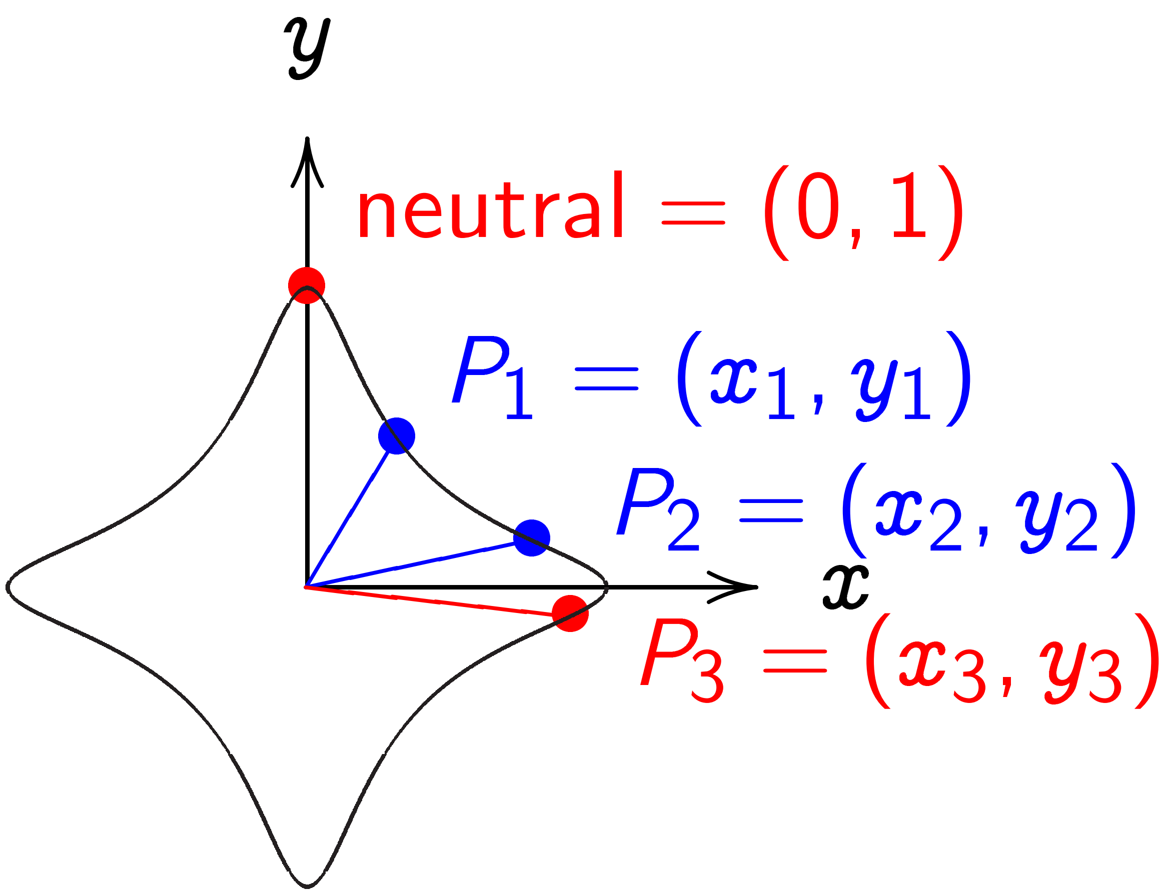
$$
\begin{tikzcd}
\substack{\text{Alice's} \\ \text{secret key } a} \arrow[d] \arrow[dd, bend right] & & \substack{\text{Bob's} \\ \text{secret key } b} \arrow[d] \arrow[dd, bend left] \\
\substack{\text{Alice's} \\ \text{public key} \\ a(X,Y)} \arrow[drr] & & \substack{\text{Bob's} \\ \text{public key} \\ b(X,Y)} \arrow[dll] \\
\substack{\{\text{Alice}, \text{Bob}\}\text{'s} \\ \text{shared secret} \\ ab(X,Y)} & = & \substack{\{\text{Bob}, \text{Alice}\}\text{'s} \\ \text{shared secret} \\ ba(X,Y)}
\end{tikzcd}
$$

$$
\begin{tikzcd}
\text{Alice's secret key } a \arrow[d] \arrow[dd, bend right] & & \text{Bob's secret key } b \arrow[d] \arrow[dd, bend left] \\
\text{Alice's public key } a(X,Y) \arrow[drr] & & \text{Bob's public key } b(X,Y) \arrow[dll] \\
\{\text{Alice}, \text{Bob}\}\text{'s shared secret } ab(X,Y) & = & \{\text{Bob}, \text{Alice}\}\text{'s shared secret } ba(X,Y)
\end{tikzcd}
$$

Warning: Clocks aren't elliptic!
Can attack clock cryptography, e.g., compute a from public key, by combining congruences.
To match RSA-3072 security need $p \approx 2^{1536}$.

Addition on an Edwards curve

Change the curve on which Alice and Bob work.

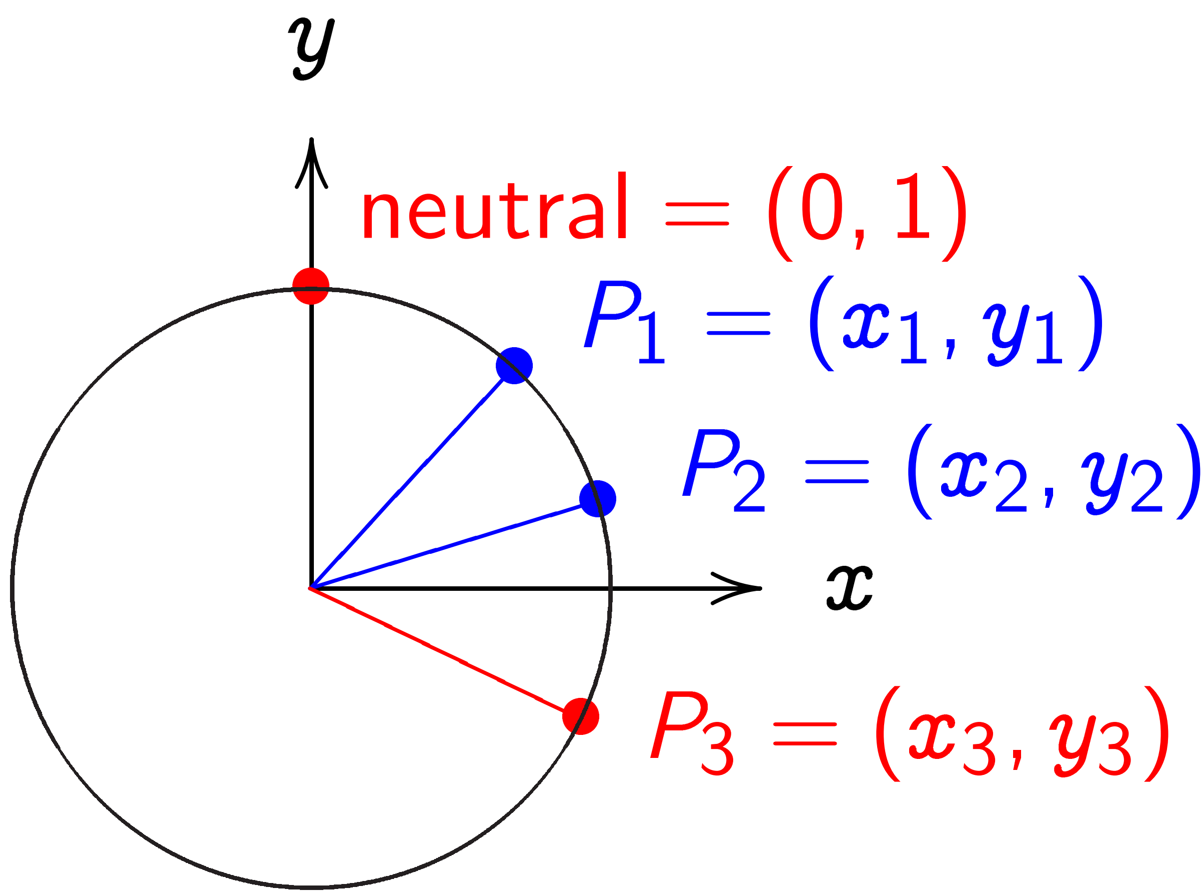


$x^2 + y^2 = 1 - 30x^2y^2$.

Sum of (x_1, y_1) and (x_2, y_2) is

$$((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2),$$
$$(y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2)).$$

The clock again, for comparison:



$x^2 + y^2 = 1$.

Sum of (x_1, y_1) and (x_2, y_2) is

$(x_1 y_2 + y_1 x_2,$

$y_1 y_2 - x_1 x_2)$.

"Hey, there were divisions
in the Edwards addition law!
What if the denominators are 0?"

Answer: They aren't!

If $x_i = 0$ or $y_i = 0$ then
$1 \pm 30 x_1 x_2 y_1 y_2 = 1 \neq 0$.

If $x^2 + y^2 = 1 - 30x^2y^2$
then $30x^2y^2 < 1$
so $\sqrt{30}\,|xy| < 1$.

"Hey, there were divisions
in the Edwards addition law!
What if the denominators are 0?"

Answer: They aren't!

If $x_i = 0$ or $y_i = 0$ then
$1 \pm 30x_1x_2y_1y_2 = 1 \neq 0$.

If $x^2 + y^2 = 1 - 30x^2y^2$
then $30x^2y^2 < 1$
so $\sqrt{30}\,|xy| < 1$.

If $x_1^2 + y_1^2 = 1 - 30x_1^2y_1^2$
and $x_2^2 + y_2^2 = 1 - 30x_2^2y_2^2$
then $\sqrt{30}\,|x_1y_1| < 1$
and $\sqrt{30}\,|x_2y_2| < 1$

"Hey, there were divisions
in the Edwards addition law!
What if the denominators are 0?"

Answer: They aren't!

If $x_i = 0$ or $y_i = 0$ then
$1 \pm 30x_1x_2y_1y_2 = 1 \neq 0$.

If $x^2 + y^2 = 1 - 30x^2y^2$
then $30x^2y^2 < 1$
so $\sqrt{30}\,|xy| < 1$.

If $x_1^2 + y_1^2 = 1 - 30x_1^2y_1^2$
and $x_2^2 + y_2^2 = 1 - 30x_2^2y_2^2$
then $\sqrt{30}\,|x_1y_1| < 1$
and $\sqrt{30}\,|x_2y_2| < 1$
so $30\,|x_1y_1x_2y_2| < 1$
so $1 \pm 30x_1x_2y_1y_2 > 0$.

The Edwards addition law

$(x_1, y_1) + (x_2, y_2) =$
$((x_1 y_2 + y_1 x_2)/(1 - 30 x_1 x_2 y_1 y_2),$
$(y_1 y_2 - x_1 x_2)/(1 + 30 x_1 x_2 y_1 y_2))$

is a group law for the curve $x^2 + y^2 = 1 - 30x^2y^2$.

Some calculation required:
addition result is on curve;
addition law is associative.

Other parts of proof are easy:
addition law is commutative;
$(0, 1)$ is neutral element;
$(x_1, y_1) + (-x_1, y_1) = (0, 1)$.

More Edwards curves

Fix an odd prime power q.
Fix a *non-square* $d \in \mathbf{F}_q$.

$$\{(x, y) \in \mathbf{F}_q \times \mathbf{F}_q : \\ x^2 + y^2 = 1 + dx^2y^2\}$$

is a commutative group with $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ defined by Edwards addition law:

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2},$$

$$y_3 = \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2}.$$

Denominators are never 0.

But need different proof;

"$x^2 + y^2 > 0$" doesn't work in $\mathbf{F}_q$.

Denominators are never 0.

But need different proof;

"$x^2 + y^2 > 0$" doesn't work in $\mathbf{F}_q$.

If $x_1^2 + y_1^2 = 1 + dx_1^2 y_1^2$
and $x_2^2 + y_2^2 = 1 + dx_2^2 y_2^2$
and $dx_1 x_2 y_1 y_2 = \pm 1$

Denominators are never 0.

But need different proof;

"$x^2 + y^2 > 0$" doesn't work in $\mathbf{F}_q$.

If $x_1^2 + y_1^2 = 1 + dx_1^2 y_1^2$

and $x_2^2 + y_2^2 = 1 + dx_2^2 y_2^2$

and $dx_1 x_2 y_1 y_2 = \pm 1$

then $dx_1^2 y_1^2 (x_2 + y_2)^2$

$= dx_1^2 y_1^2 (x_2^2 + y_2^2 + 2x_2 y_2)$

Denominators are never 0.

But need different proof;

"$x^2 + y^2 > 0$" doesn't work in $\mathbf{F}_q$.

If $x_1^2 + y_1^2 = 1 + dx_1^2 y_1^2$
and $x_2^2 + y_2^2 = 1 + dx_2^2 y_2^2$
and $dx_1 x_2 y_1 y_2 = \pm 1$
then $dx_1^2 y_1^2 (x_2 + y_2)^2$
$= dx_1^2 y_1^2 (x_2^2 + y_2^2 + 2x_2 y_2)$
$= dx_1^2 y_1^2 (dx_2^2 y_2^2 + 1 + 2x_2 y_2)$

Denominators are never 0.

But need different proof;

"$x^2 + y^2 > 0$" doesn't work in $\mathbf{F}_q$.

If $x_1^2 + y_1^2 = 1 + dx_1^2 y_1^2$
and $x_2^2 + y_2^2 = 1 + dx_2^2 y_2^2$
and $dx_1 x_2 y_1 y_2 = \pm 1$
then $dx_1^2 y_1^2 (x_2 + y_2)^2$
$= dx_1^2 y_1^2 (x_2^2 + y_2^2 + 2x_2 y_2)$
$= dx_1^2 y_1^2 (dx_2^2 y_2^2 + 1 + 2x_2 y_2)$
$= d^2 x_1^2 y_1^2 x_2^2 y_2^2 + dx_1^2 y_1^2 + 2dx_1^2 y_1^2 x_2 y_2$

Denominators are never 0.

But need different proof;
"$x^2 + y^2 > 0$" doesn't work in $\mathbf{F}_q$.

If $x_1^2 + y_1^2 = 1 + dx_1^2 y_1^2$
and $x_2^2 + y_2^2 = 1 + dx_2^2 y_2^2$
and $dx_1 x_2 y_1 y_2 = \pm 1$
then $dx_1^2 y_1^2 (x_2 + y_2)^2$
$= dx_1^2 y_1^2 (x_2^2 + y_2^2 + 2x_2 y_2)$
$= dx_1^2 y_1^2 (dx_2^2 y_2^2 + 1 + 2x_2 y_2)$
$= d^2 x_1^2 y_1^2 x_2^2 y_2^2 + dx_1^2 y_1^2 + 2dx_1^2 y_1^2 x_2 y_2$
$= 1 + dx_1^2 y_1^2 \pm 2x_1 y_1$

Denominators are never 0.

But need different proof;

"$x^2 + y^2 > 0$" doesn't work in $\mathbf{F}_q$.

If $x_1^2 + y_1^2 = 1 + dx_1^2y_1^2$
and $x_2^2 + y_2^2 = 1 + dx_2^2y_2^2$
and $dx_1x_2y_1y_2 = \pm 1$
then $dx_1^2y_1^2(x_2 + y_2)^2$
$= dx_1^2y_1^2(x_2^2 + y_2^2 + 2x_2y_2)$
$= dx_1^2y_1^2(dx_2^2y_2^2 + 1 + 2x_2y_2)$
$= d^2x_1^2y_1^2x_2^2y_2^2 + dx_1^2y_1^2 + 2dx_1^2y_1^2x_2y_2$
$= 1 + dx_1^2y_1^2 \pm 2x_1y_1$
$= x_1^2 + y_1^2 \pm 2x_1y_1$

Denominators are never 0.

But need different proof;

"$x^2 + y^2 > 0$" doesn't work in $\mathbf{F}_q$.

If $x_1^2 + y_1^2 = 1 + dx_1^2 y_1^2$
and $x_2^2 + y_2^2 = 1 + dx_2^2 y_2^2$
and $dx_1 x_2 y_1 y_2 = \pm 1$
then $dx_1^2 y_1^2 (x_2 + y_2)^2$
$= dx_1^2 y_1^2 (x_2^2 + y_2^2 + 2x_2 y_2)$
$= dx_1^2 y_1^2 (dx_2^2 y_2^2 + 1 + 2x_2 y_2)$
$= d^2 x_1^2 y_1^2 x_2^2 y_2^2 + dx_1^2 y_1^2 + 2dx_1^2 y_1^2 x_2 y_2$
$= 1 + dx_1^2 y_1^2 \pm 2x_1 y_1$
$= x_1^2 + y_1^2 \pm 2x_1 y_1$
$= (x_1 \pm y_1)^2$.

Case 1: $x_2 + y_2 \neq 0$. Then

$$d = \left(\frac{x_1 \pm y_1}{x_1 y_1 (x_2 + y_2)} \right)^2,$$

contradiction.

Case 1: $x_2 + y_2 \neq 0$. Then

$$d = \left(\frac{x_1 \pm y_1}{x_1 y_1 (x_2 + y_2)} \right)^2,$$

contradiction.

Case 2: $x_2 - y_2 \neq 0$. Then

$$d = \left(\frac{x_1 \mp y_1}{x_1 y_1 (x_2 - y_2)} \right)^2,$$

contradiction.

Case 1: $x_2 + y_2 \neq 0$. Then

$$d = \left(\frac{x_1 \pm y_1}{x_1 y_1 (x_2 + y_2)} \right)^2,$$

contradiction.

Case 2: $x_2 - y_2 \neq 0$. Then

$$d = \left(\frac{x_1 \mp y_1}{x_1 y_1 (x_2 - y_2)} \right)^2,$$

contradiction.

Case 3: $x_2 + y_2 = x_2 - y_2 = 0$.
Then $x_2 = 0$ and $y_2 = 0$,
contradiction.

Using ECC sensibly

Typical starting point:
Client knows secret key a
and server's public key $b(X, Y)$.
Client computes (and caches)
shared secret $ab(X, Y)$.

Client has packet for server.
Generates unique nonce.
Uses shared secret to encrypt
and authenticate packet.

Total packet overhead:
24 bytes for nonce,
16 bytes for authenticator,
32 bytes for client's public key.

Server receives packet,
sees client's public key $a(X, Y)$.
Server computes (and caches)
shared secret $ab(X, Y)$.

Server uses shared secret
to verify authenticator
and decrypt packet.

Client and server encrypt,
authenticate, verify, and decrypt
all subsequent packets
in the same way,
using the same shared secret.

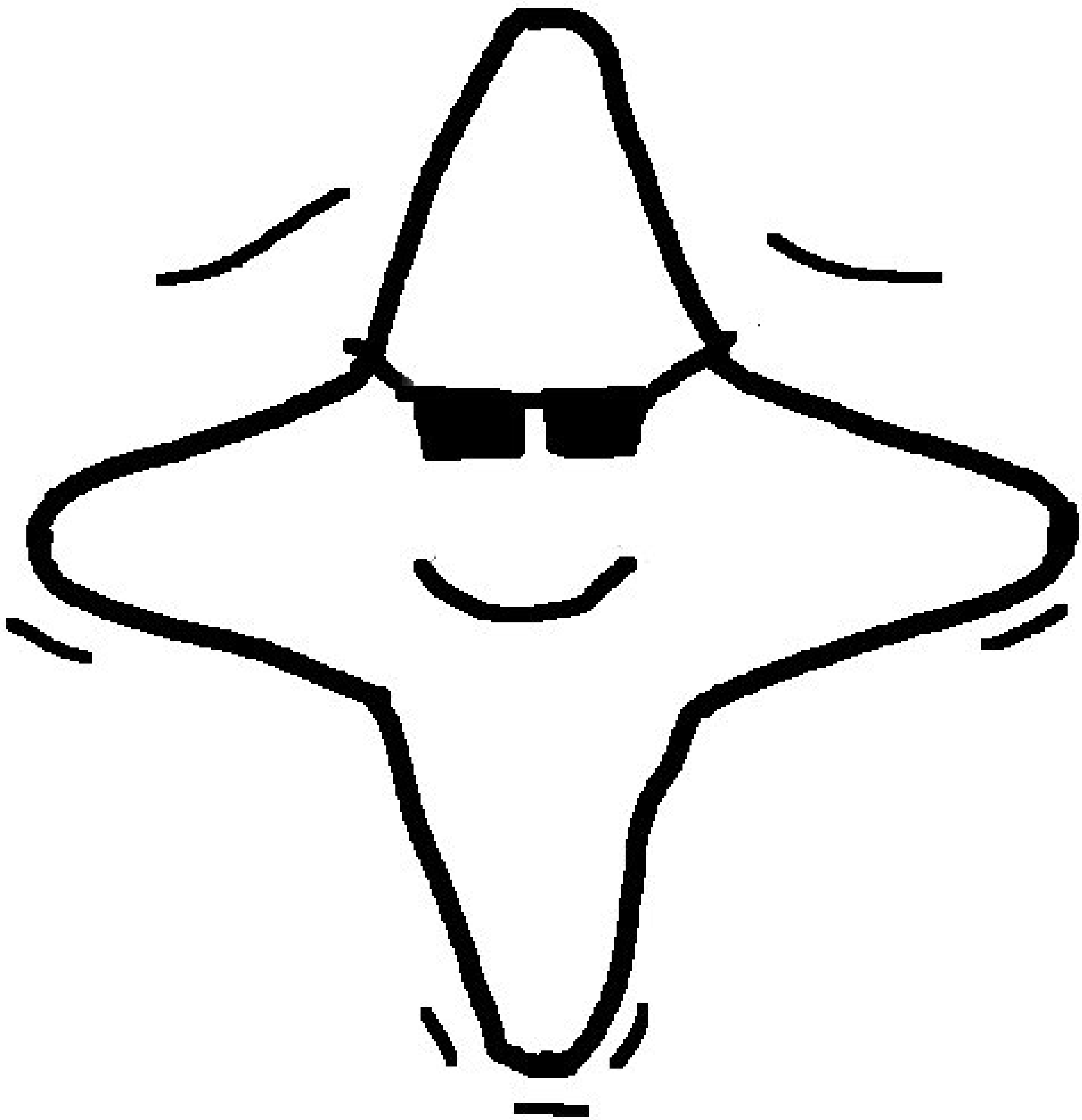
Easy-to-use packet protection:
`crypto_box` from
`nacl.cace-project.eu`.

High-security curve (Curve25519).
High-security implementation
(e.g., no secret array indices).
Extensive code validation.

Server can compute shared secrets
for 1000000 new clients
in 40 seconds of computation
on a Core 2 Quad.

Not much hope for attacker
if ECC user is running this!

Edwards curves are cool



How to compute aP

Use binary representation of a
to compute $a(X, Y)$
in $\lfloor \log_2 a \rfloor$ doublings
and at most that many additions.

E.g. $a = 23 = (10111)_2$:
$23P = 2(2(2(2P) + P) + P) + P$.
For $a = (1, a_{n-1}, \ldots, a_1, a_0)_2$;
compute *scalar multiplication*
$aP = 2(\cdots 2(2(2P + a_{n-1}P) + a_{n-2}P) + \cdots + a_1P) + a_0P$.

There exist other, more efficient methods.

Faster group operations

Can split computation aP into additions and doublings.

Formulas $(x_1, y_1) + (x_2, y_2) =$
$((x_1y_2+y_1x_2)/(1 + dx_1x_2y_1y_2),$
$(y_1y_2-x_1x_2)/(1 - dx_1x_2y_1y_2))$
use expensive divisions.

Better: postpone divisions and work with fractions.

Represent (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 $Z_1 \neq 0$.

Addition formulas in these *projective* coordinates:

$A = Z_1 \cdot Z_2$; $B = A^2$;
$C = X_1 \cdot X_2$; $D = Y_1 \cdot Y_2$;
$E = d \cdot C \cdot D$; $F = B - E$;
$G = B + E$; $X_3 = A \cdot F \cdot ((X_1 + Y_1) \cdot (X_2 + Y_2) - C - D)$;
$Y_3 = A \cdot G \cdot (D - C)$; $Z_3 = F \cdot G$.

Needs $1\mathbf{S}+10\mathbf{M}+1\mathbf{M}_d$.

Uses
$(X_1 + Y_1) \cdot (X_2 + Y_2) - C - D = X_1X_2 + X_1Y_2 + Y_1X_2 + Y_1Y_2 - X_1X_2 - Y_1Y_2 = X_1Y_2 + Y_1X_2$.

Doubling means $P_2 = P_1$, i.e.,

$$(x_1, y_1) + (x_1, y_1) =$$
$$((x_1y_1+y_1x_1)/(1 + dx_1x_1y_1y_1),$$
$$(y_1y_1-x_1x_1)/(1-dx_1x_1y_1y_1)) =$$
$$((2x_1y_1)/(1 + dx_1^2y_1^2),$$
$$(y_1^2-x_1^2)/(1 - dx_1^2y_1^2)).$$

Remember $P_1 = (x_1, y_1)$ is a point on the Edwards curve, thus $x_1^2 + y_1^2 = 1 + dx_1^2y_1^2$ and

$$2P_1 = ((2x_1y_1)/(x_1^2 + y_1^2),$$
$$(y_1^2-x_1^2)/(2 - (x_1^2 + y_1^2))).$$

This transformation reduced the total degree of the equation from 4 to 2.

Doubling formulas in projective coordinates:

$B = (X_1 + Y_1)^2;\; C = X_1^2;$
$D = Y_1^2;\; E = C + D;\; H = Z_1^2;$
$J = E - 2H;$
$X_3 = (B - E) \cdot J;$
$Y_3 = E \cdot (C - D);\; Z_3 = E \cdot J.$

Needs 4**S**+3**M**.

Doubling formulas in projective coordinates:

$B = (X_1 + Y_1)^2;\ C = X_1^2;$
$D = Y_1^2;\ E = C + D;\ H = Z_1^2;$
$J = E - 2H;$
$X_3 = (B - E) \cdot J;$
$Y_3 = E \cdot (C - D);\ Z_3 = E \cdot J.$

Needs 4**S**+3**M**.

1 doubling far less expensive than 1 addition.
Usual scalar multiplication uses many more doublings than additions.

More variations of addition:

e.g., in inverted coordinates $(X_1 : Y_1 : Z_1)$ corresponds to $x_1 = Z_1/X_1$ and $y_1 = Z_1/Y_1$.

Alternative addition formulas:

$$(x_1, y_1) + (x_2, y_2) = ((x_1y_1 + x_2y_2)/(x_1x_2 + y_1y_2),\ (x_1y_1 - x_2y_2)/(x_1y_2 - x_2y_1)).$$

Attention: these formulas fail for doubling.

Curious fact: formulas do not involve curve parameter d.

Twisted Edwards curves

Generalization to cover more curves over given finite field $\mathbf{F}_q$: Use $a, d \in \mathbf{F}_q^*$ with $a \neq d$ and consider twisted Edwards curve $ax^2 + y^2 = 1 + dx^2y^2$.

Why do users want more curves? One answer: Speed! Particularly fast choice: $a = -1$ gives additions in $8\mathbf{M}$.

ECDSA

Users can sign messages using Edwards curves.

Take a point P on an Edwards curve modulo a prime $q > 2$.

ECDSA signer needs to know the *order of* P.

There are only finitely many other points; about q in total.
Adding P to itself will eventually reach $(0, 1)$; let ℓ be the smallest integer > 0 with $\ell P = (0, 1)$.
This ℓ is the order of P.

The signature scheme has as system parameters a curve E; a base point P; and a hash function h with output length at least $\lfloor \log_2 \ell \rfloor + 1$.
Alice's secret key is an integer a and her public key is $P_A = aP$.

To sign message m,
Alice computes $h(m)$;
picks random k;
computes $R = kP = (x_1, y_1)$;
puts $r \equiv y_1 \bmod \ell$; computes $s \equiv k^{-1}(h(m) + r \cdot a) \bmod \ell$.

The signature on m is (r, s).

Anybody can verify signature given m and (r, s):
Compute $w_1 \equiv s^{-1}h(m) \bmod \ell$ and $w_2 \equiv s^{-1} \cdot r \bmod \ell$.
Check whether the y-coordinate of $w_1 P + w_2 P_A$ equals r modulo ℓ and if so, accept signature.

Alice's signatures are valid:

$$
\begin{aligned}
w_1 P + w_2 P_A &= \\
&(s^{-1}h(m))P + (s^{-1} \cdot r)P_A = \\
&(s^{-1}(h(m) + ra))P = kP
\end{aligned}
$$

and so the y-coordinate of this expression equals r,
the y-coordinate of kP.

Attacker's view on signatures

Anybody can produce an $R = kP$.
Alice's private key is only used in
$s \equiv k^{-1}(h(m) + r \cdot a) \bmod \ell$.

Can fake signatures if one can break the DLP, i.e., if one can compute a from P_A.
But ECC is attractive because the DLP is hard.
Sometimes attacks are easier...

If k is known for some $m, (r, s)$ then $a \equiv (sk - h(m))/r \bmod \ell$.

If two signatures $m_1, (r, s_1)$ and $m_2, (r, s_2)$ have the same value for r: assume $k_1 = k_2$; observe $s_1 - s_2 = k_1^{-1}(h(m_1) + ra - (h(m_2) + ra))$; compute $k = (s_1 - s_2)/(h(m_1) - h(m_2))$. Continue as above.

If bits of many k's are known (biased PRNG) can attack $s \equiv k^{-1}(h(m) + r \cdot a) \bmod \ell$ as hidden number problem using lattice basis reduction.

Malicious signer

Alice can set up her public key so that two messages of her choice share the same signature,
i.e., she can claim to have signed m_1 or m_2 at will:
$R = (x_1, y_1)$ and $-R = (-x_1, y_1)$ have the same y-coordinate.
Thus, (r, s) fits $R = kP$,
$s \equiv k^{-1}(h(m_1) + ra) \bmod \ell$ and
$-R = (-k)P$,
$s \equiv -k^{-1}(h(m_2) + ra) \bmod \ell$ if
$a \equiv -(h(m_1)+h(m_2))/2r \bmod \ell$.

Malicious signer

Alice can set up her public key so that two messages of her choice share the same signature,
i.e., she can claim to have signed m_1 or m_2 at will:
$R = (x_1, y_1)$ and $-R = (-x_1, y_1)$ have the same y-coordinate.
Thus, (r, s) fits $R = kP$,
$s \equiv k^{-1}(h(m_1) + ra) \bmod \ell$ and
$-R = (-k)P$,
$s \equiv -k^{-1}(h(m_2) + ra) \bmod \ell$ if
$a \equiv -(h(m_1)+h(m_2))/2r \bmod \ell$.

(Easy tweak: include bit of x_1.)

Elliptic curves

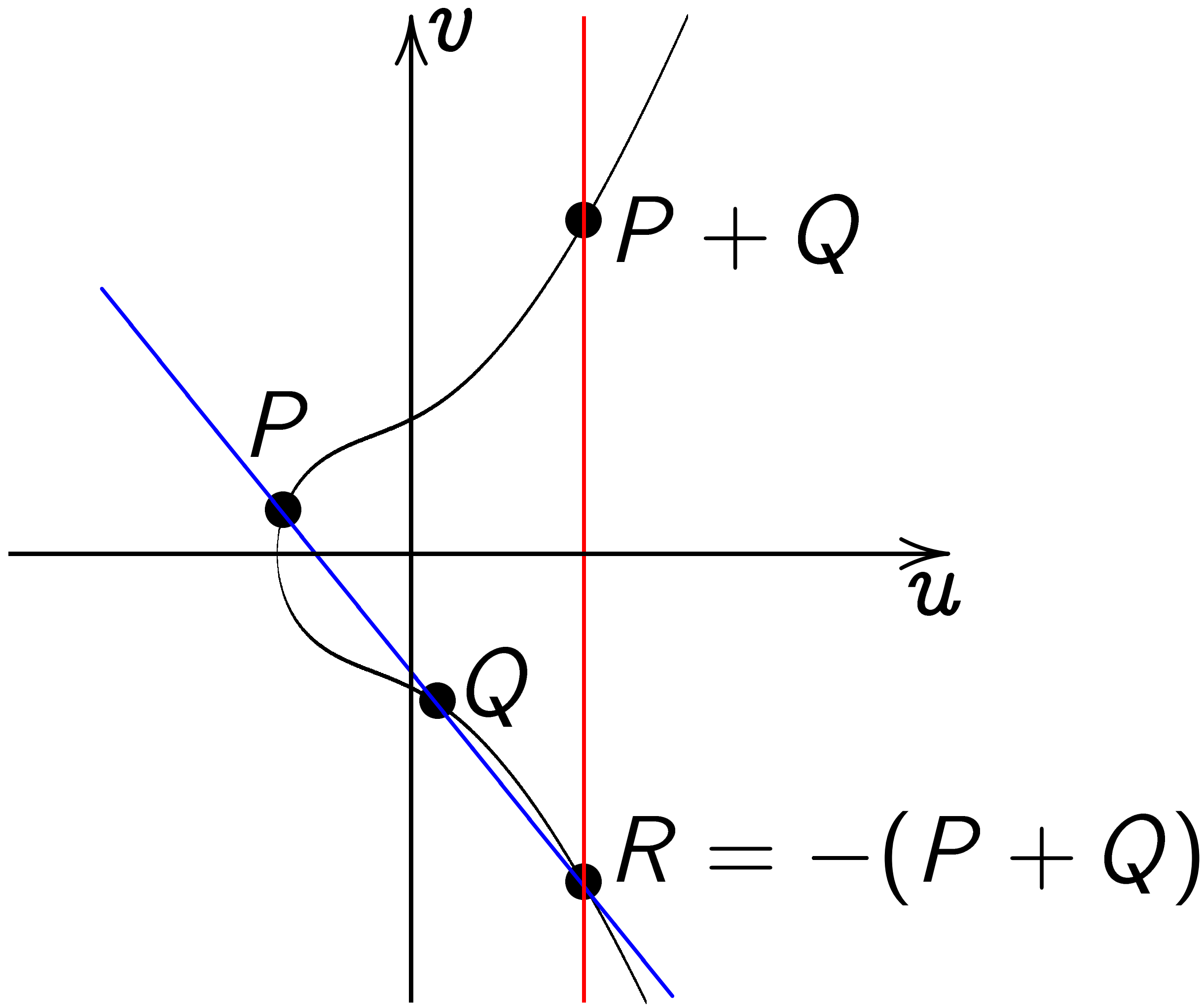
Why do we talk about
Edwards curves in this tutorial?
Edwards curves are elliptic;
easiest way to understand
elliptic curves is Edwards.

But more elliptic curves exist!
Most common representation:
Weierstrass curve
$v^2 = u^3 + a_2u^2 + a_4u + a_6$.

(Weierstrass has different
form in characteristic 2 or 3.)

Addition on Weierstrass curve

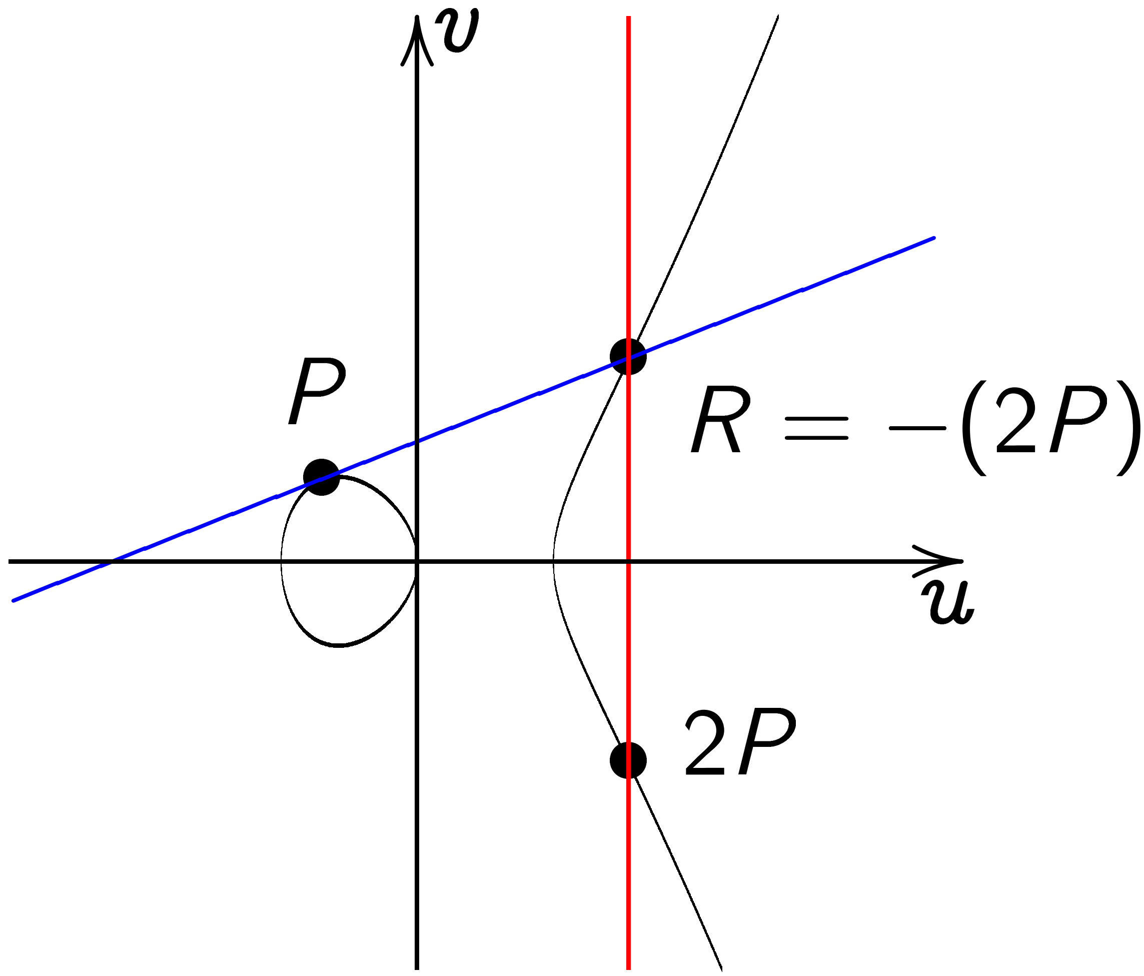
$$v^2 = u^3 + u^2 + u + 1$$



Slope $\lambda = (v_2 - v_1)/(u_2 - u_1)$
(note $u_1 \neq u_2$).

Doubling on Weierstrass curve

$v^2 = u^3 - u$



Slope $\lambda = (3u_1^2 - 1)/(2v_1)$.

General addition:

$(u_P, v_P) + (u_R, v_R) =$
$(u_{P+R}, v_{P+R}) =$
$(\lambda^2 - u_P - u_R, \lambda(u_P - u_{P+R}) - v_P)$.

$u_P \neq u_R$, "addition":
$\lambda = (v_R - v_P)/(u_R - u_P)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$.

$P = R$ and $v_P \neq 0$, "doubling":
$\lambda = (3u_P^2 + 2a_2u_P + a_4)/(2v_P)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

Also handle some exceptions:
$(u_P, v_P) = (u_R, -v_R)$;
inputs at ∞.

Birational equivalence

Starting from point (x, y)
on $x^2 + y^2 = 1 + dx^2y^2$:

Define $A = 2(1 + d)/(1 - d)$,
$B = 4/(1 - d)$;
$u = (1 + y)/(B(1 - y))$,
$v = u/x = (1 + y)/(Bx(1 - y))$.
(Skip a few exceptional points.)

Then (u, v) is a point on
a Weierstrass curve:
$v^2 = u^3 + (A/B)u^2 + (1/B^2)u$.

Easily invert this map:
$x = u/v$, $y = (Bu - 1)/(Bu + 1)$.

Attacker can transform Edwards curve to Weierstrass curve and vice versa; $n(x, y) \mapsto n(u, v)$.
$\Rightarrow$ Same discrete-log security!
Can choose curve representation so that implementation of attack is faster/easier.

System designer can choose curve representation so that protocol runs fastest; no need to worry about security degradation.

Faster group operations

Designer has choice of
curve representation and
point representation.

Most protocols do
a full scalar multiplication,
many doublings and additions,
before they need a
unique representative.

Can double and add using
inversion-free systems such as
projective Edwards coordinates.
Faster, but more storage.

Montgomery curves

1987 Montgomery:

Use $by^2 = x^3 + ax^2 + x$.
Choose small $(a+2)/4$.

$2(x_2, y_2) = (x_4, y_4)$
$$\Rightarrow x_4 = \frac{(x_2^2 - 1)^2}{4x_2(x_2^2 + ax_2 + 1)}.$$

$(x_3, y_3) - (x_2, y_2) = (x_1, y_1)$,
$(x_3, y_3) + (x_2, y_2) = (x_5, y_5)$
$$\Rightarrow x_5 = \frac{(x_2 x_3 - 1)^2}{x_1(x_2 - x_3)^2}.$$

Represent (x, y)
as $(X{:}Z)$ satisfying $x = X/Z$.

$B = (X_2 + Z_2)^2$,
$C = (X_2 - Z_2)^2$,
$D = B - C$, $X_4 = B \cdot C$,
$Z_4 = D \cdot (C + D(a + 2)/4) \Rightarrow$
$2(X_2{:}Z_2) = (X_4{:}Z_4)$.

$(X_3{:}Z_3) - (X_2{:}Z_2) = (X_1{:}Z_1)$,
$E = (X_3 - Z_3) \cdot (X_2 + Z_2)$,
$F = (X_3 + Z_3) \cdot (X_2 - Z_2)$,
$X_5 = Z_1 \cdot (E + F)^2$,
$Z_5 = X_1 \cdot (E - F)^2 \Rightarrow$
$(X_3{:}Z_3) + (X_2{:}Z_2) = (X_5{:}Z_5)$.

This representation does not allow ADD but it allows DADD, "differential addition": $Q, R, Q - R \mapsto Q + R$.

e.g. $2P, P, P \mapsto 3P$.
e.g. $3P, 2P, P \mapsto 5P$.
e.g. $6P, 5P, P \mapsto 11P$.

$2\mathbf{M} + 2\mathbf{S} + 1\mathbf{D}$ for DBL.
$4\mathbf{M} + 2\mathbf{S}$ for DADD.
Save $1\mathbf{M}$ if $Z_1 = 1$.

Easily compute $n(X_1 : Z_1)$ using $\approx \lg n$ DBL, $\approx \lg n$ DADD.
Almost as fast as Edwards nP.
Relatively slow for $mP + nQ$ etc.

More addition formulas

Explicit-Formulas Database:
hyperelliptic.org/EFD

EFD has 581 computer-verified formulas and operation counts for ADD, DBL, etc.
in 51 representations
on 13 shapes of elliptic curves.

Not yet handled by computer:
generality of curve shapes (e.g., Hessian order $\in 3\mathbf{Z}$);
complete addition algorithms (e.g., checking for ∞).

Notation

If
$$n = \sum_{i=0}^{l-1} n_i 2^i$$

we write n in binary representation
$$n = (n_{l-1} \ldots n_0)_2.$$
E.g. $n = 35 = 32+2+1 = 1\cdot 2^5 + 0\cdot 2^4 + 0\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0$, then $35 = (100011)_2$.

The following algorithms are stated in some group $(G, +)$ with neutral element O. Scalar multiplication is denoted by $nP = P + P + \cdots + P$ (n terms).

Right–to–Left Binary

`IN`: An element $P \in G$ and a positive integer $n = (n_{l-1}\dots n_0)_2$.

OUT: The element $nP \in G$.

$R \leftarrow O$, $Q \leftarrow P$,

$\texttt{for } i = 0 \texttt{ to } l-2 \texttt{ do}$

$\quad \texttt{if } n_i = 1 \texttt{ then } R \leftarrow P + Q$

$\quad Q \leftarrow 2Q$

$\texttt{if } n_{l-1} = 1 \texttt{ then } R \leftarrow P + Q$

$\texttt{return } R$

This algorithm computes $35P = 2^5P + 2^1P + P$.

At the end of step 2, $Q = 2^{i+1}P$ and $R = (n_i \dots n_0)_2 P$.

Left–to–Right Binary

`IN`: An element $P \in G$ and a positive integer $n = (n_{l-1} \dots n_0)_2, n_{l-1} = 1$.
OUT: The element $nP \in G$.

$R \leftarrow P$

`for` $i = l - 2$ `to` 0 `do`

$\quad R \leftarrow 2R$

$\quad$ `if` $n_i = 1$ `then` $R \leftarrow P + R$

`return` R

This algorithm computes $35P = 2(2(2(2(2P))) + P) + P$.
The intermediate variable R holds $(n_{l-1} \dots n_i)_2 P$.

Number of additions

For each 1 in the binary representation of n we compute an addition. On average there are $l/2$ non-zero coefficients.

In some groups (e.g. elliptic curves) $P+Q$ has the same cost as $P-Q$, so it makes sense to use negative coefficients. This gives signed binary expansions.

Note that $31 = 2^4 + 2^3 + 2^2 + 2 + 1 = 2^5 - 1$ and so $31P = 2(2(2(2P+P)+P)+P)+P)$

$$= 2(2(2(2(2P))))) - P$$

Can always replace two adjacent 1's in the binary expansion by $10\bar{1}$ since $(11)_2 = (10\bar{1})_s$.
($\bar{1}$ denotes -1).

By systematically replacing runs of 1's we can guarantee that there are no two adjacent bits that are nonzero.

A representation fulfilling this is called a "non-adjacent form" (NAF).

NAF's have the lowest density among all signed binary expansions (with coefficients in $\{0, 1, -1\}$).

$$(10010100110111010\underline{11}0)_2 =$$
$$(100101001101110\underline{11}0\bar{1}0)_2 =$$
$$(10010100110\underline{1111}0\bar{1}0\bar{1}0)_2 =$$
$$(10010100\underline{111}000\bar{1}0\bar{1}0\bar{1}0)_2 =$$
$$(1001010100\bar{1}000\bar{1}0\bar{1}0\bar{1}0)_2$$

Results no worse, but not necessarily better:
$35 = (100011)_2 = (10010\bar{1})_s$.

Non-Adjacent Form

IN: Positive integer $n = (n_l n_{l-1} \ldots n_0)_2$, $n_l = n_{l-1} = 0$.

OUT: NAF of n, $(n'_{l-1} \ldots n'_0)_s$.

$c_0 \leftarrow 0$

$\texttt{for } i = 0 \texttt{ to } l - 1 \texttt{ do}$

$\quad c_{i+1} \leftarrow \lfloor (c_i + n_i + n_{i+1})/2 \rfloor$

$\quad n'_i \leftarrow c_i + n_i - 2c_{i+1}$

$\texttt{return } (n'_{l-1} \ldots n'_0)_s$

Resulting signed binary expansion has length at most $l + 1$, so longer by at most 1 bit.

On average there are $l/3$ non-zero coefficients.

Can also do left-to-right.

NAF – example

$c_0 \leftarrow 0$

$\texttt{for } i = 0 \texttt{ to } l - 1 \texttt{ do}$

$\qquad c_{i+1} \leftarrow \lfloor (c_i + n_i + n_{i+1})/2 \rfloor$

$\qquad n'_i \leftarrow c_i + n_i - 2c_{i+1}$

$\texttt{return } (n'_{l-1} \dots n'_0)_s$

$35 = (00100011)_2$, $c_0 = 0$

$c_1 = \lfloor (0 + 1 + 1)/2 \rfloor = 1$,

$n_0 = 0 + 1 - 2 = -1$

$c_2 = \lfloor (1 + 1 + 0)/2 \rfloor = 1$,

$n_1 = 1 + 1 - 2 = 0$

$c_3 = \lfloor (1 + 0 + 0)/2 \rfloor = 0$,

$n_2 = 1 + 0 - 0 = 1$

$$c_4 = \lfloor (0+0+0)/2 \rfloor = 0,$$
$$n_3 = 0+0-0 = 0$$

$$c_5 = \lfloor (0+0+1)/2 \rfloor = 0,$$
$$n_4 = 0+0-0 = 0$$

$$c_6 = \lfloor (0+1+0)/2 \rfloor = 0,$$
$$n_5 = 0+1-0 = 1$$

$$c_7 = \lfloor (0+0+0)/2 \rfloor = 0,$$
$$n_6 = 0+0-0 = 0$$

$$\Rightarrow 35 = (10010\bar{1})_s$$

Generalizations

So far all expansions in base 2 (signed or unsigned).

Generalize to larger base; often 2^w $(w > 1)$. Then the coefficients are in $[0, 2^w - 1]$. Also fractional windows have been suggested.

w is called window width.

Assume that mP for $m \in [0, 2^w - 1]$ are precomputed.

Easiest way: just group w bits.

Sliding windows: Group w bits and skip forward if LSB is 0 (requires only odd integers in $[0, 2^w - 1]$) as coefficients and leads to $l/(w+1)$ additions).

If $-$ is cheap, use signed sliding windows; this leads to $l/(w+2)$ additions.

$(\underline{10}\,\underline{01}\,\underline{01}\,\underline{00}\,\underline{11}\,\underline{01}\,\underline{11}\,\underline{01}\,\underline{01}\,\underline{10})_2 =$
$(02\,01\,01\,00\,03\,01\,03\,01\,01\,02)_2 =$
$(2110313112)_4$,
needs 8 additions and precomputed $2P$ and $3P$.

$$(10010100\underline{11}01\,\underline{11}010\underline{11}0)_2 =$$
$$(1001010003\,0103010030)_2,$$

needs 7 additions and only precomputed $3P$.

$$(1001010011011101\underline{011}0)_2 =$$
$$(10010100\underline{111}0000\bar{3}0030)_2 =$$
$$(10010\underline{101}00\bar{1}0000\bar{3}0030)_2 =$$
$$(10\underline{011}00\bar{3}00\bar{1}0000\bar{3}0030)_2 =$$
$$(1000300\bar{3}00\bar{1}0000\bar{3}0030)_2$$

needs 5 additions and precomputed $3P$, assuming that $-$ is available.

Fixed base point

If the same point P is used with different scalars and if storage is not too restricted one can precompute and store all $P_i = 2^i P$. Then one can obtain nP as

$$nP = \sum n_i P_i$$

using only additions.

Combining this approach with windowing means precomputing $m2^{iw}P$ for $m \in [0, 2^w - 1]$ and $i \in [0, l/w]$. Note that in this case the windows do not slide.

$(2110313112)_4P$ is computed from the $3 \cdot 10$ precomputed points $mP, m4P, m16P, \ldots, m4^9P$ for $m \in \{1, 2, 3\}$ in 8 additions and 0 doublings.

Montgomery Ladder

Consider the intermediate results (i is decreasing).

$Q_i = \sum_{j=i}^{l} n_j 2^{j-i} P$, put $R_i = Q_i + P$, then $Q_i = 2Q_{i+1} + n_i P = Q_{i+1} + R_{i+1} + n_i P - P = 2R_{i+1} + n_i P - 2P$.

This implies $(Q_i, R_i) = \begin{cases} (2Q_{i+1}, Q_{i+1} + R_{i+1}) & \text{if } n_i = 0 \\ (Q_{i+1} + R_{i+1}, 2R_{i+1}) & \text{if } n_i = 1 \end{cases}$

$13 = (1101)_2$:

$(Q_3, R_3) = (P, 2P)$

$(Q_2, R_2) = (3P, 4P)$

$(Q_1, R_1) = (6P, 7P)$

$(Q_0, R_0) = (13P, 14P)$

Idea of joint doublings

To compute

$n_1 P_1 + n_2 P_2 + \cdots + n_m P_m$

compute the doublings together,

i.e. write scalars n_i in binary:

$$
\begin{array}{llllll}
n_1 & = n_{1,l-1} 2^{l-1} & + n_{1,l-2} 2^{l-2} & + & \ldots + n_{1,0} \\
n_2 & = n_{2,l-1} 2^{l-1} & + n_{2,l-2} 2^{l-2} & + & \ldots + n_{2,0} \\
\vdots & = \vdots & \vdots & & \vdots \\
n_m & = n_{m,l-1} 2^{l-1} & + n_{m,l-2} 2^{l-2} & + & \ldots + n_{m,0}
\end{array}
$$

Idea of joint doublings

To compute

$$n_1 P_1 + n_2 P_2 + \cdots + n_m P_m$$

compute the doublings together, i.e. write scalars n_i in binary:

$$\begin{aligned}
n_1 &= n_{1,l-1} 2^{l-1} + n_{1,l-2} 2^{l-2} + \ldots + n_{1,0} \\
n_2 &= n_{2,l-1} 2^{l-1} + n_{2,l-2} 2^{l-2} + \ldots + n_{2,0} \\
\vdots &= \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
n_m &= n_{m,l-1} 2^{l-1} + n_{m,l-2} 2^{l-2} + \ldots + n_{m,0}
\end{aligned}$$

Compute as

$$2\Big(2(n_{1,l-1} P_1 + n_{2,l-1} P_2 + \cdots + n_{m,l-1} P_m) + (n_{1,l-2} P_1 + n_{2,l-2} P_2 + \cdots + n_{m,l-2} P_m)\Big) + \cdots$$

etc. Even with precomputations, many more adds than doublings.

Applications

ECDSA verification uses 2 scalar multiplications ... just to add the results.

If base point P is fixed, precompute $R = 2^{l/2}P$ and include in the curve parameters. Split scalar $n = n_1 2^{l/2} + n_0$ and compute

$$n_1 R + n_0 P.$$

GLV curves split scalar in two halves to get faster scalar multiplication.

Verification in accelerated ECDSA can be extended to use 4 or even 6 scalars. Splitting of the scalar is done by LLL techniques.

Further applications in batch verification of signatures — many scalars — by taking random linear combinations.

Examples

$1338P + 1715Q = (10100111010)_2 P + (11010110011)_2 Q$ takes 20 doublings and 12 additions.

Given precomputed $P+Q$, we can compute the same in 10 doublings and 8 additions.

If $-$ is efficient $(0101010\bar{0}\bar{1}010)_2 P + (100\bar{1}0\bar{1}0\bar{1}010\bar{1})_2 Q$ reduces (compared to first line) the number of additions to 10, but needs 21 doublings.

Given precomputed
$P+Q$ and $P-Q$:
$(0101010 0\bar{1}010)_2 P+$
$(100\bar{1}0\bar{1}0\bar{1}010\bar{1})_2 Q$
needs 11 doublings
and 8 additions.

The joint Hamming weight
has not decreased,
and length has increased.

Asymptotic speeds

1939 Brauer algorithm:

$\approx (1 + 1/\lg\lg H)\lg H$
additions to compute
$P \mapsto nP$ if $n \le H$.

1964 Straus algorithm:

$\approx (1 + k/\lg\lg H)\lg H$
additions to compute
$P_1, \ldots, P_k \mapsto n_1 P_1 + \cdots + n_k P_k$
if $n_1, \ldots, n_k \le H$.

1976 Yao algorithm:

$\approx (1 + k/\lg\lg H)\lg H$
additions to compute
$P \mapsto n_1 P, \ldots, n_k P$
if $n_1, \ldots, n_k \le H$.

1976 Pippenger algorithm:

Similar asymptotics,
but replace $\lg\lg H$ with $\lg(k \lg H)$.
Faster than Straus and Yao
if k is large.
(Knuth summary is wrong.)

More generally, Pippenger's algorithm computes
ℓ sums of multiples of k inputs.

$$\approx \left(\min\{k,\ell\} + \frac{k\ell}{\lg(k\ell \lg H)}\right) \lg H$$

additions
if all coefficients are below H.
Within $1+\epsilon$ of optimal.

More generally, Pippenger's algorithm computes ℓ sums of multiples of k inputs.

$$\approx \left(\min\{k, \ell\} + \frac{k\ell}{\lg(k\ell \lg H)}\right) \lg H$$

additions
if all coefficients are below H.
Within $1 + \epsilon$ of optimal.

Various special cases of Pippenger's algorithm were reinvented and patented by 1993 Brickell–Gordon–McCurley–Wilson, 1995 Lim–Lee, etc.
Is that the end of the story?

No! 1989 Bos–Coster:

If $n_1 \ge n_2 \ge \cdots$ then
$n_1 P_1 + n_2 P_2 + n_3 P_3 + \cdots =$
$(n_1 - q n_2) P_1 + n_2 (q P_1 + P_2) +$
$n_3 P_3 + \cdots$ where $q = \lfloor n_1 / n_2 \rfloor$.

Remarkably simple;
competitive with Pippenger
for random choices of n_i's;
much better memory usage.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P_1$, $16P_2$, $300P_3$, $146P_4$, $77P_5$, $2P_6$, $1P_7$.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$010011010 = 154 \leftarrow$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $154P$, $146P$, $77P$, $2P$, $1P$.

Plus one extra addition: add $146x$ into $154x$, obtaining $300x$.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$000001000 = 8 \leftarrow$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

plus 2 additions.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$000001000 = 8$

$001000101 = 69 \leftarrow$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

plus 3 additions.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$000001000 = 8$

$001000101 = 69$

$000001000 = 8 \leftarrow$

$000000010 = 2$

$000000001 = 1$

plus 4 additions.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$000001000 = 8$

$000100101 = 37 \leftarrow$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 5 additions.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$000001000 = 8$

$000000101 = 5 \leftarrow$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 6 additions.

Reduce largest row:

$000010000 = 16 \leftarrow$

$000010000 = 16$

$000001000 = 8$

$000000101 = 5$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 7 additions.

Reduce largest row:

$000000000 = 0$

$000010000 = 16$

$000001000 = 8$

$000000101 = 5$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 7 additions.

Reduce largest row:

$000000000 = 0$

$000001000 = 8 \leftarrow$

$000001000 = 8$

$000000101 = 5$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 8 additions.

Reduce largest row:

$$000000000 = 0$$
$$000000000 = 0 \leftarrow$$
$$000001000 = 8$$
$$000000101 = 5$$
$$000001000 = 8$$
$$000000010 = 2$$
$$000000001 = 1$$

plus 8 additions.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0 \leftarrow$

$000000101 = 5$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 8 additions.

Reduce largest row:

$$000000000 = 0$$
$$000000000 = 0$$
$$000000000 = 0$$
$$000000101 = 5$$
$$000000011 = 3 \leftarrow$$
$$000000010 = 2$$
$$000000001 = 1$$

plus 9 additions.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000010 = 2 \leftarrow$

$000000011 = 3$

$000000010 = 2$

$000000001 = 1$

plus 10 additions.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000010 = 2$

$000000001 = 1 \leftarrow$

$000000010 = 2$

$000000001 = 1$

plus 11 additions.

Reduce largest row:

$$000000000 = 0$$

$$000000000 = 0$$

$$000000000 = 0$$

$$000000000 = 0 \leftarrow$$

$$000000001 = 1$$

$$000000010 = 2$$

$$000000001 = 1$$

plus 11 additions.

Reduce largest row:

$$000000000 = 0$$
$$000000000 = 0$$
$$000000000 = 0$$
$$000000000 = 0$$
$$000000001 = 1$$
$$000000001 = 1 \leftarrow$$
$$000000001 = 1$$

plus 12 additions.

Reduce largest row:

$$000000000 = 0$$

$$000000000 = 0$$

$$000000000 = 0$$

$$000000000 = 0$$

$$000000000 = 0 \leftarrow$$

$$000000001 = 1$$

$$000000001 = 1$$

plus 12 additions.

Reduce largest row:

$$000000000 = 0$$

$$000000000 = 0$$

$$000000000 = 0$$

$$000000000 = 0$$

$$000000000 = 0$$

$$000000000 = 0 \leftarrow$$

$$000000001 = 1$$

plus 12 additions.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0 \leftarrow$

plus 12 additions.

Final addition chain: 1, 2, 3, 5, 8, 16, 32, 37, 69, 77, 146, 154, 300.

Short, no temporary storage, low two-operand complexity.

NaCl

The cryptography part of the Networking-and-Cryptography library has 2 main functionalities:

Authenticated encryption:
Use particular Montgomery curve (Curve25519) over $\mathbf{F}_p$ with $p = 2^{255} - 19$ for DH key exchange; use Poly1305 for authentication and Salsa20 for encryption.

Signatures:
Same curve in Edwards form;
do scalar multiplication in
fixed basepoint scenario
with windowing; use EdDSA
protocol to avoid problems with
randomness:

Secret key of user produces a
used in $P_A = aP$ and string b.
Nonce k is computed as
$k =$hash(b, m).
$R = kP$ as in ECDSA.
Hash value h made dependent on
R and P_A as $h =$hash(R, P_A, m).

Signing and verification avoid inversions
$s \equiv k + ha \bmod \ell$.

Verification: $sP = R + hP_A$?

Batch verification of (R_i, s_i) with h_i =hash(R_i, P_{A_i}, m_i) picks random scalars z_i and checks whether
$(-\sum_i z_i s_i \bmod \ell)P + \sum_i z_i R_i + \sum_i (z_i h_i \bmod \ell) P_{A_i} = 0$.
Batch verification uses Bos-Coster algorithm.