ECCHacks:
a gentle introduction
to elliptic-curve cryptography

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

Tanja Lange
Technische Universiteit Eindhoven

---

ecchacks.cr.yp.to

Cryptography

Public-key signatures:
e.g., RSA, DSA, ECDSA.
Some uses: signed OS updates,
SSL certificates, e-passports.

Public-key encryption:
e.g., RSA, DH, ECDH.
Some uses: SSL key exchange,
locked iPhone mail download.

Secret-key encryption:
e.g., AES, Salsa20.
Some uses: disk encryption,
bulk SSL encryption.

Why ECC?

“Index calculus”: fastest method we know to break original DH and RSA.

Long history,
including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS “cryptopocalypse”.

(FFS is not relevant to RSA.)

Also many smaller improvements:
$\approx$ 100 scientific papers.

Approximate costs of these algorithms for breaking RSA-1024, RSA-2048:

CFRAC: 2^{120}, 2^{170}.
LS: 2^{110}, 2^{160}.
QS: 2^{100}, 2^{150}.
NFS: 2^{80}, 2^{112}.

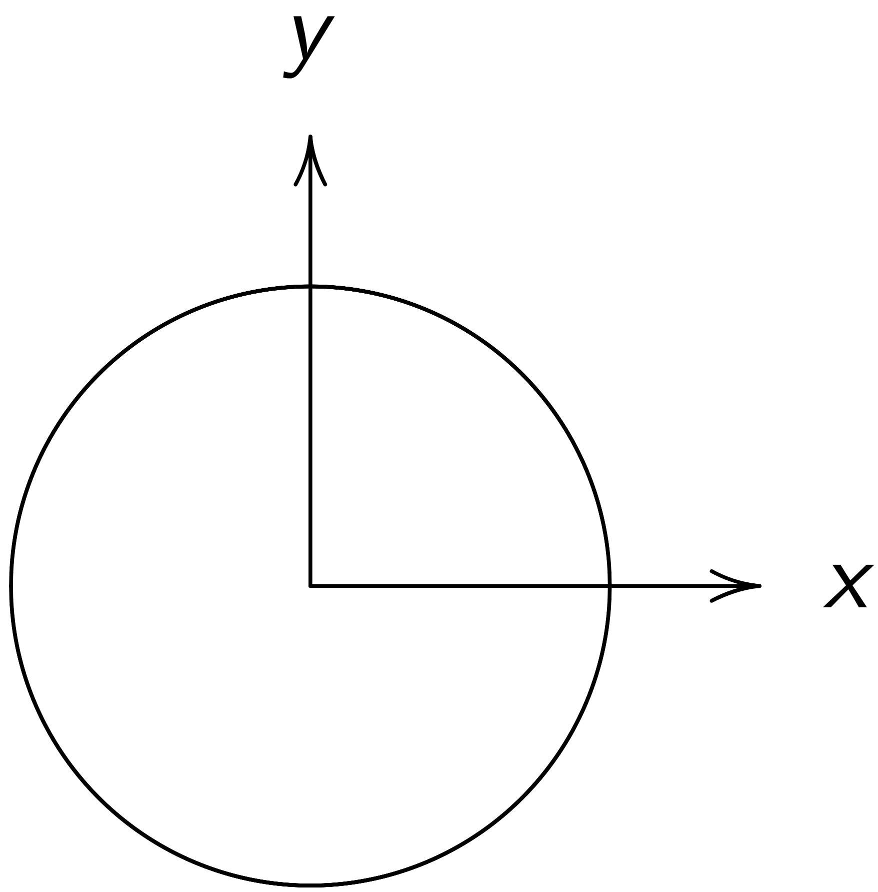
Also many smaller improvements:
$\approx$ 100 scientific papers.

Approximate costs of these algorithms for breaking RSA-1024, RSA-2048:

CFRAC: 2^{120}, 2^{170}.
LS: 2^{110}, 2^{160}.
QS: 2^{100}, 2^{150}.
NFS: 2^{80}, 2^{112}.

1985 Miller
"Use of elliptic curves in cryptography":
"It is extremely unlikely that an 'index calculus' attack on the elliptic curve method will ever be able to work."

The clock



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".

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

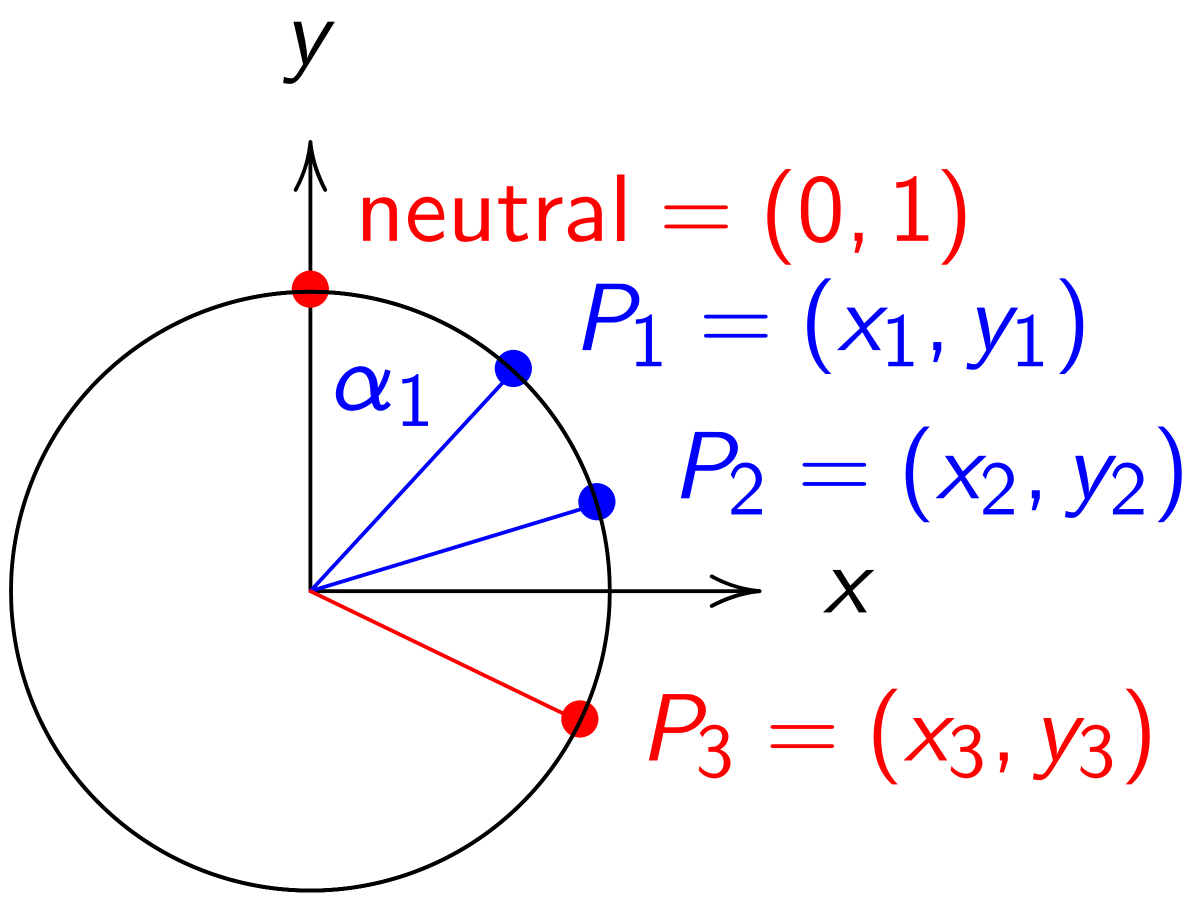
$(3/5, -4/5)$. $(-3/5, -4/5)$.

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

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

Many more.

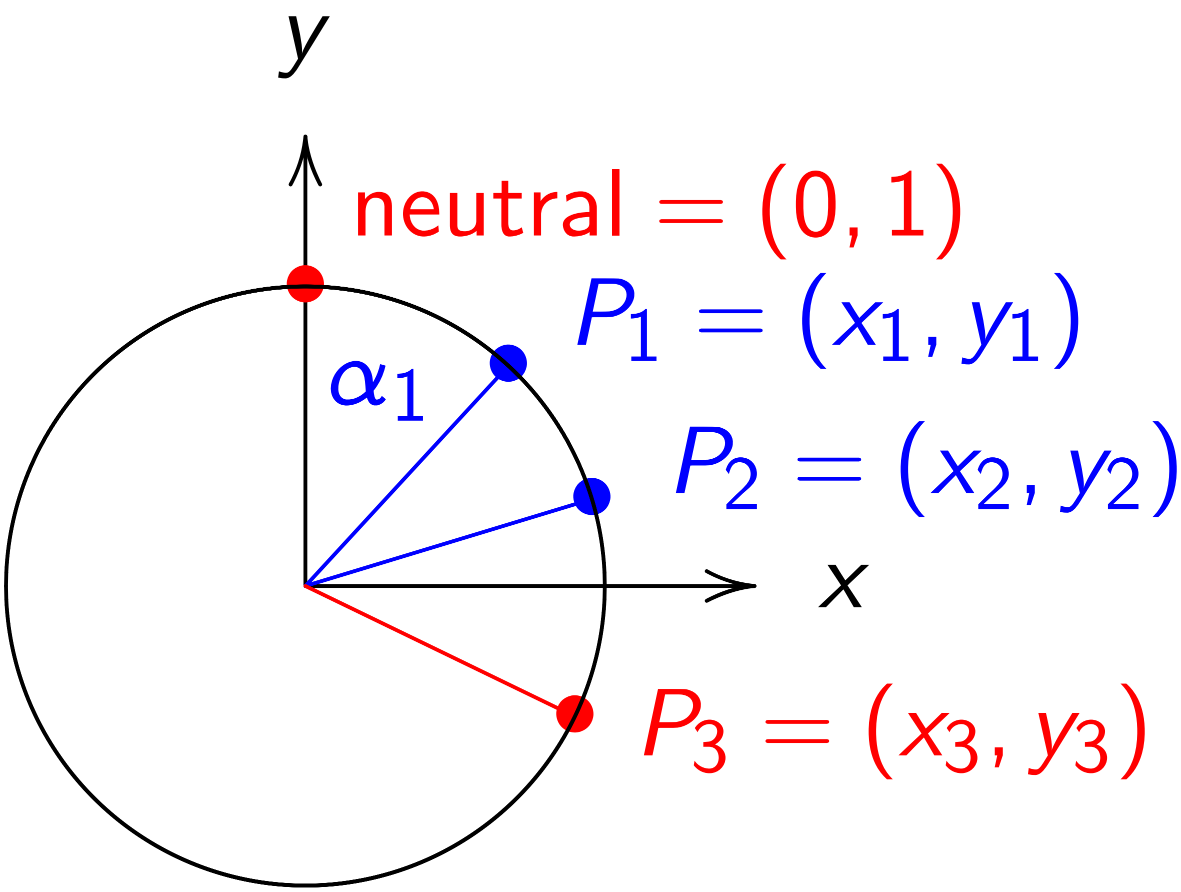
Addition on the clock:



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

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

Addition on the clock:

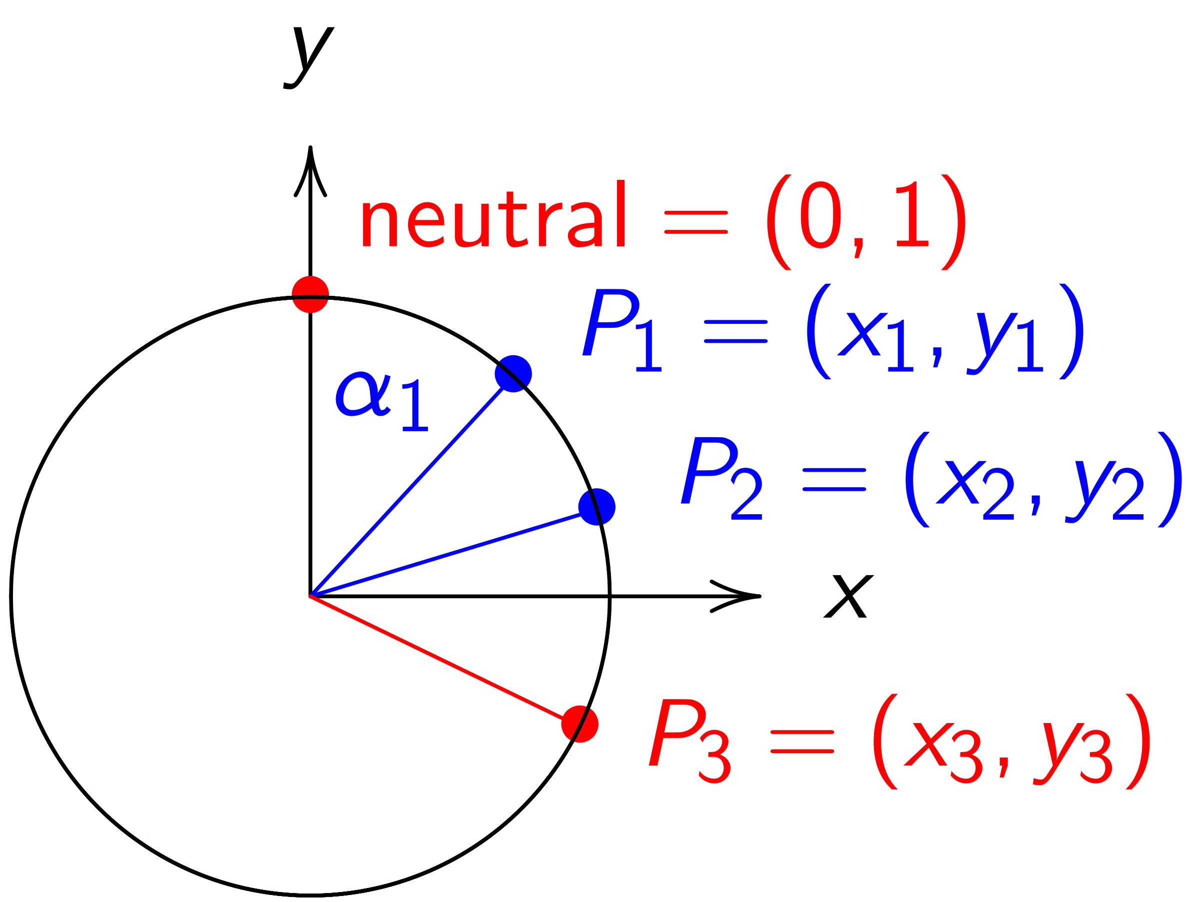


$x^2 + y^2 = 1$, parametrized by

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

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

Addition on the clock:



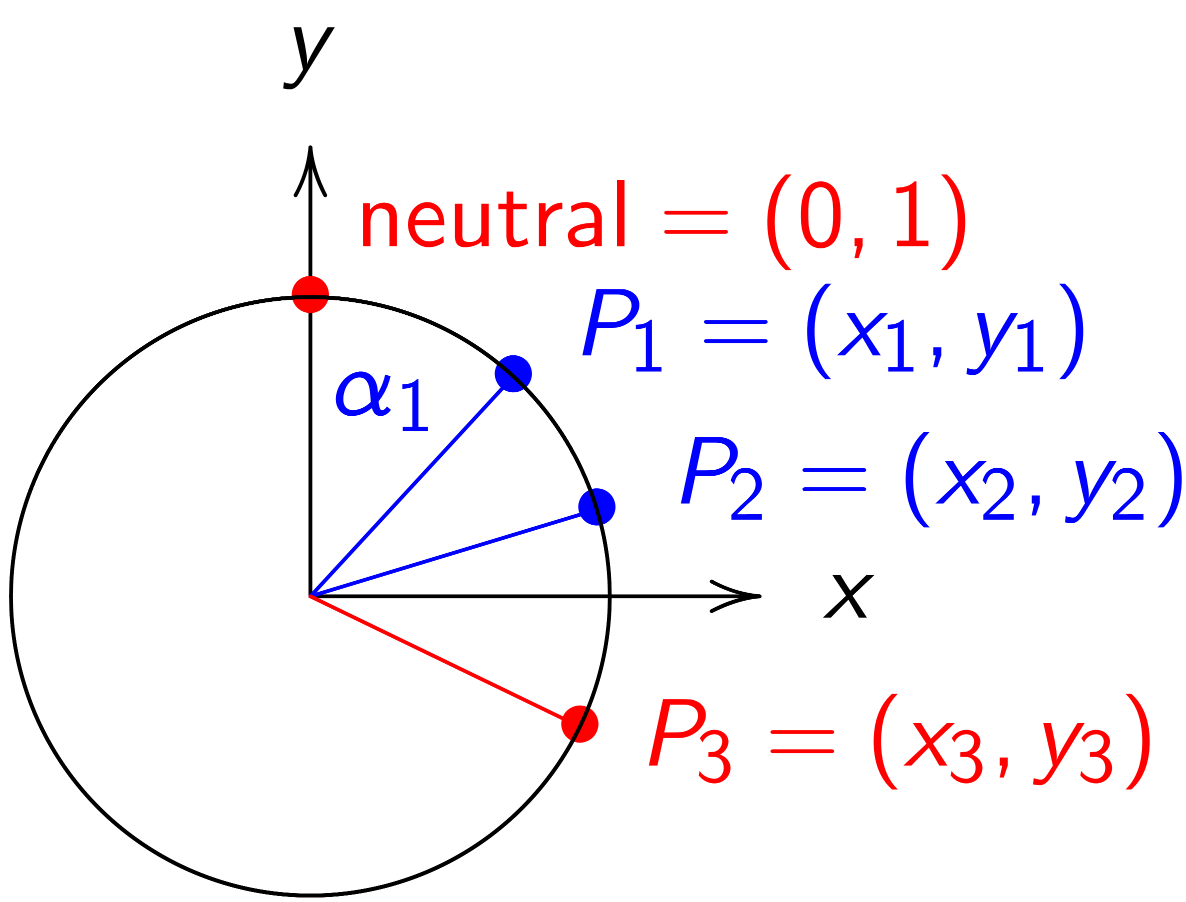
$x^2 + y^2 = 1$, parametrized by

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

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

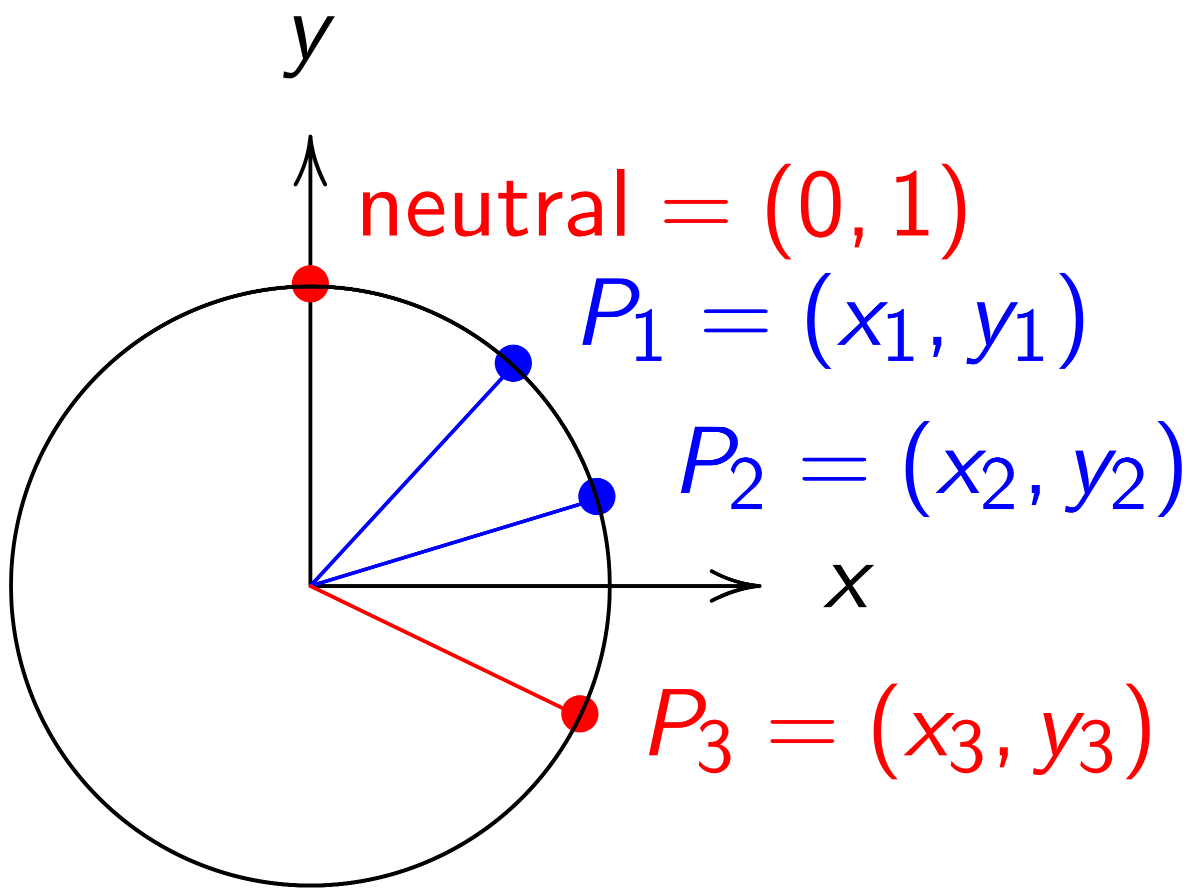
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$

Addition on the clock:



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

Clock addition without sin, cos:



Use Cartesian coordinates for addition.
Addition formula
for the clock $x^2 + y^2 = 1$:
sum of (x_1, y_1) and (x_2, y_2) is
$(x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

Examples of clock addition:

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

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

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

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:

"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"

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

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

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

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

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

Examples of clock addition:

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

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

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

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

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

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

Examples of clock addition:

"2:00" + "5:00"

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

"5:00" + "9:00"

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

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

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

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

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

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

Examples of clock addition:

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

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

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

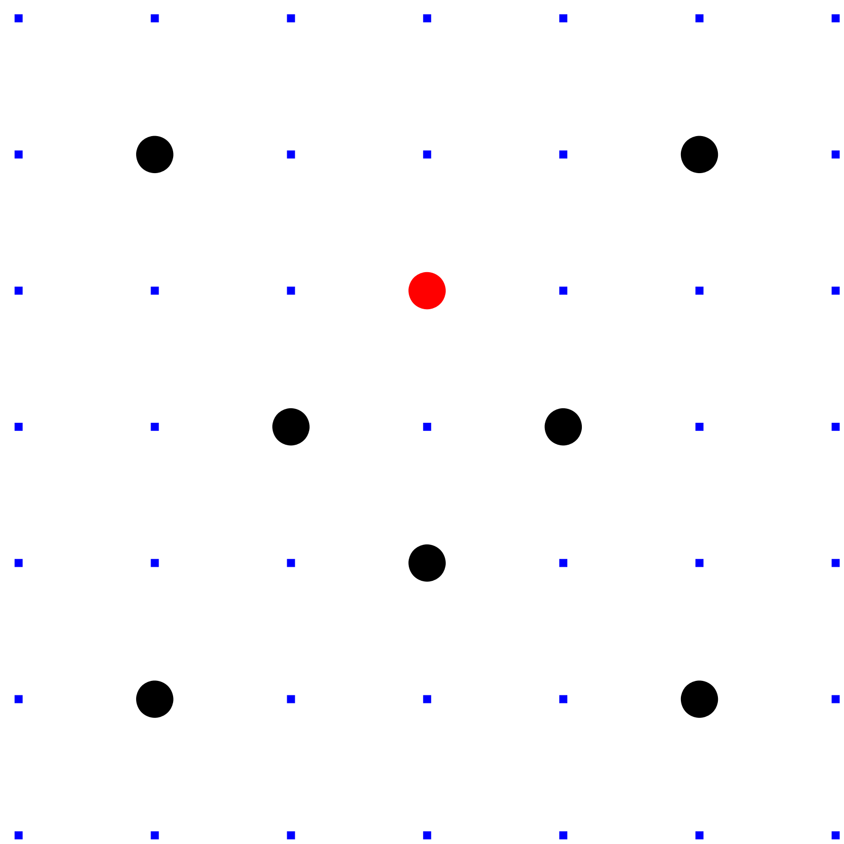
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

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

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

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

Clocks over finite fields



$\mathrm{Clock}(\mathbf{F}_7) = \{(x, y) \in \mathbf{F}_7 \times \mathbf{F}_7 : x^2 + y^2 = 1\}$.

Here $\mathbf{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$

$= \{0, 1, 2, 3, -3, -2, -1\}$

with arithmetic modulo 7.

e.g. $2 \cdot 5 = 3$ and $3/2 = 5$ in $\mathbf{F}_7$.

```
>>> for x in range(7):
...   for y in range(7):
...     if (x*x+y*y) % 7 == 1:
...       print (x,y)
...
(0, 1)
(0, 6)
(1, 0)
(2, 2)
(2, 5)
(5, 2)
(5, 5)
(6, 0)
>>>
```

```
>>> class F7:
...   def __init__(self,x):
...     self.int = x % 7
...   def __str__(self):
...     return str(self.int)
...   __repr__ = __str__
...
>>> print F7(2)
2
>>> print F7(6)
6
>>> print F7(7)
0
>>> print F7(10)
3
```

```
>>> F7.__eq__ = \
...   lambda a,b: a.int == b.int
>>>
>>> print F7(7) == F7(0)
True
>>> print F7(10) == F7(3)
True
>>> print F7(-3) == F7(4)
True
>>> print F7(0) == F7(1)
False
>>> print F7(0) == F7(2)
False
>>> print F7(0) == F7(3)
False
```

```
>>> F7.__add__ = \
...    lambda a,b: F7(a.int + b.int)
>>> F7.__sub__ = \
...    lambda a,b: F7(a.int - b.int)
>>> F7.__mul__ = \
...    lambda a,b: F7(a.int * b.int)
>>>
>>> print F7(2) + F7(5)
0
>>> print F7(2) - F7(5)
4
>>> print F7(2) * F7(5)
3
>>>
```

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

```
p = 1000003
class Fp:
  ...

def clockadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = x1*y2+y1*x2
  y3 = y1*y2-x1*x2
  return x3,y3
```

```
>>> P = (Fp(1000),Fp(2))
>>> P2 = clockadd(P,P)
>>> print P2
(4000, 7)
>>> P3 = clockadd(P2,P)
>>> print P3
(15000, 26)
>>> P4 = clockadd(P3,P)
>>> P5 = clockadd(P4,P)
>>> P6 = clockadd(P5,P)
>>> print P6
(780000, 1351)
>>> print clockadd(P3,P3)
(780000, 1351)
>>>
```

```
>>> def scalarmult(n,P):
...   if n == 0: return (Fp(0),Fp(1))
...   if n == 1: return P
...   Q = scalarmult(n//2,P)
...   Q = clockadd(Q,Q)
...   if n % 2: Q = clockadd(P,Q)
...   return Q
...
>>> n = oursixdigitsecret
>>> scalarmult(n,P)
(947472, 736284)
>>>
```

Can you figure out our secret n?

Clock cryptography

The "Clock Diffie–Hellman protocol":

Standardize a large prime p
and **base point** $(x, y) \in \mathrm{Clock}(\mathbf{F}_p)$.

Alice chooses big secret a.
Alice computes her public key $a(x, y)$.

Bob chooses big secret b.
Bob computes his public key $b(x, y)$.

Alice computes $a(b(x, y))$.
Bob computes $b(a(x, y))$.
They use this shared secret
to encrypt with AES-GCM etc.

$$
\begin{tikzcd}[column sep=small]
\text{Alice's secret key } a \arrow[d] \arrow[dd, bend right=60] & & \text{Bob's secret key } b \arrow[d] \arrow[dd, bend left=60] \\
\begin{array}{c}\text{Alice's public key}\\ a(x,y)\end{array} \arrow[drr] & & \begin{array}{c}\text{Bob's public key}\\ b(x,y)\end{array} \arrow[dll] \\
\begin{array}{c}\{\text{Alice}, \text{Bob}\}\text{'s}\\ \text{shared secret}\\ ab(x,y)\end{array} & = & \begin{array}{c}\{\text{Bob}, \text{Alice}\}\text{'s}\\ \text{shared secret}\\ ba(x,y)\end{array}
\end{tikzcd}
$$

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

Warning #1: Many choices of p are unsafe!

Warning #2: Clocks aren't elliptic!
Can use index calculus
to attack clock cryptography.
To match RSA-3072 security
need $p \approx 2^{1536}$.

Warning #3: Attacker sees more than the public keys $a(x, y)$ and $b(x, y)$.

Attacker sees how much *time* Alice uses to compute $a(b(x, y))$. Often attacker can see time for *each operation* performed by Alice, not just total time. This reveals secret scalar a.

Some timing attacks: 2011 Brumley–Tuveri; 2013 "Lucky Thirteen" (not ECC); 2014 Benger–van de Pol–Smart–Yarom; etc.

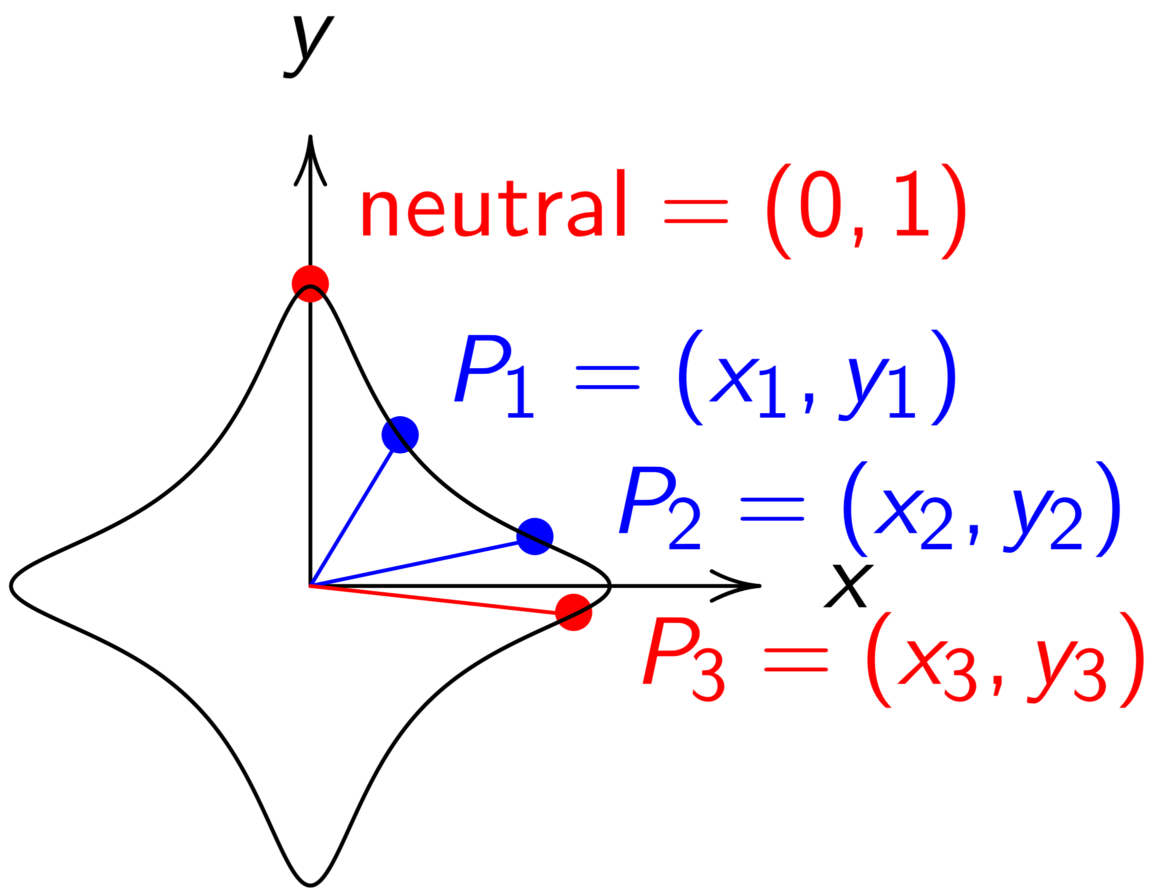
Warning #3: Attacker sees more than the public keys $a(x, y)$ and $b(x, y)$.

Attacker sees how much *time* Alice uses to compute $a(b(x, y))$. Often attacker can see time for *each operation* performed by Alice, not just total time. This reveals secret scalar a.

Some timing attacks: 2011 Brumley–Tuveri; 2013 "Lucky Thirteen" (not ECC); 2014 Benger–van de Pol–Smart–Yarom; etc.

Fix: **constant-time** code, performing same operations no matter what scalar is.

Addition on an elliptic curve



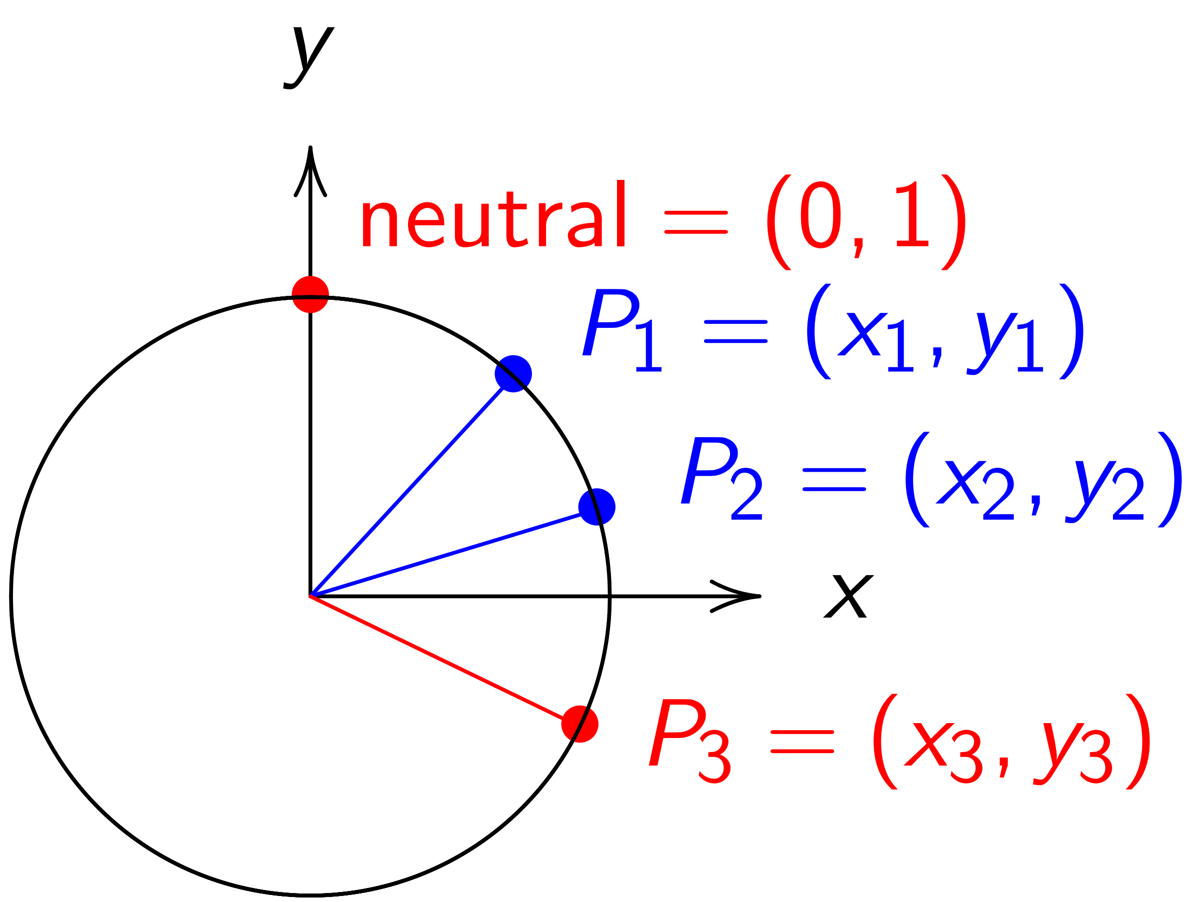
$x^2 + y^2 = 1 - 30x^2y^2$.

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

$(x_1 y_2 + y_1 x_2,$

$\ y_1 y_2 - x_1 x_2).$

More elliptic curves

Choose an odd prime p.

Choose a *non-square* $d \in \mathbf{F}_p$.

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

is a "complete Edwards curve".

```
def edwardsadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = (x1*y2+y1*x2)/(1+d*x1*x2*y1*y2)
  y3 = (y1*y2-x1*x2)/(1-d*x1*x2*y1*y2)
  return x3,y3
```

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

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

Answer: Can prove that
the denominators are never 0.
Addition law is **complete**.

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

Answer: Can prove that
the denominators are never 0.
Addition law is **complete**.

This proof relies on
choosing *non-square* d.

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

Answer: Can prove that
the denominators are never 0.
Addition law is **complete**.

This proof relies on
choosing *non-square d*.

If we instead choose square d:
curve is still elliptic, and
addition *seems to work*,
but there are failure cases,
often exploitable by attackers.
Safe code is more complicated.

"Hey, divisions are really slow!"

“Hey, divisions are really slow!”

Instead of dividing a by b,
store fraction a/b as pair (a, b).
Remember arithmetic on fractions?

"Hey, divisions are really slow!"

Instead of dividing a by b,
store fraction a/b as pair (a, b).
Remember arithmetic on fractions?

One option: "projective coordinates".
Store (X, Y, Z) representing $(X/Z, Y/Z)$.

Another option: "extended coordinates".
Store projective (X, Y, Z) and $T = XY/Z$.

See "Explicit Formulas Database"
for many more options and speedups:
`hyperelliptic.org/EFD`

Elliptic-curve cryptography

Standardize prime p, safe non-square d, base point (x, y) on elliptic curve.

Alice knows her secret key a and Bob's public key $b(x, y)$. Alice computes (and caches) shared secret $ab(x, y)$.

Alice uses shared secret to encrypt and authenticate packet for Bob.

Packet overhead at high security level:
32 bytes for Alice's public key,
24 bytes for nonce,
16 bytes for authenticator.

Bob receives packet,
sees Alice's public key $a(x, y)$.
Bob computes (and caches)
shared secret $ab(x, y)$.

Bob uses shared secret to
verify authenticator and decrypt packet.

Alice and Bob
reuse the same shared secret to
encrypt, authenticate, verify, and decrypt
all subsequent packets.

All of this is so fast that
we can afford to encrypt all packets.

A safe example

Choose $p = 2^{255} - 19$.

Choose $d = 121665/121666$;

this is non-square in $\mathbf{F}_p$.

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

is a safe curve for ECC.

A safe example

Choose $p = 2^{255} - 19$.

Choose $d = 121665/121666$;

this is non-square in $\mathbf{F}_p$.

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

is a safe curve for ECC.

$-x^2 + y^2 = 1 - dx^2y^2$

is another safe curve

using the same p and d.

A safe example

Choose $p = 2^{255} - 19$.

Choose $d = 121665/121666$;

this is non-square in $\mathbf{F}_p$.

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

is a safe curve for ECC.

$-x^2 + y^2 = 1 - dx^2y^2$

is another safe curve

using the same p and d.

Actually, the second curve

is the first curve in disguise:

replace x in first curve

by $\sqrt{-1} \cdot x$, using $\sqrt{-1} \in \mathbf{F}_p$.

Even more elliptic curves

Edwards curves:
$x^2 + y^2 = 1 + dx^2y^2$.

Twisted Edwards curves:
$ax^2 + y^2 = 1 + dx^2y^2$.

Weierstrass curves:
$y^2 = x^3 + a_4x + a_6$.

Montgomery curves:
$By^2 = x^3 + Ax^2 + x$.

Many relationships:
e.g., obtain Edwards (x, y)
given Montgomery (x', y') by
computing $x = x'/y'$, $y = (x' - 1)/(x' + 1)$.

Addition on Weierstrass curves

$y^2 = x^3 + a_4 x + a_6$:

Addition on Weierstrass curves

$y^2 = x^3 + a_4 x + a_6$:

for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;

for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;

$(x_1, y_1) + (x_1, -y_1) = \infty$;

$(x_1, y_1) + \infty = (x_1, y_1)$;

$\infty + (x_2, y_2) = (x_2, y_2)$;

$\infty + \infty = \infty$.

Addition on Weierstrass curves

$y^2 = x^3 + a_4 x + a_6$:

for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;

for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;

$(x_1, y_1) + (x_1, -y_1) = \infty$;

$(x_1, y_1) + \infty = (x_1, y_1)$;

$\infty + (x_2, y_2) = (x_2, y_2)$;

$\infty + \infty = \infty$.

Messy to implement and test.

Much nicer than Weierstrass: Montgomery curves with the "Montgomery ladder".

```
def scalarmult(n,x1):
  x2,z2,x3,z3 = 1,0,x1,1
  for i in reversed(range(maxnbits)):
    bit = 1 & (n >> i)
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
    x3,z3 = ((x2*x3-z2*z3)^2,
             x1*(x2*z3-z2*x3)^2)
    x2,z2 = ((x2^2-z2^2)^2,
             4*x2*z2*(x2^2+A*x2*z2+z2^2))
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
  return x2*z2^(p-2)
```

Curve selection

How to defend yourself against
an attacker armed with a mathematician:

1999 ANSI X9.62.
2000 IEEE P1363.
2000 Certicom SEC 2.
2000 NIST FIPS 186-2.
2001 ANSI X9.63.
2005 Brainpool.
2005 NSA Suite B.
2010 Certicom SEC 2 v2.
2010 OSCCA SM2.
2011 ANSSI FRP256V1.

You can pick any of these standards.

What your chosen standard achieves:
No known attack will compute
ECC user's secret key from public key.
("Elliptic-curve discrete-log problem.")

Example of criterion in all standards:
Standard base point (x, y)
has huge prime "order" ℓ,
i.e., exactly ℓ different multiples.

All criteria are computer-verifiable.
See our evaluation site for scripts:
`safecurves.cr.yp.to`

You do everything right.

You pick the Brainpool curve
`brainpoolP256t1`: huge prime p,
$y^2 = x^3 - 3x + \text{somehugenumber}$,
standard base point.

This curve isn't compatible
with Edwards or Montgomery.
So you check and test every case
in the Weierstrass formulas.

You make it all constant-time.
It's horrendously slow,
but it's secure.

Actually, it's not. **You're screwed.**

The attacker sent you (x', y') with

$x' =$ `1025b35abab9150d86770f6bda12f8ec` `1e86bec6c6bac120535e4134fea87831` and

$y' =$ `12ace5eeae9a5b0bca8ed1c0f9540d05` `d123d55f68100099b65a99ac358e3a75`.

You computed "shared secret" $a(x', y')$ using the Weierstrass formulas.

You encrypted data using AES-GCM with a hash of $a(x', y')$ as a key.

Actually, it's not. **You're screwed.**

The attacker sent you (x', y') with

$x' =$ `1025b35abab9150d86770f6bda12f8ec` `1e86bec6c6bac120535e4134fea87831` and

$y' =$ `12ace5eeae9a5b0bca8ed1c0f9540d05` `d123d55f68100099b65a99ac358e3a75`.

You computed "shared secret" $a(x', y')$ using the Weierstrass formulas.
You encrypted data using AES-GCM with a hash of $a(x', y')$ as a key.

What you never noticed:
(x', y') isn't his public key $b(x, y)$;
it isn't even a point on `brainpoolP256t1`;
it's a point on $y^2 = x^3 - 3x + 5$
of order only 4999.

Your formulas worked for $y^2 = x^3 - 3x + 5$ because they work for any $y^2 = x^3 - 3x + a_6$:

Addition on Weierstrass curves
$y^2 = x^3 + a_4 x + a_6$:
for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;
for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;
$(x_1, y_1) + (x_1, -y_1) = \infty$;
$(x_1, y_1) + \infty = (x_1, y_1)$;
$\infty + (x_2, y_2) = (x_2, y_2)$;
$\infty + \infty = \infty$.
Messy to implement and test.

No a_6 here!

Why this matters: (x', y') has order 4999.
$a(x', y')$ is determined by a mod 4999.
The attacker tries all 4999 possibilities,
compares to the AES-GCM output,
learns your secret a mod 4999.

Why this matters: (x', y') has order 4999.

$a(x', y')$ is determined by a mod 4999.

The attacker tries all 4999 possibilities,

compares to the AES-GCM output,

learns your secret a mod 4999.

Attacker then tries again with

$x' = \begin{matrix}\texttt{9bc001a0d2d5c43863aadb0f881df3bb}\\ \texttt{af3a5ea81eedd2385e6525521aa8b1e2}\end{matrix}$ and

$y' = \begin{matrix}\texttt{0d124e9e94dcede52aa0e3bcac1852cf}\\ \texttt{ed28eb86039c0d8e0cfaa4ae703eac07}\end{matrix}$,

a point of order 19559

on $y^2 = x^3 - 3x + 211$;

learns your secret a mod 19559.

Etc. Uses “Chinese remainder theorem”

to combine this information.

Traditional response to this security failure:
Blame the implementor.

“You should have checked that the incoming (x', y') was on the right curve and had the right order.”
(And maybe paid patent fees to Certicom.)

Traditional response to this security failure:
Blame the implementor.

"You should have checked that
the incoming (x', y') was on the right curve
and had the right order."
(And maybe paid patent fees to Certicom.)

But it's much better to
design the system without traps.

Never send uncompressed (x, y)**.**
Design protocols to compress
one coordinate down to 1 bit, or 0 bits!
Drastically limits possibilities
for attacker to choose points.

Always multiply DH scalar by cofactor.

If the curve has $c \cdot \ell$ points
and the base point P has order ℓ
then c is called the cofactor
and $c \cdot \ell$ is called the curve order.

Design DH protocols to multiply by c.

Always choose twist-secure curves.

Montgomery formulas use only A,
but modifying B gives only *two* different
curve orders. Require both of these orders
to be large primes times small cofactors.

DH protocols with all of these protections
are robust against
every common DH implementation error.

ECC standards: the next generation

Fix the standard curves and protocols so that **simple** implementations are **secure** implementations.

Bonus: next-generation curves such as Curve25519 are faster than the standards!

2010.03 Adam Langley, TLS mailing list: "Curve25519 doesn't currently appear on IANA's list . . . and we [Google] would like to see it included."

ECC standards: the next generation

Fix the standard curves and protocols so that **simple** implementations are **secure** implementations.

Bonus: next-generation curves such as Curve25519 are faster than the standards!

2010.03 Adam Langley, TLS mailing list: "Curve25519 doesn't currently appear on IANA's list . . . and we [Google] would like to see it included."

2013.05 Bernstein–Krasnova–Lange specify a procedure to generate a next-generation curve at any security level.

2013.09 Patrick Pelletier: “Given the doubt that’s recently been cast on the NIST curves, is it time to revive the idea of adding curve25519 as a named curve?”

2013.09 Patrick Pelletier: "Given the doubt that's recently been cast on the NIST curves, is it time to revive the idea of adding curve25519 as a named curve?"

2013.09 Douglas Stebila: Reasons to support Curve25519 are "efficiency and resistance to side-channel attacks" rather than concerns about backdoors.

2013.09 Nick Mathewson: "In the FOSS cryptography world nowadays, I see many more new users of curve25519 than of the NIST curves, because of efficiency and ease-of-implementation issues."

2013.09 Nico Williams:

“Agreed, we need curve25519 cipher suites
because of its technical advantages,
not due to any FUD about the other
ECDH curves that we have.”

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

2013.10 We announce SafeCurves site.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 "chosen curves", including 13 next-generation curves.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 “chosen curves”, including 13 next-generation curves.

2014.06 CFRG announces change of leadership.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 “chosen curves”, including 13 next-generation curves.

2014.06 CFRG announces change of leadership. Previous co-chair from NSA “will work with the two new chairs until he retires next year”.

[. . . more than 1000 email messages . . .]

[. . . more than 1000 email messages . . .]

2014.12 CFRG discussion is continuing.

[. . . more than 1000 email messages . . .]

2014.12 CFRG discussion is continuing.

Sage scripts to verify criteria for
ECDLP security and ECC security:
`safecurves.cr.yp.to`

Analysis of manipulability of various
curve-generation methods:
`safecurves.cr.yp.to/bada55.html`

Many computer-verified addition formulas:
`hyperelliptic.org/EFD/`

Python scripts for this talk:
`ecchacks.cr.yp.to`