Discrete-log attacks and factorization Part I

Tanja Lange
Technische Universiteit Eindhoven

11 & 13 June 2019

with some slides by
Daniel J. Bernstein

Main goal of this course:
We are the attackers.
We want to break ECC and RSA.

First need to understand ECC.

Main motivation for ECC:
Avoid index-calculus attacks
that plague finite-field DL.

-log attacks
orization

ange
che Universiteit Eindhoven

June 2019

ne slides by
. Bernstein

Main goal of this course:
We are the attackers.
We want to break ECC and RSA.

First need to understand ECC.

Main motivation for ECC:
Avoid index-calculus attacks
that plague finite-field DL.

Diffie-He

Pick son
i.e. som
(using a

Alic
secret

Alic
public
a

{Alice,
shared
ab

ks

siteit Eindhoven

9

y

n

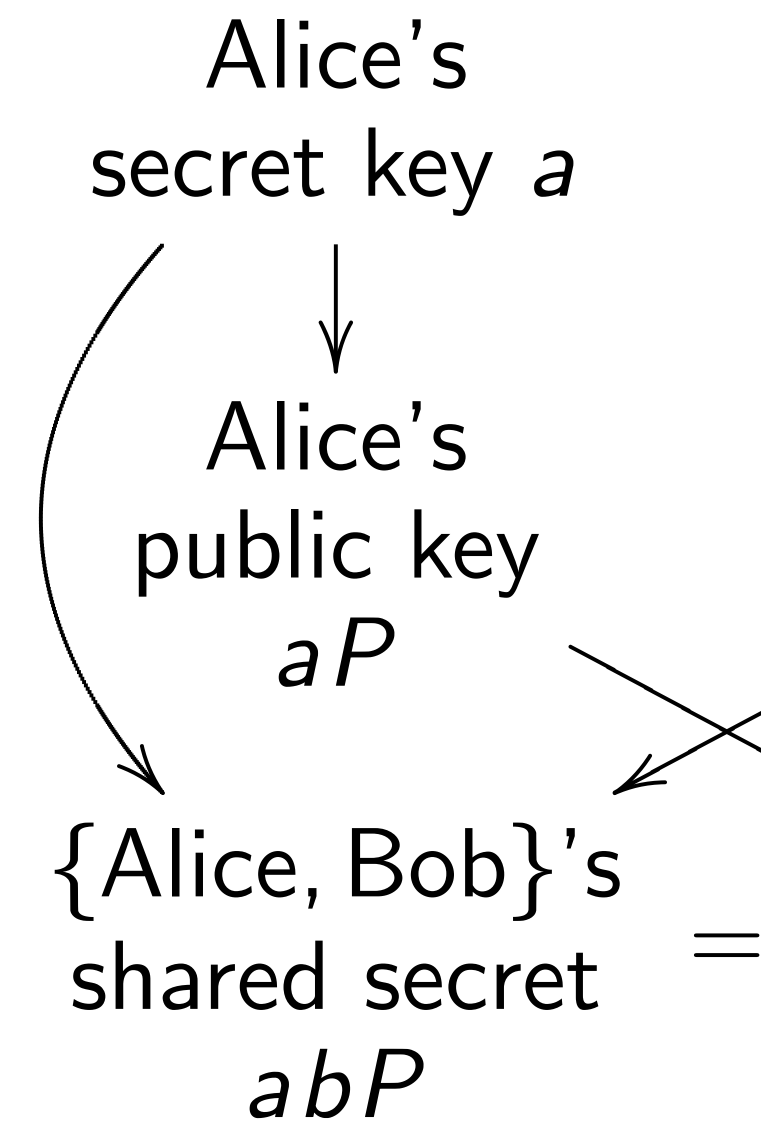
Main goal of this course:
We are the attackers.
We want to break ECC and RSA.

First need to understand ECC.

Main motivation for ECC:
Avoid index-calculus attacks
that plague finite-field DL.

Diffie-Hellman key

Pick some *generat*
i.e. some group el
(using additive no



hoven

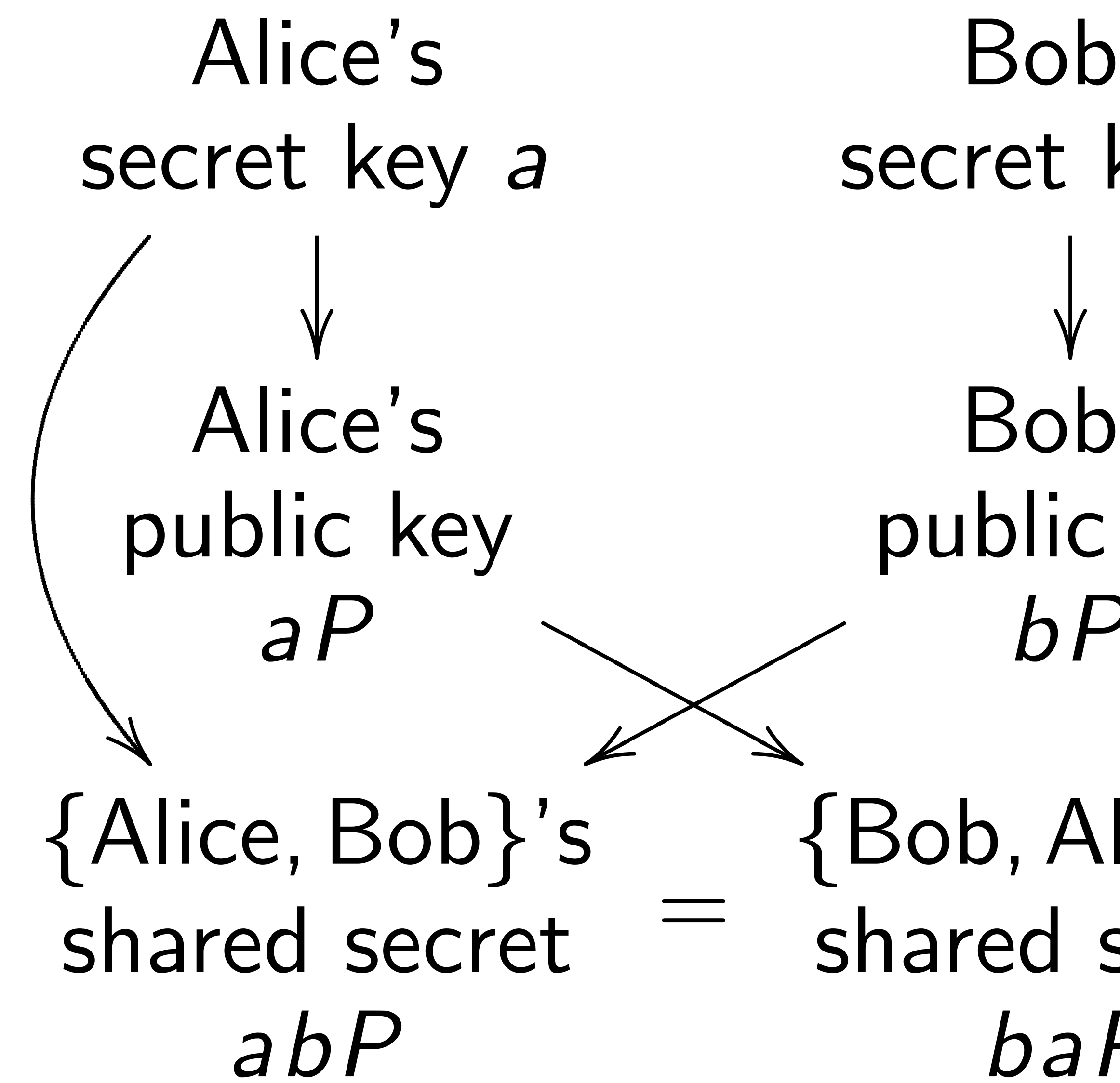
Main goal of this course:
We are the attackers.
We want to break ECC and RSA.

First need to understand ECC.

Main motivation for ECC:
Avoid index-calculus attacks
that plague finite-field DL.

Diffie-Hellman key exchange

Pick some *generator* P,
i.e. some group element
(using additive notation here



Main goal of this course:

We are the attackers.

We want to break ECC and RSA.

First need to understand ECC.

Main motivation for ECC:

Avoid index-calculus attacks

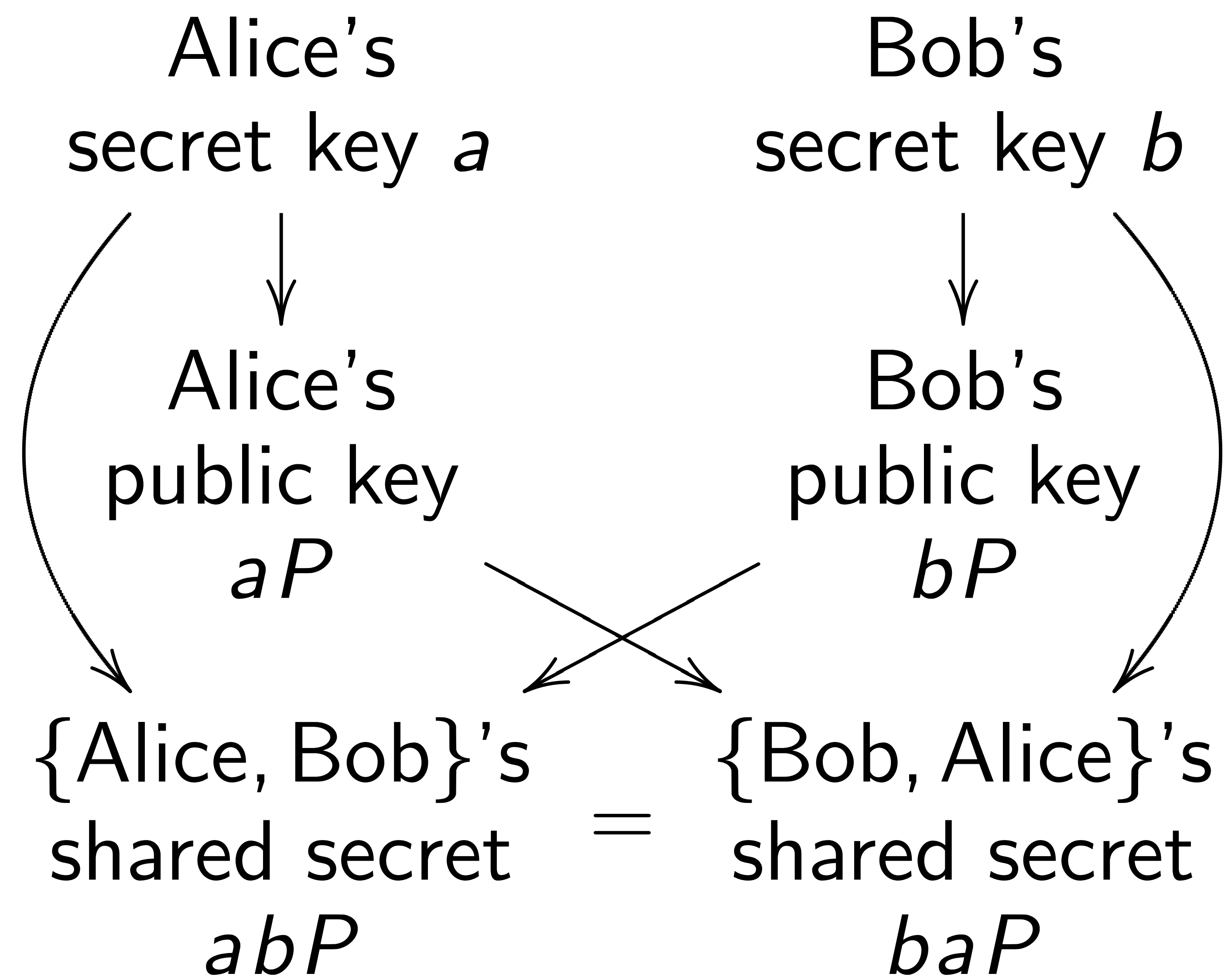
that plague finite-field DL.

Diffie-Hellman key exchange

Pick some *generator* P,

i.e. some group element

(using additive notation here).



Main goal of this course:

We are the attackers.

We want to break ECC and RSA.

First need to understand ECC.

Main motivation for ECC:

Avoid index-calculus attacks

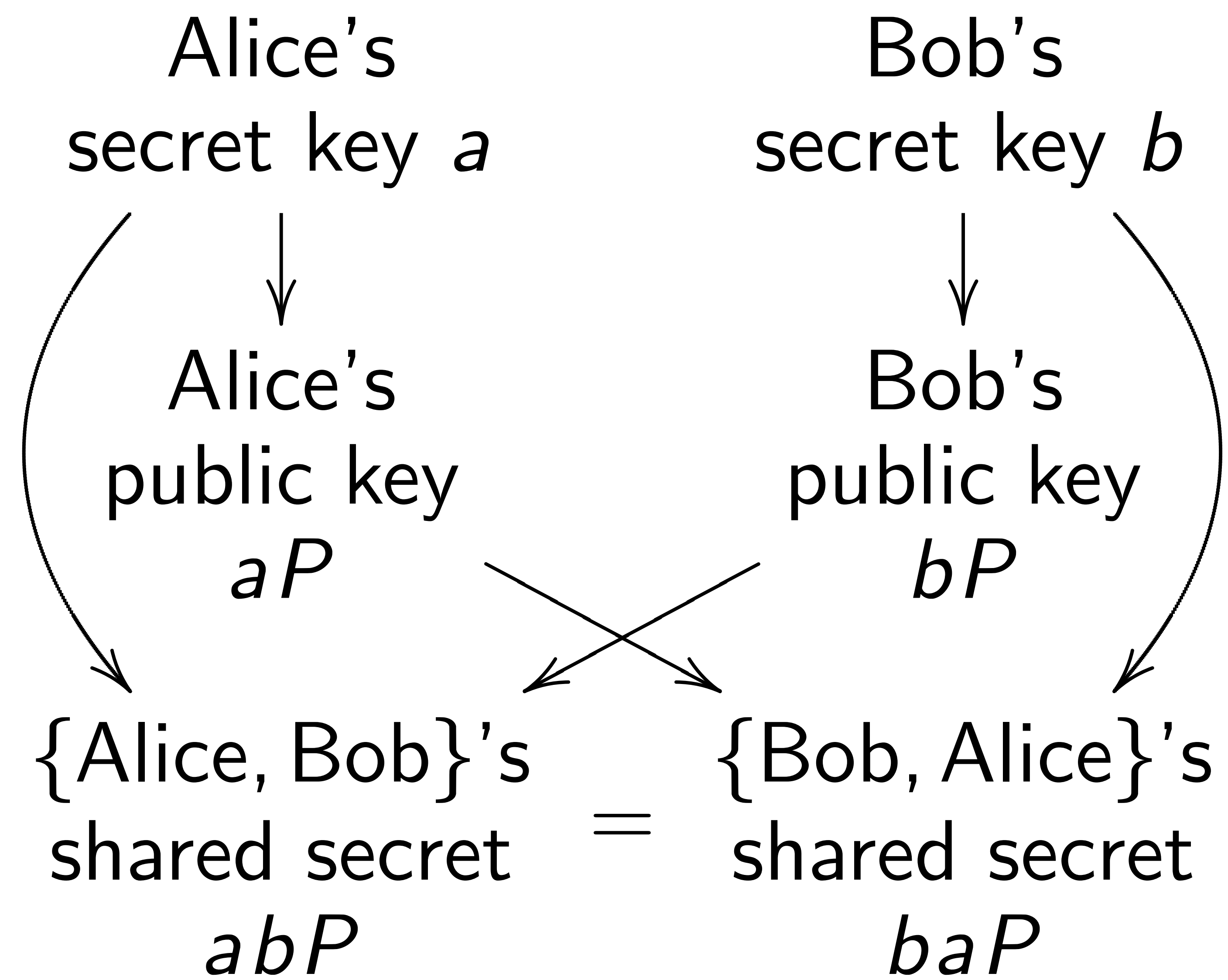
that plague finite-field DL.

Diffie-Hellman key exchange

Pick some *generator* P,

i.e. some group element

(using additive notation here).



What does P look like &

how to compute $P + Q$?

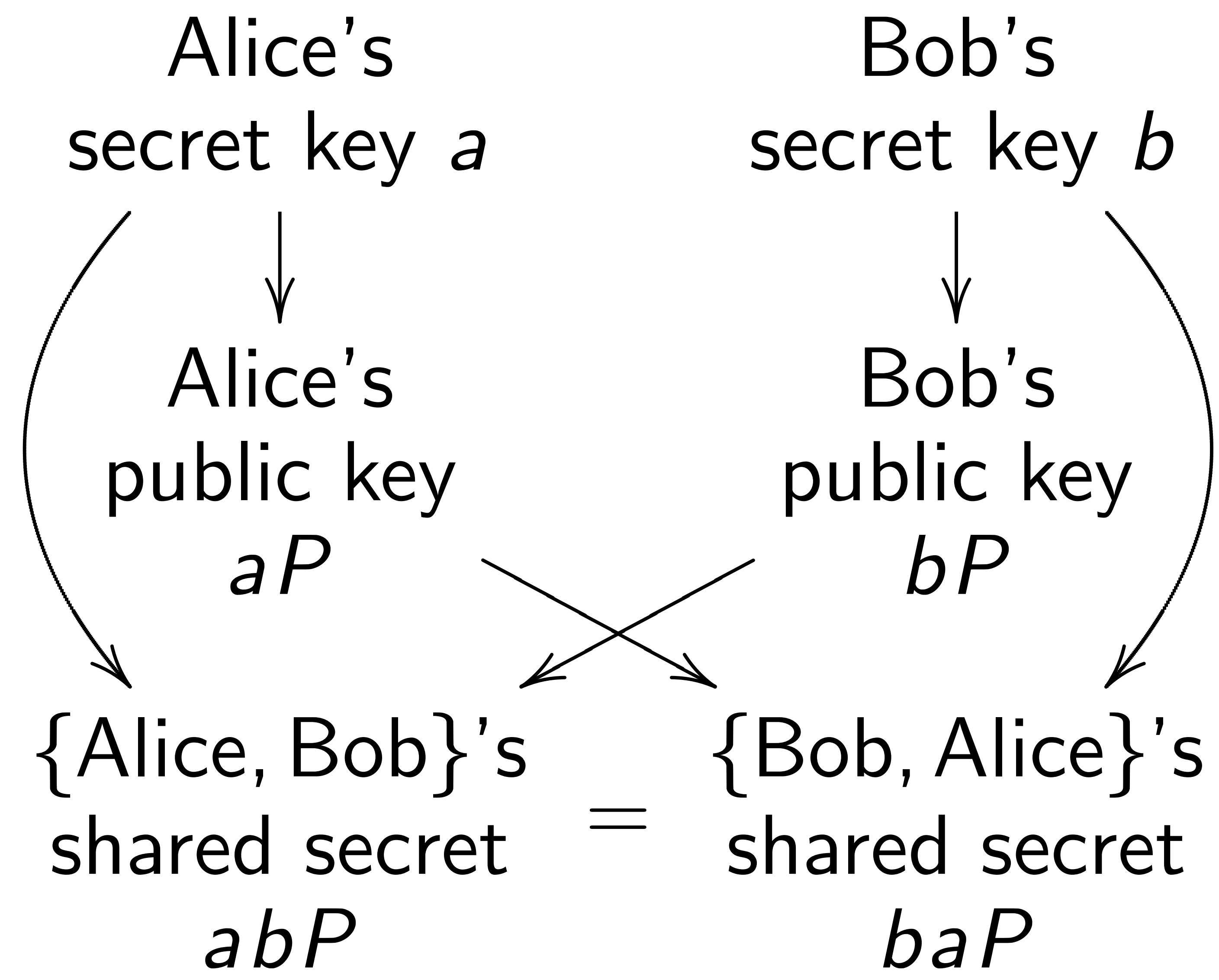
al of this course:
the attackers.
t to break ECC and RSA.

ed to understand ECC.

otivation for ECC:
dex-calculus attacks
gue finite-field DL.

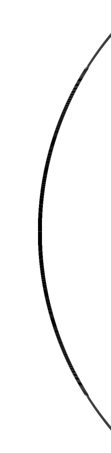
Diffie-Hellman key exchange

Pick some *generator* P,
i.e. some group element
(using additive notation here).



What does P look like &
how to compute $P + Q$?

The cloc



This is t

Warning

This is *r*

"Elliptic

course:

ers.

ECC and RSA.

erstand ECC.

or ECC:

us attacks

field DL.

Diffie-Hellman key exchange

Pick some *generator* P,
i.e. some group element
(using additive notation here).

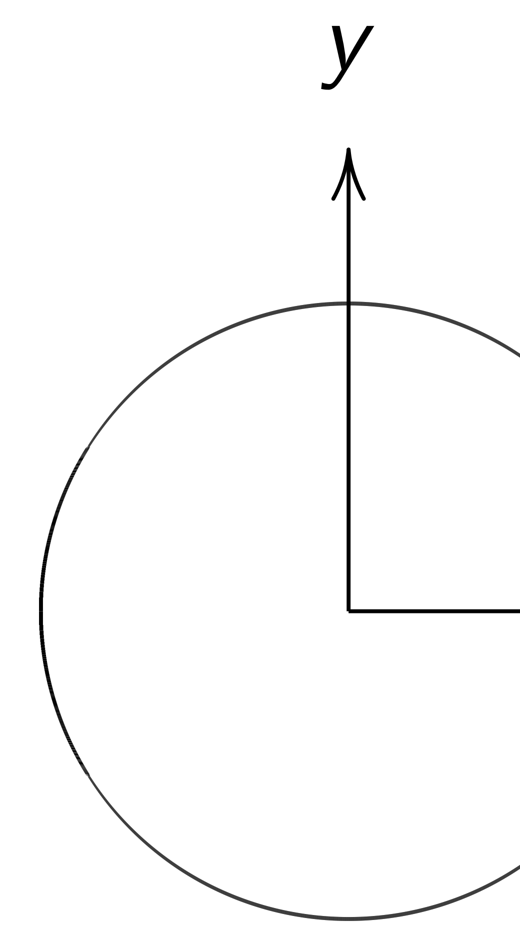
Alice's secret key a → Alice's public key aP

Bob's secret key b → Bob's public key bP

{Alice, Bob}'s shared secret abP = {Bob, Alice}'s shared secret baP

What does P look like &
how to compute $P+Q$?

The clock



This is the curve x

Warning:

This is *not* an ellip

"Elliptic curve" ≠

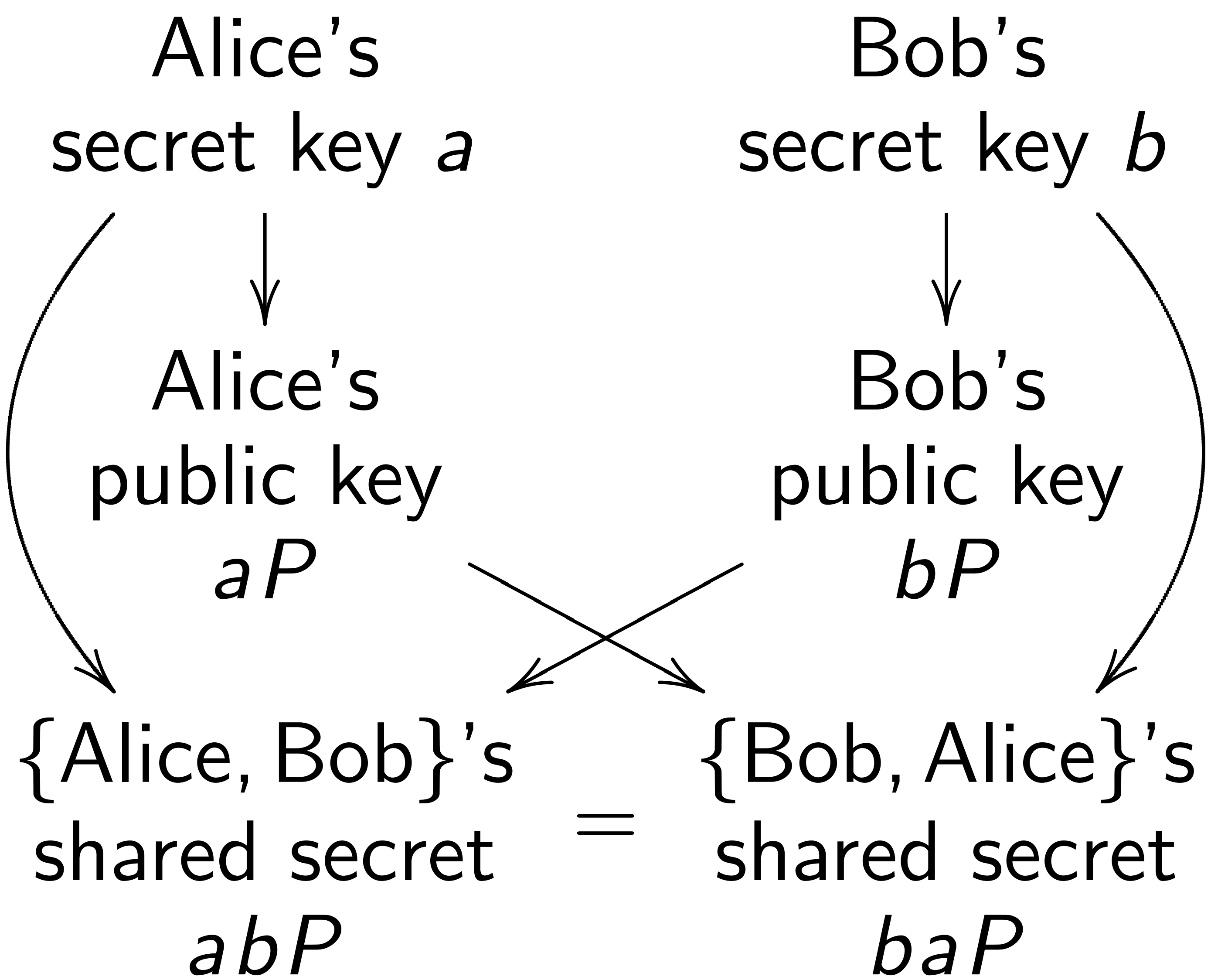
RSA.

C.

s

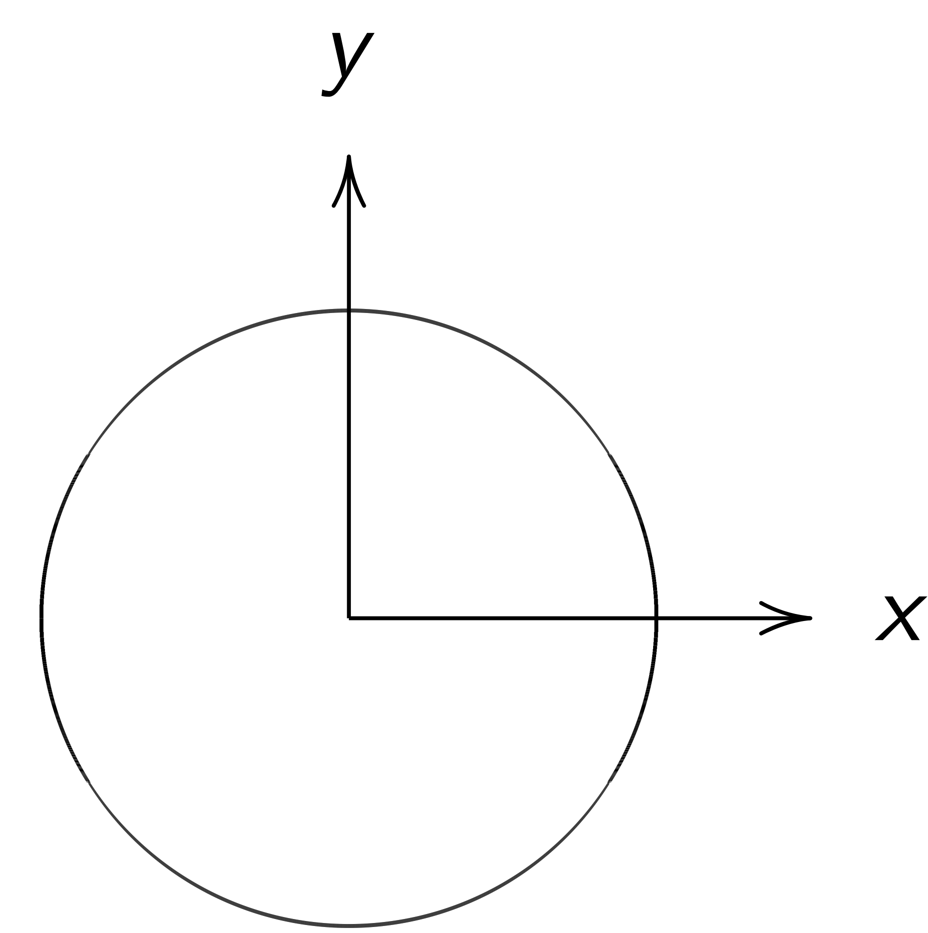
Diffie-Hellman key exchange

Pick some *generator* P,
i.e. some group element
(using additive notation here).



What does P look like &
how to compute $P + Q$?

The clock

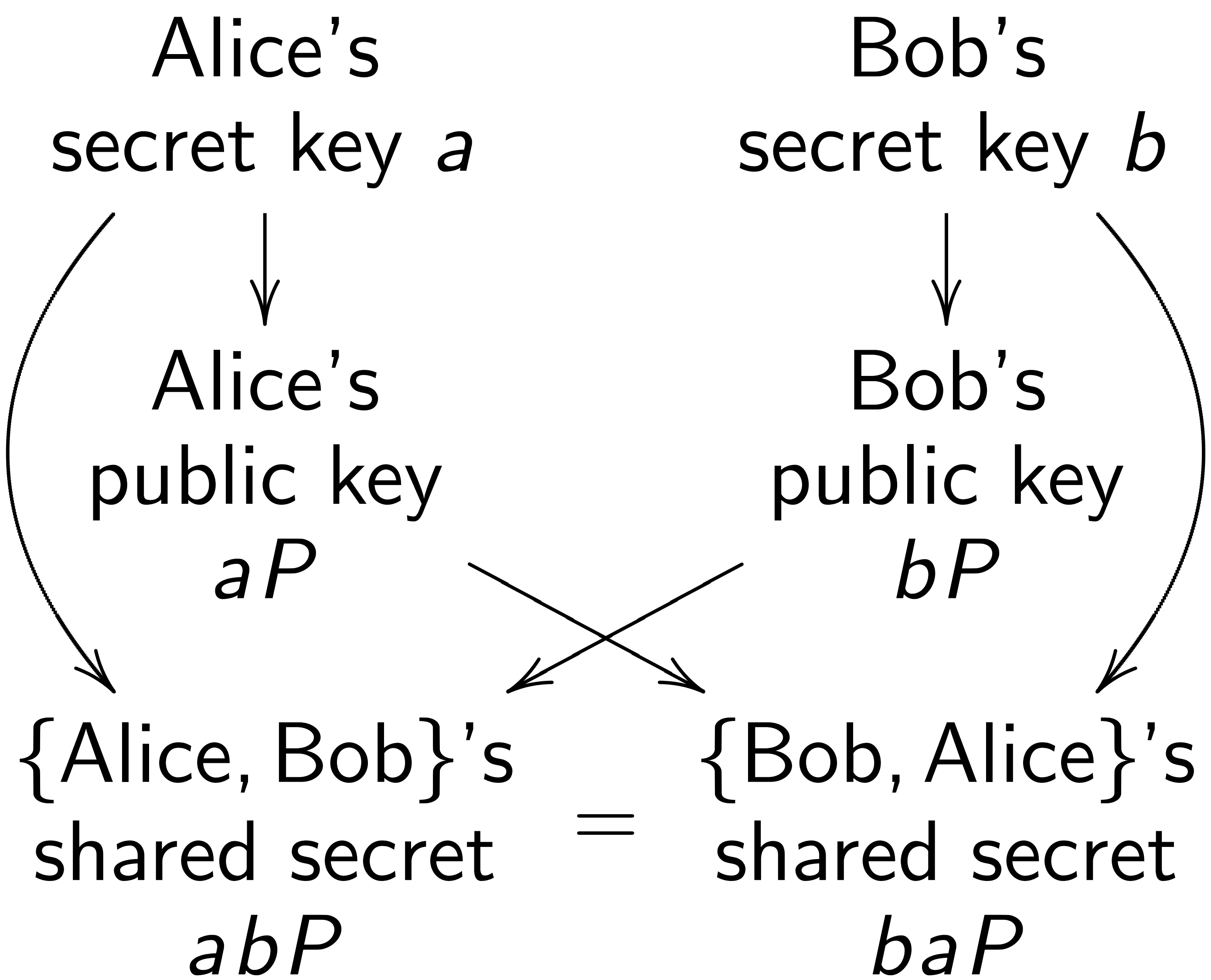


This is the curve $x^2 + y^2 =$

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

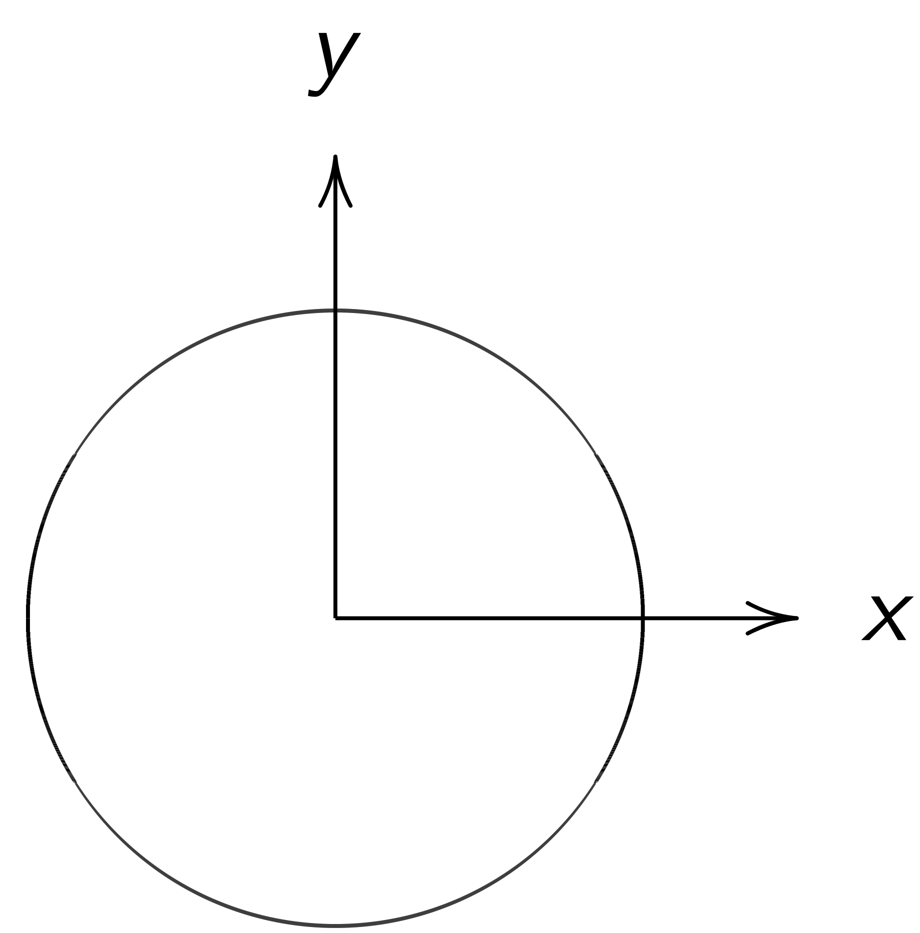
Diffie-Hellman key exchange

Pick some *generator* P,
i.e. some group element
(using additive notation here).



What does P look like &
how to compute $P + Q$?

The clock



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

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

ellman key exchange

ne *generator* P,

e group element

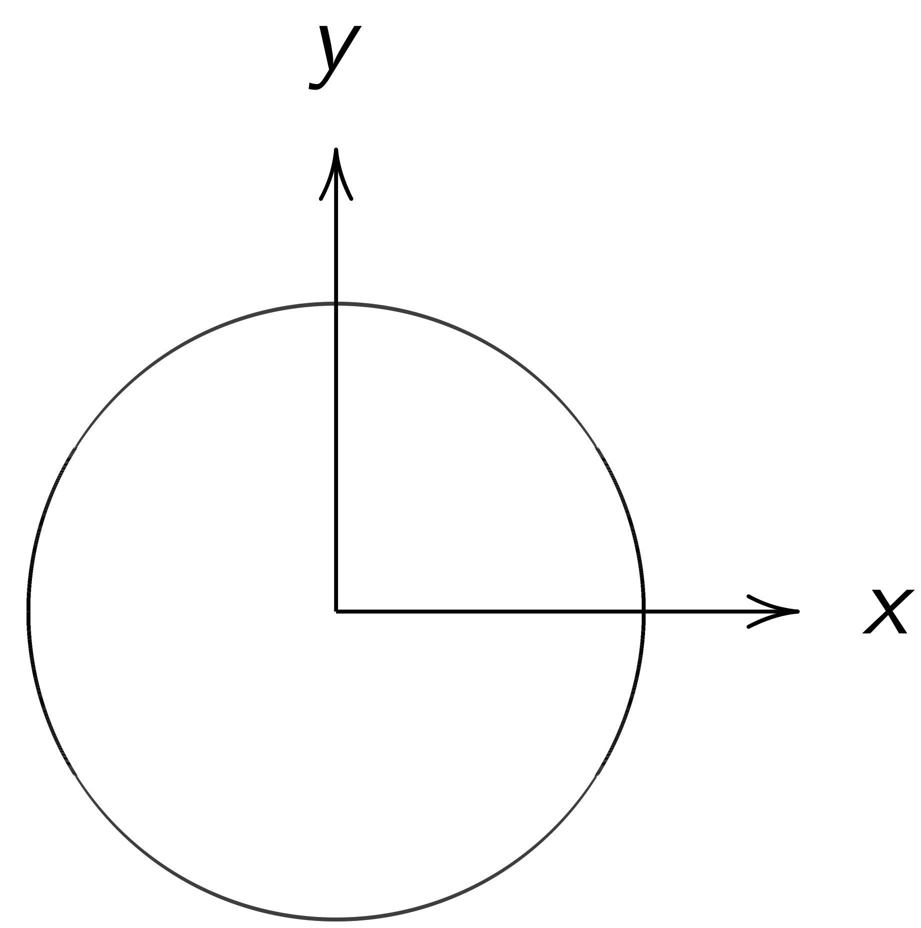
dditive notation here).

e's key a		Bob's secret key b
e's c key P		Bob's public key bP
Bob}'s secret P	$=$	{Bob, Alice}'s shared secret baP

oes P look like &

compute $P+Q$?

The clock



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

Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Example

y exchange

tor P,

ement

tation here).

Bob's
secret key b

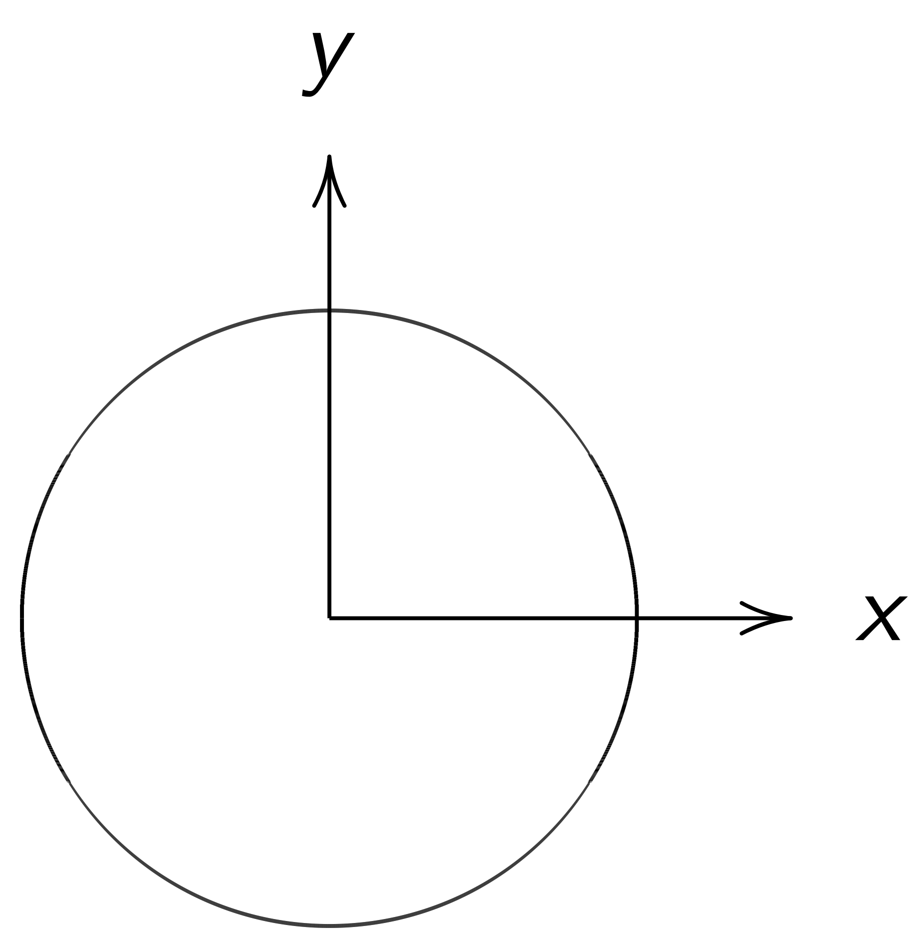
Bob's
public key
bP

{Bob, Alice}'s
shared secret
baP

k like &

$P + Q$?

The clock



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

Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of point

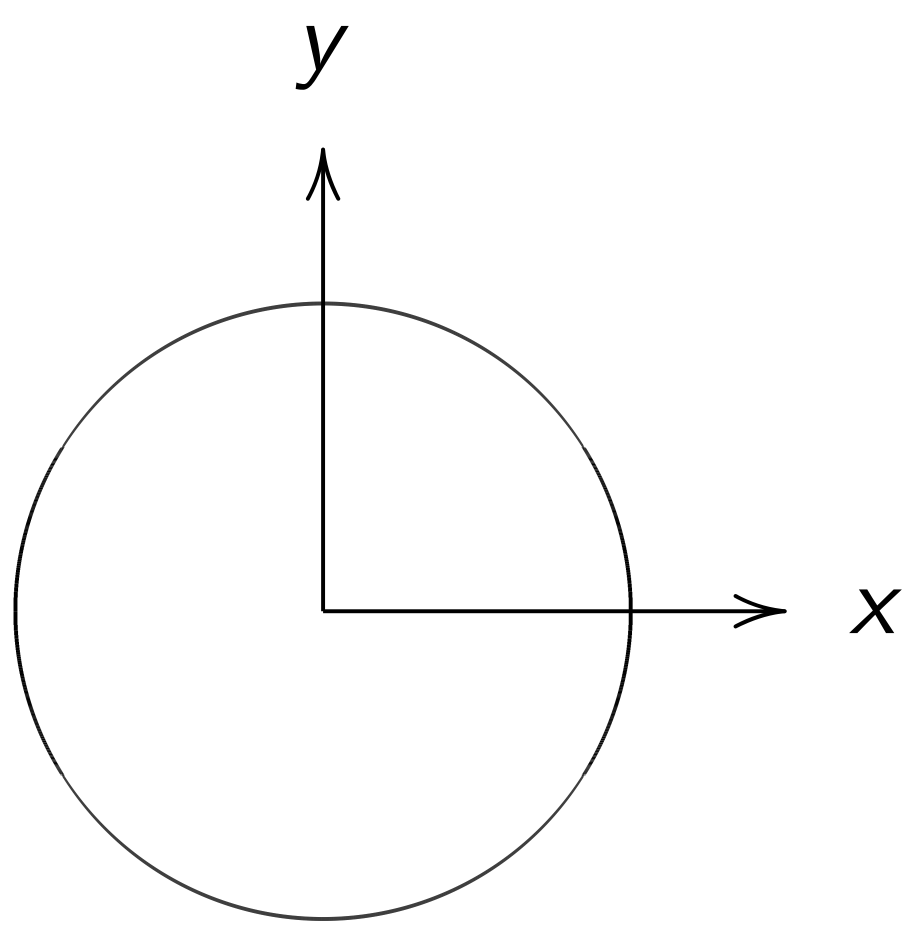
e

e).
's
key b

's
key

lice}'s
secret

The clock



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

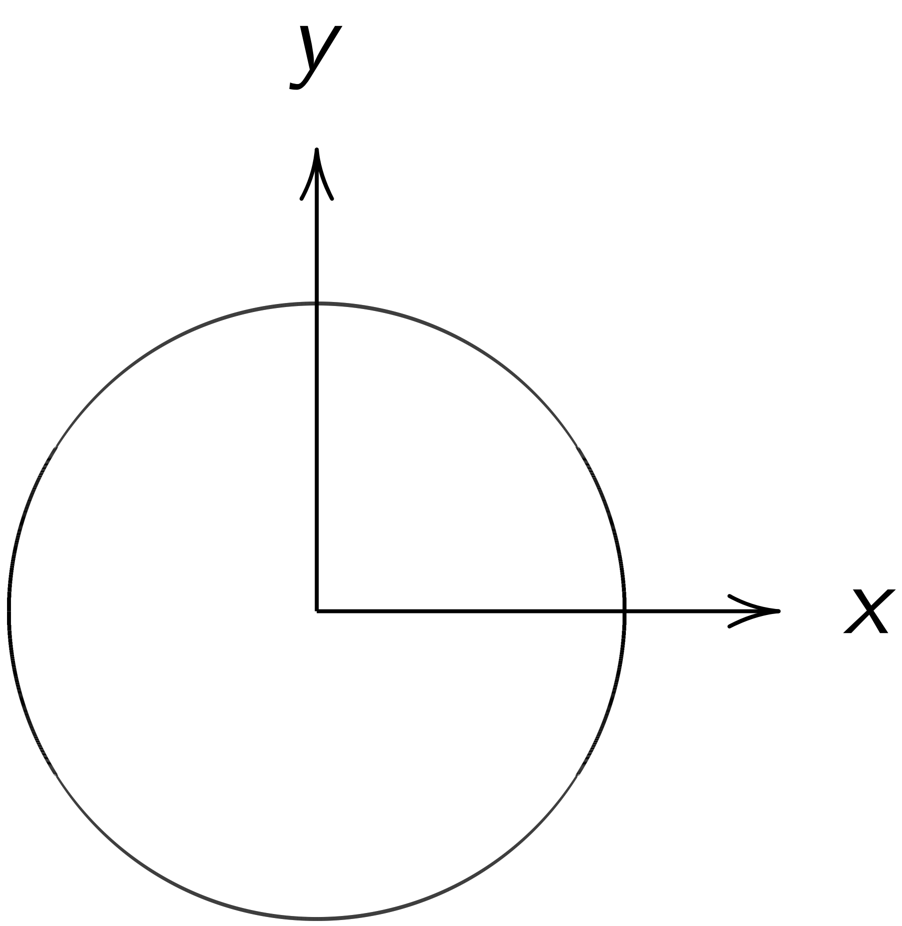
Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this c

The clock



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

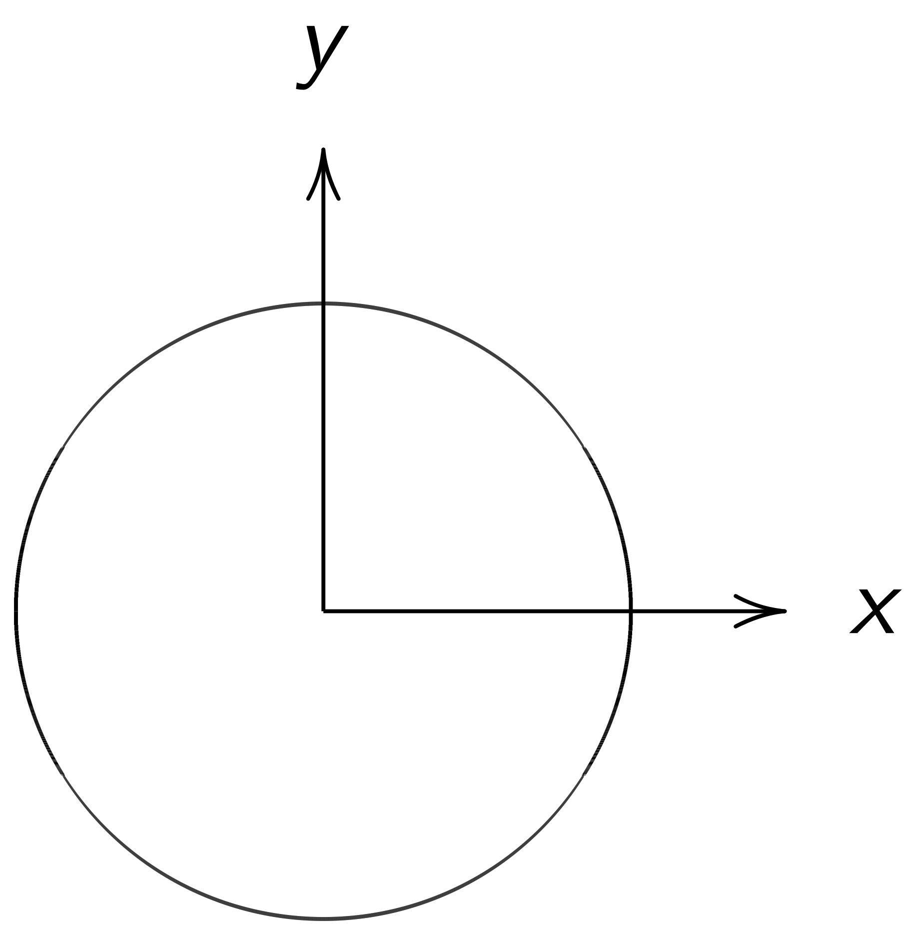
Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

The clock



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

Warning:

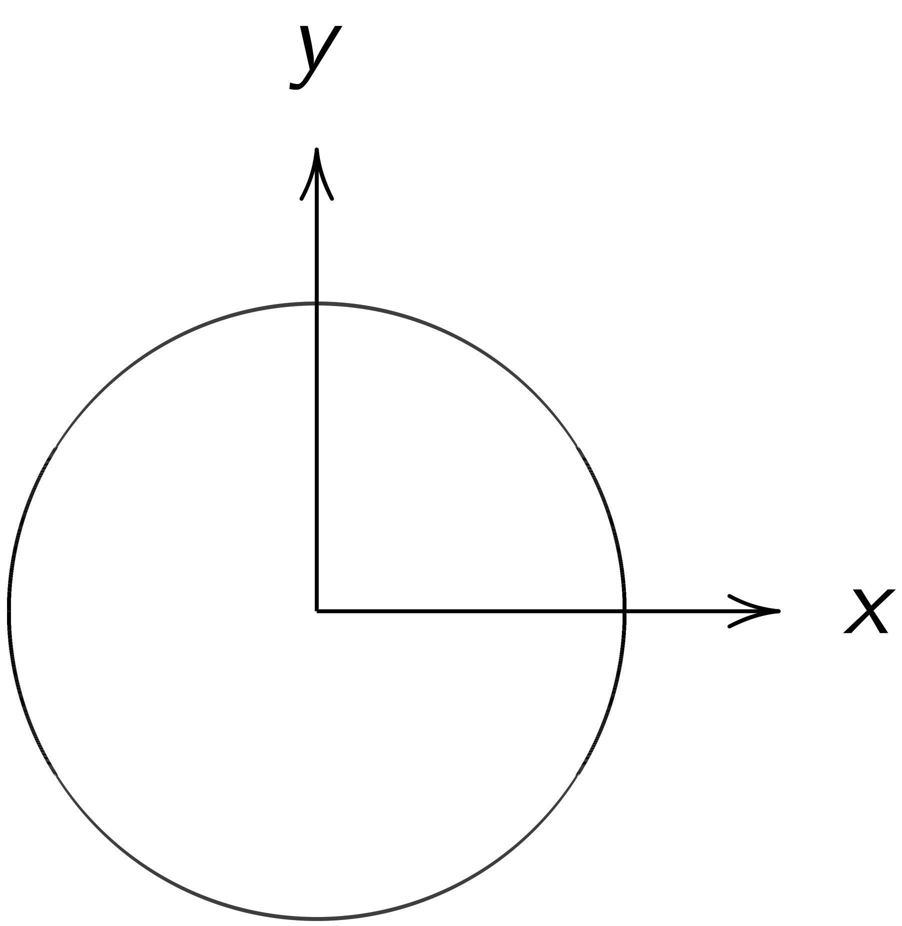
This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

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

The clock



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

Warning:

This is *not* an elliptic curve.

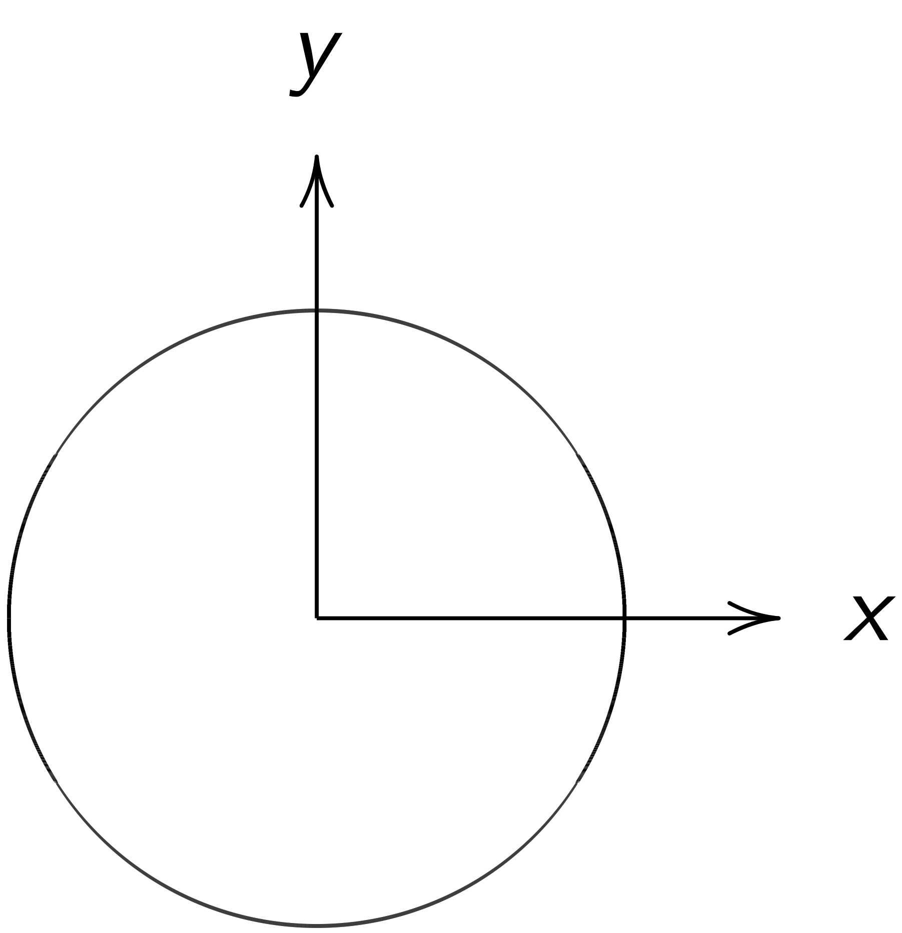
"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

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

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

The clock



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

Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

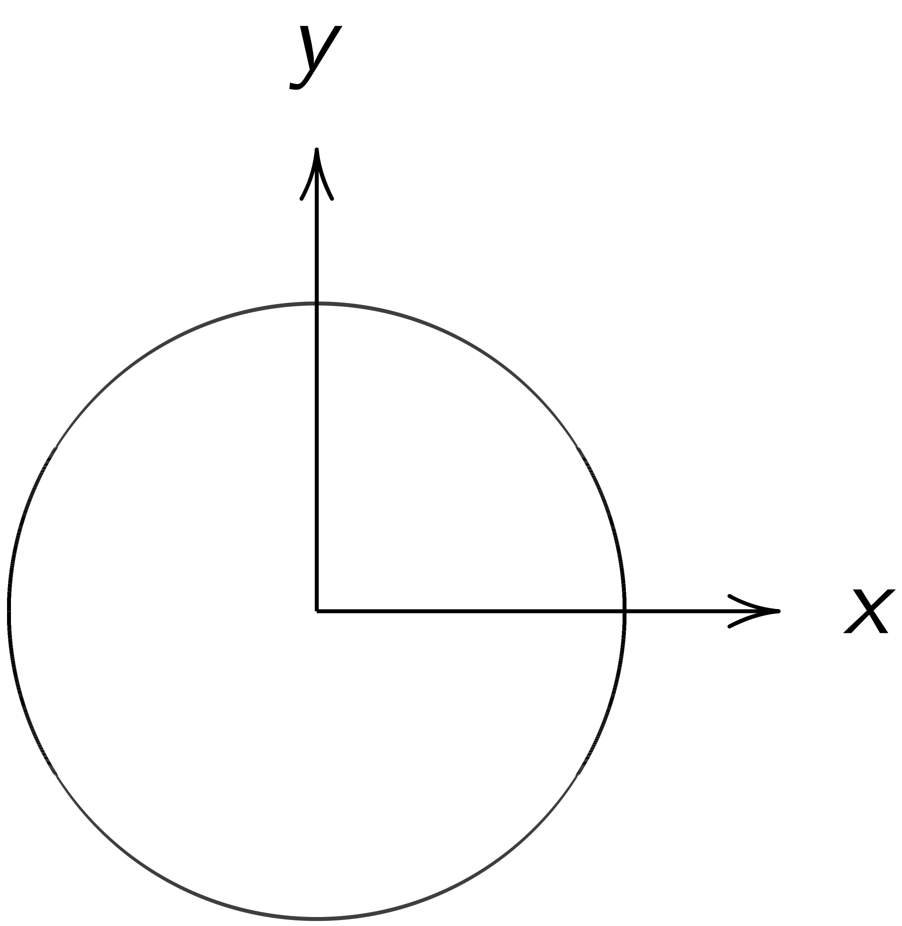
Examples of points on this curve:

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

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

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

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:

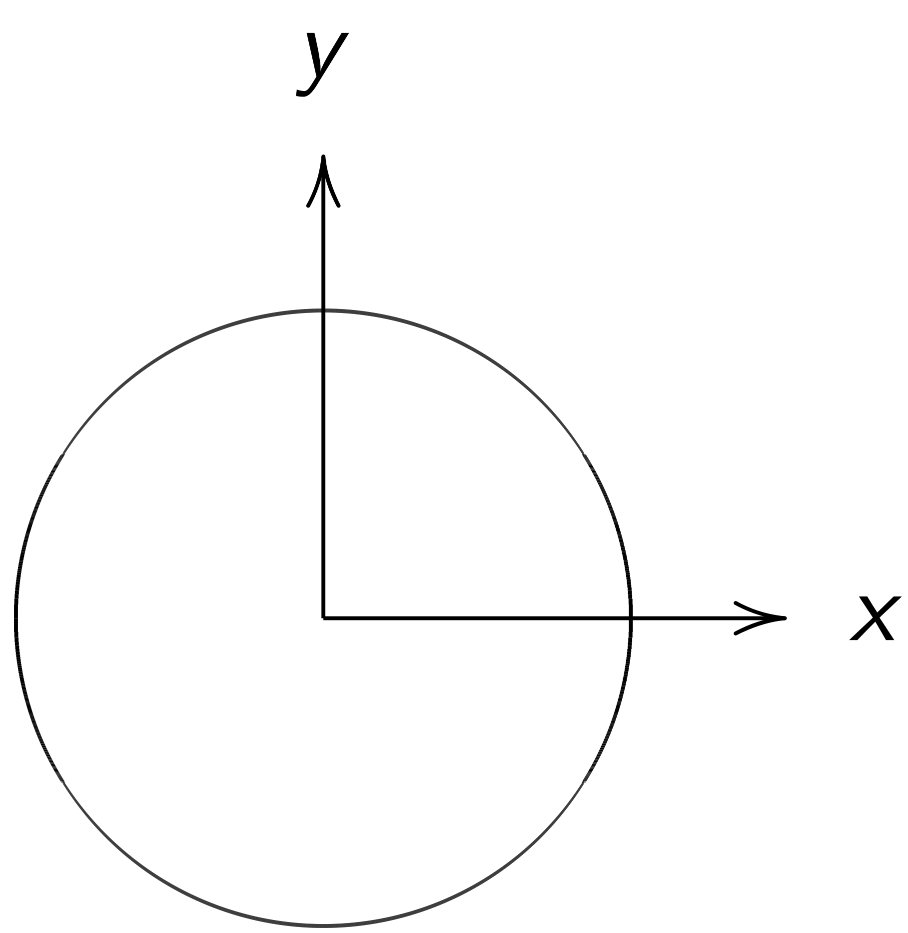
$(0, 1) =$ "12:00".

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

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

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

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:

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

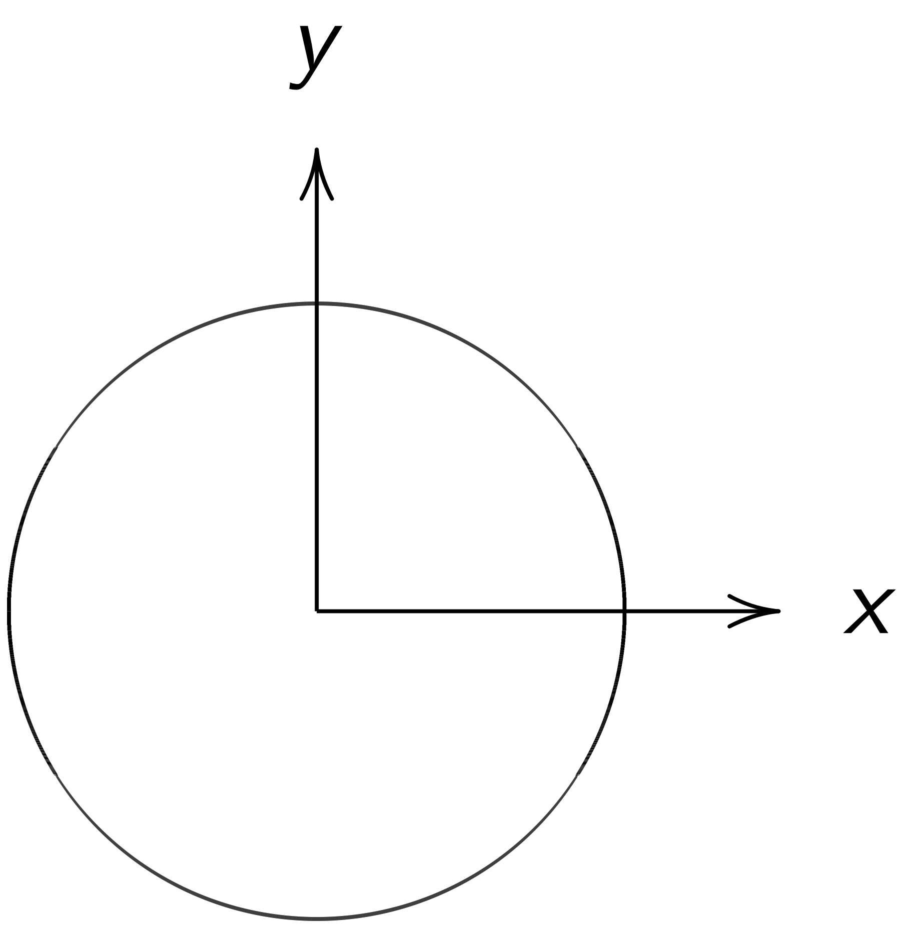
$(0, -1) =$ "6:00".

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

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

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

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:

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

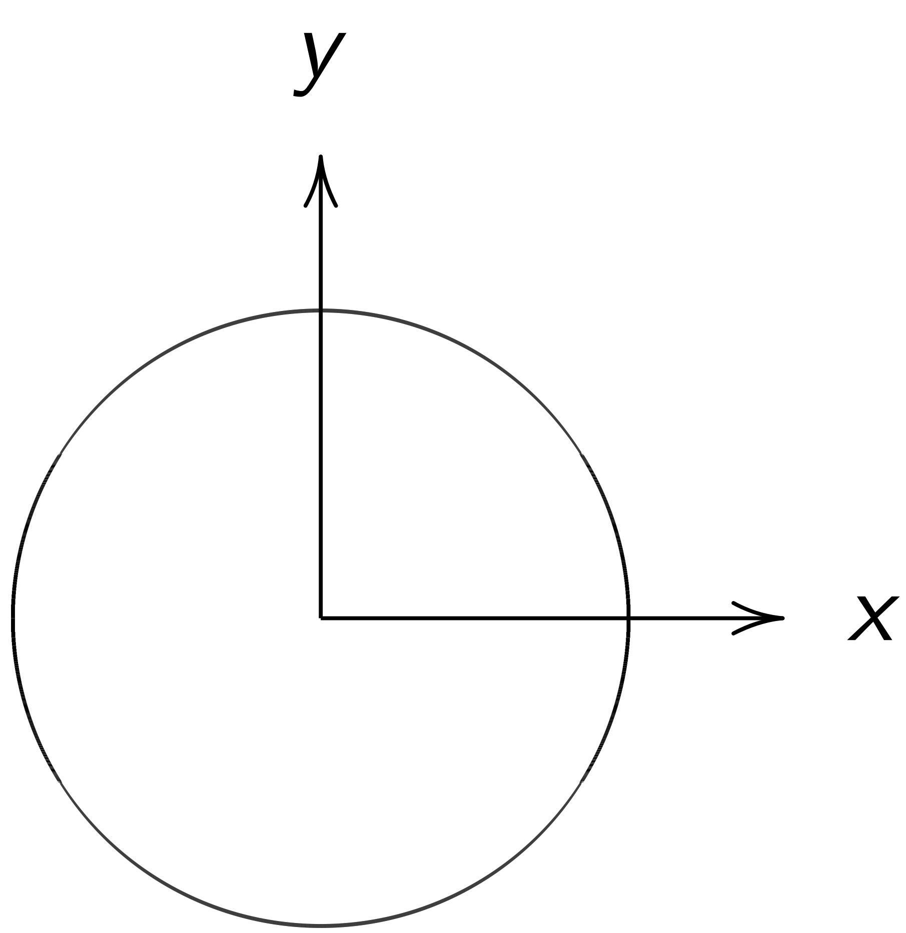
$(0, -1) =$ "6:00".

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

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

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

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:

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

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

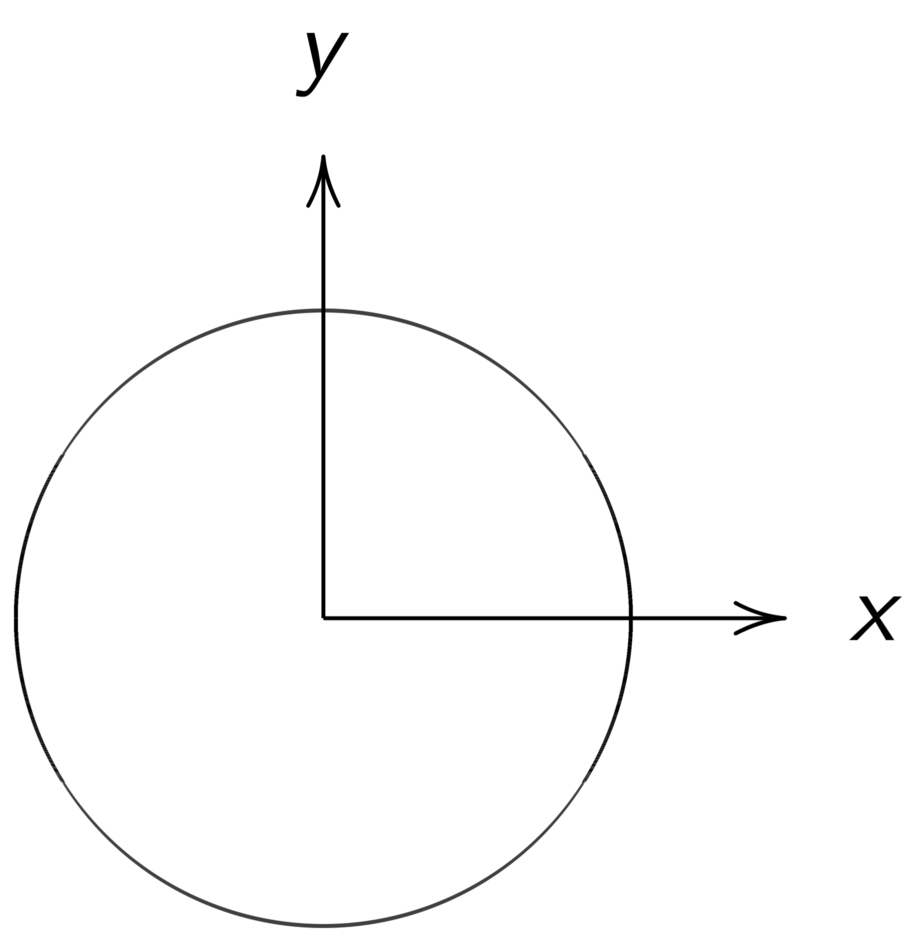
$(1, 0) =$ "3:00".

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

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

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

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:

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

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

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

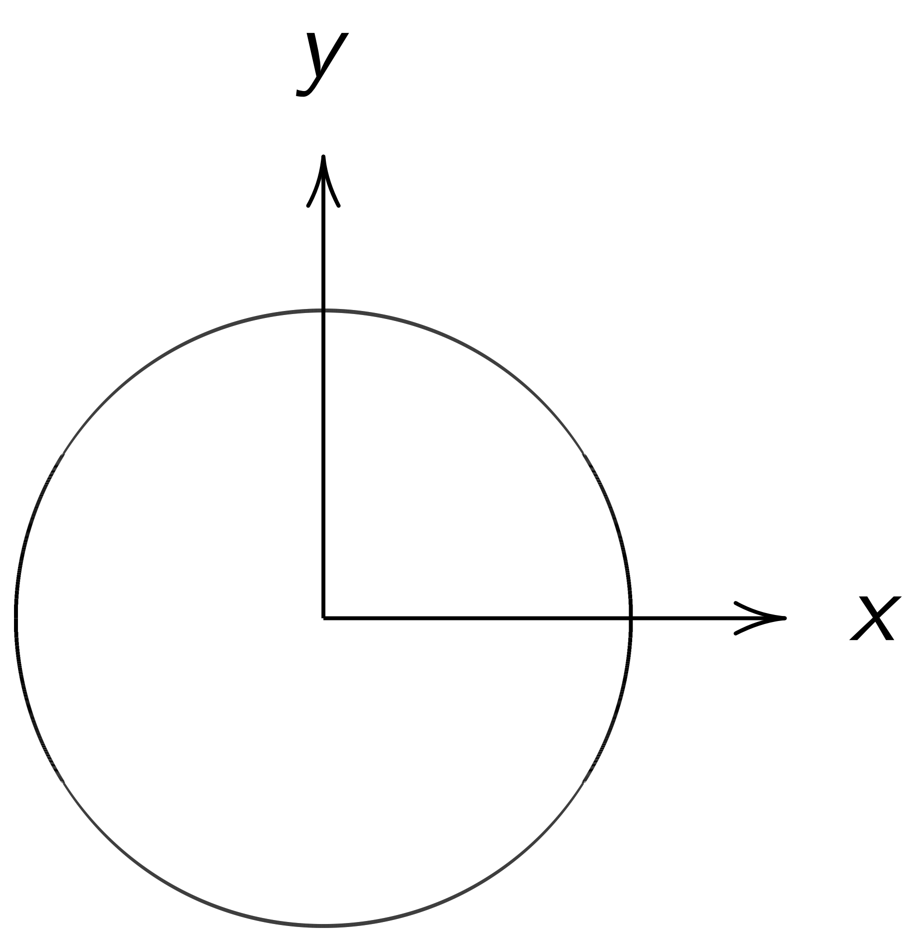
$(-1, 0) =$ "9:00".

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

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

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

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:

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

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

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

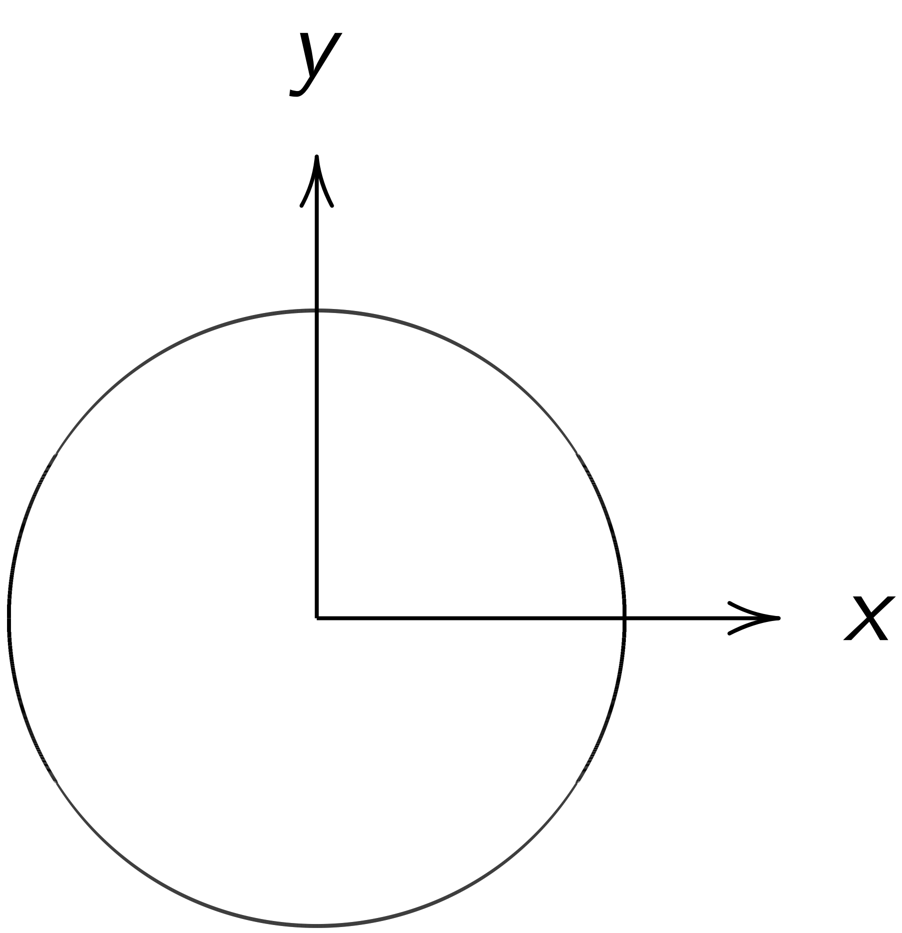
$(-1, 0) =$ "9:00".

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

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

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

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:

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

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

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

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

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

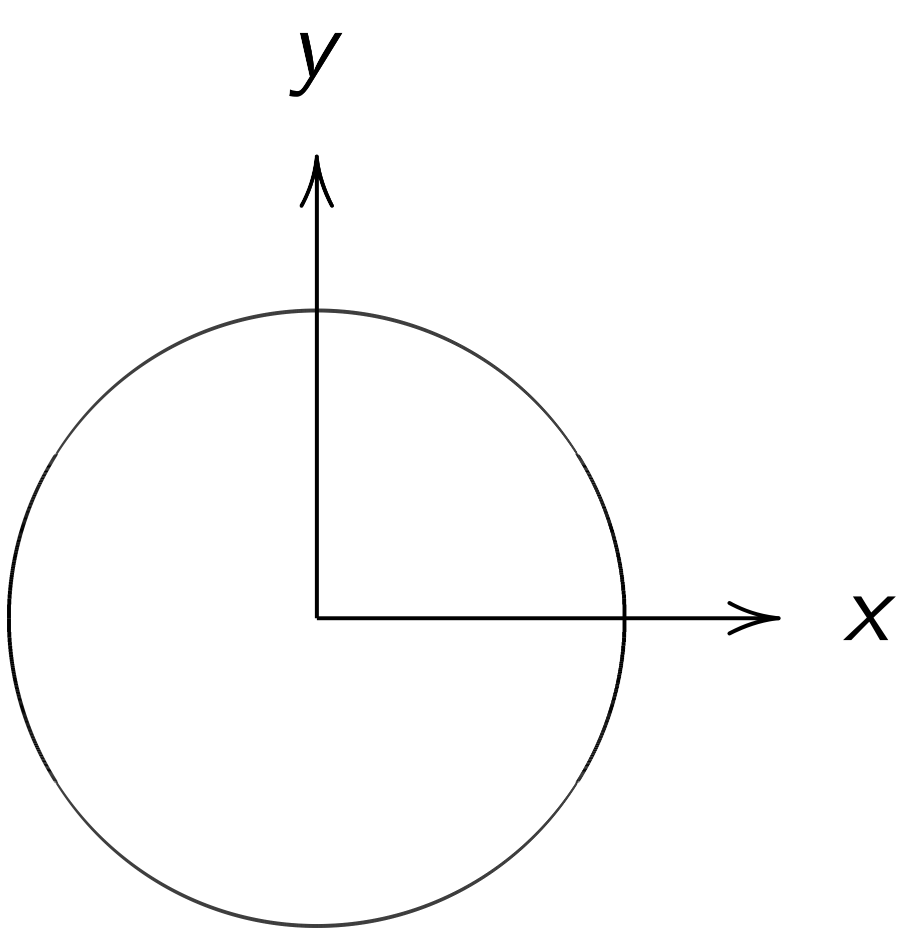
$(1/2, -\sqrt{3/4}) =$ "5:00".

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

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

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

The clock



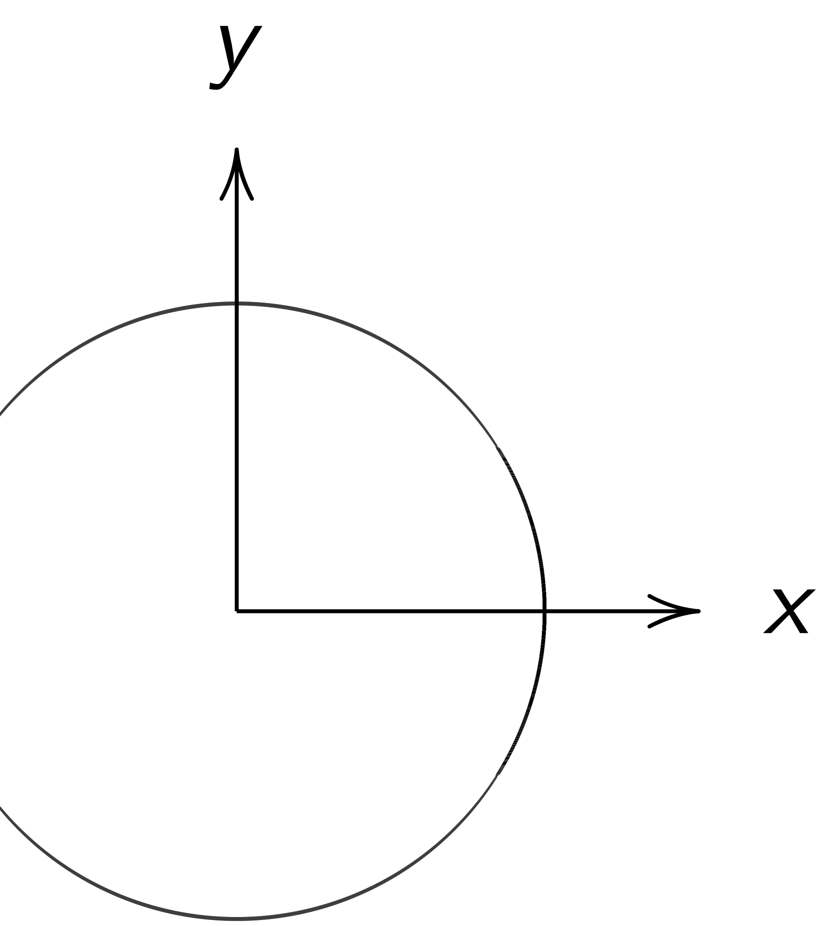
This is the curve $x^2 + y^2 = 1$.

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

Examples of points on this curve:

$(0, 1) =$ "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.

ck



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

s:

not an elliptic curve.

curve" ≠ "ellipse."

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".

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

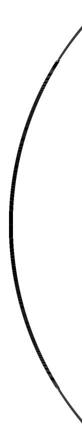
$(3/5, -4/5)$. $(-3/5, -4/5)$.

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

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

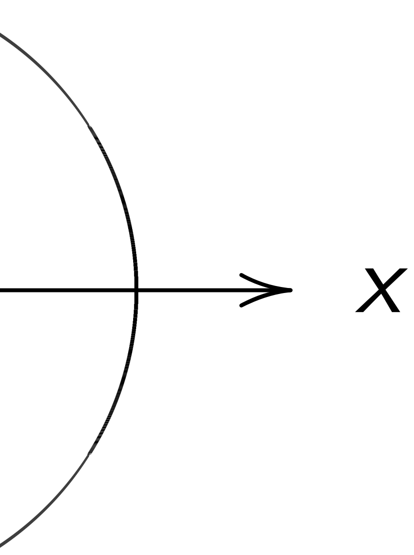
Many more.

Addition



$x^2 + y^2$

$x = \sin$



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

ptic curve.

"ellipse."

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".

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

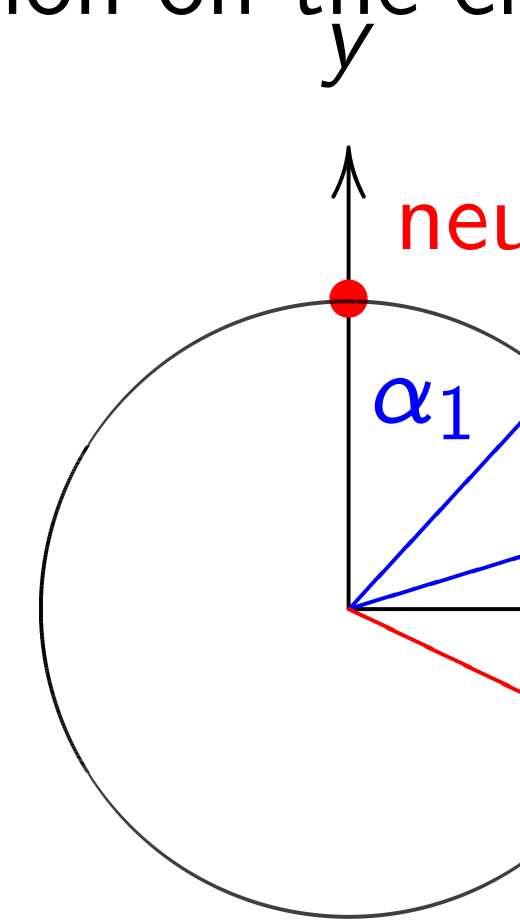
$(3/5, -4/5)$. $(-3/5, -4/5)$.

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

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

Many more.

Addition on the cl



$x^2 + y^2 = 1$, par

$x = \sin\alpha, \quad y = $ co

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".

1. $(3/5, 4/5)$. $(-3/5, 4/5)$.

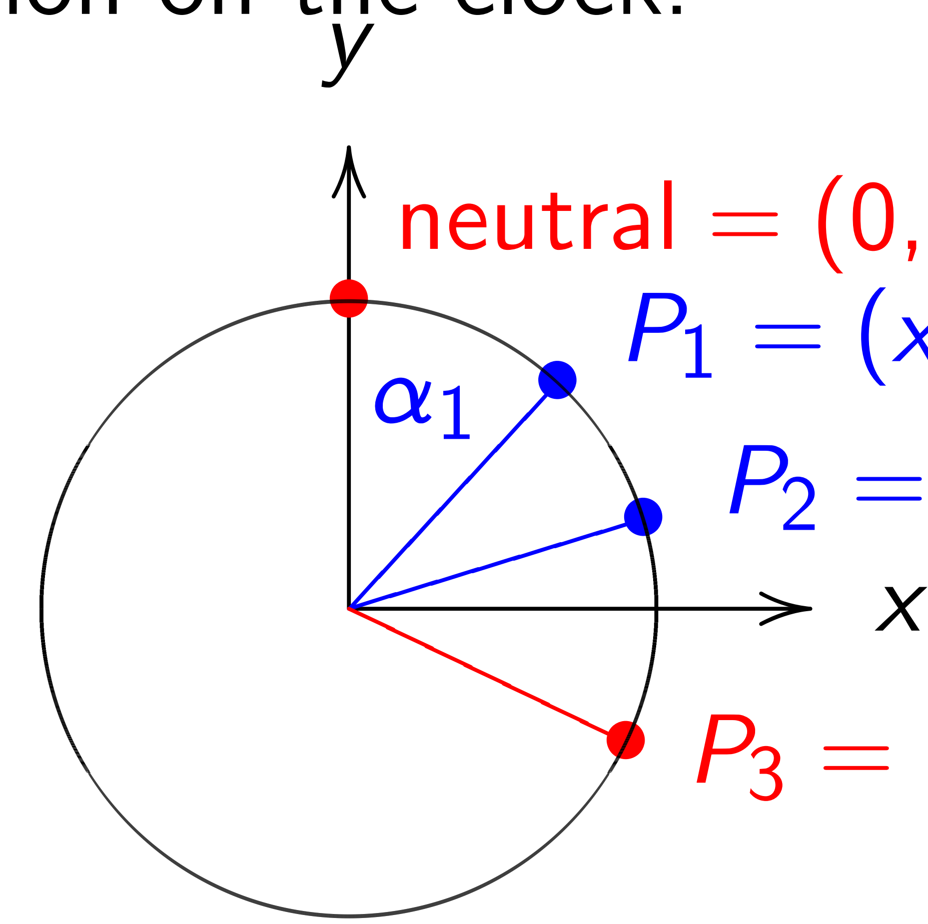
$(3/5, -4/5)$. $(-3/5, -4/5)$.

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

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

Many more.

Addition on the clock:



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

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

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".

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

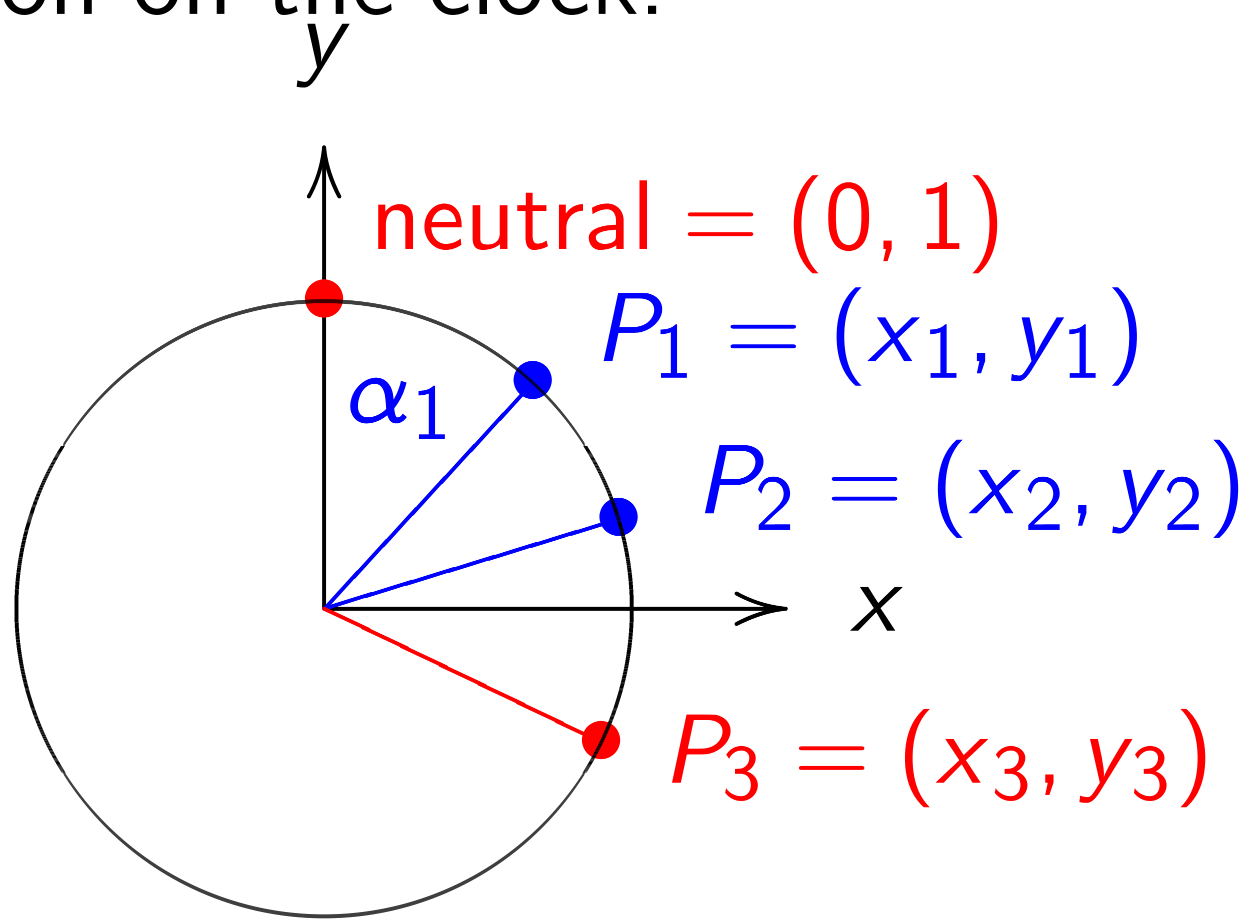
$(3/5, -4/5)$. $(-3/5, -4/5)$.

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

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

Many more.

Addition on the clock:



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

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

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".

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

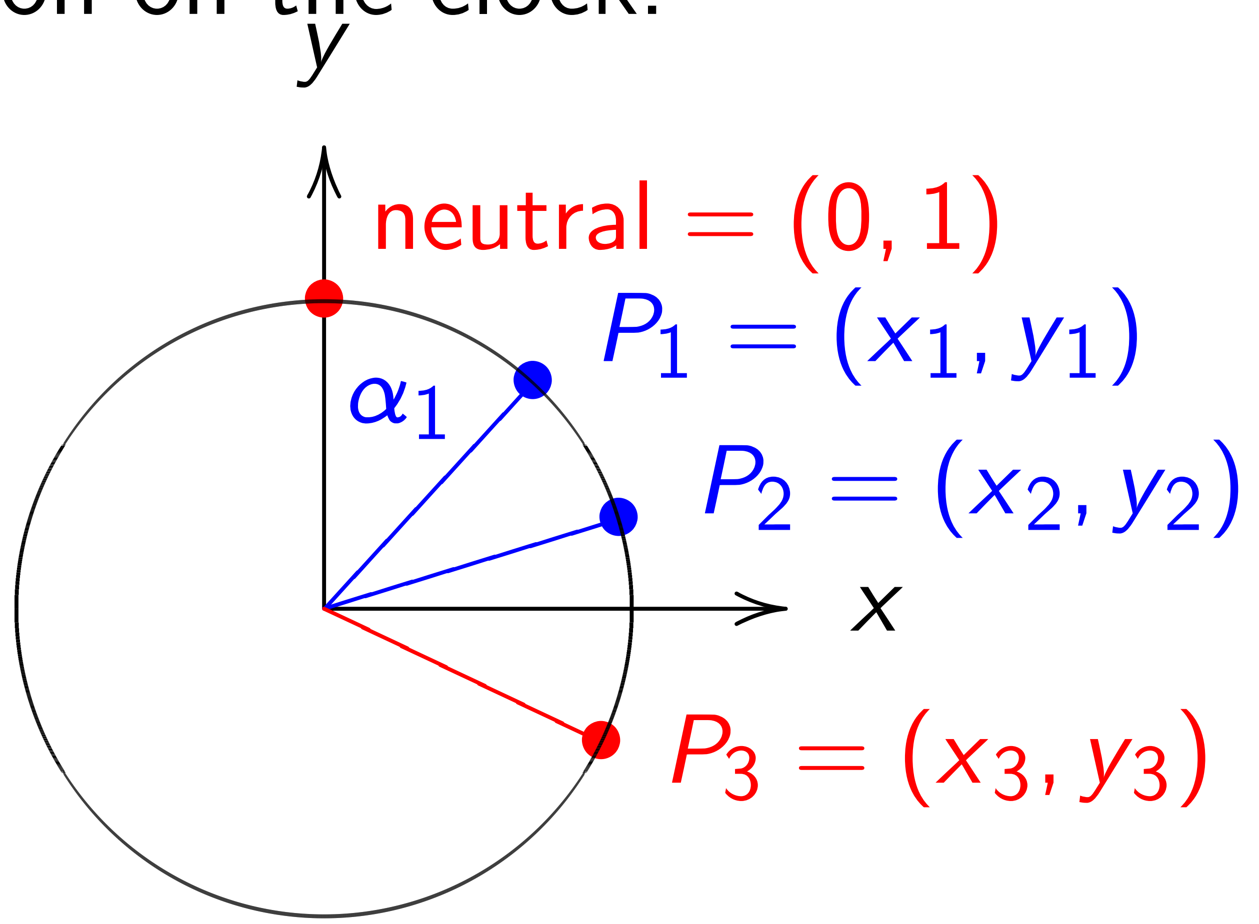
$(3/5, -4/5)$. $(-3/5, -4/5)$.

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

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

Many more.

Addition on the clock:



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

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

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

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".

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

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

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

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

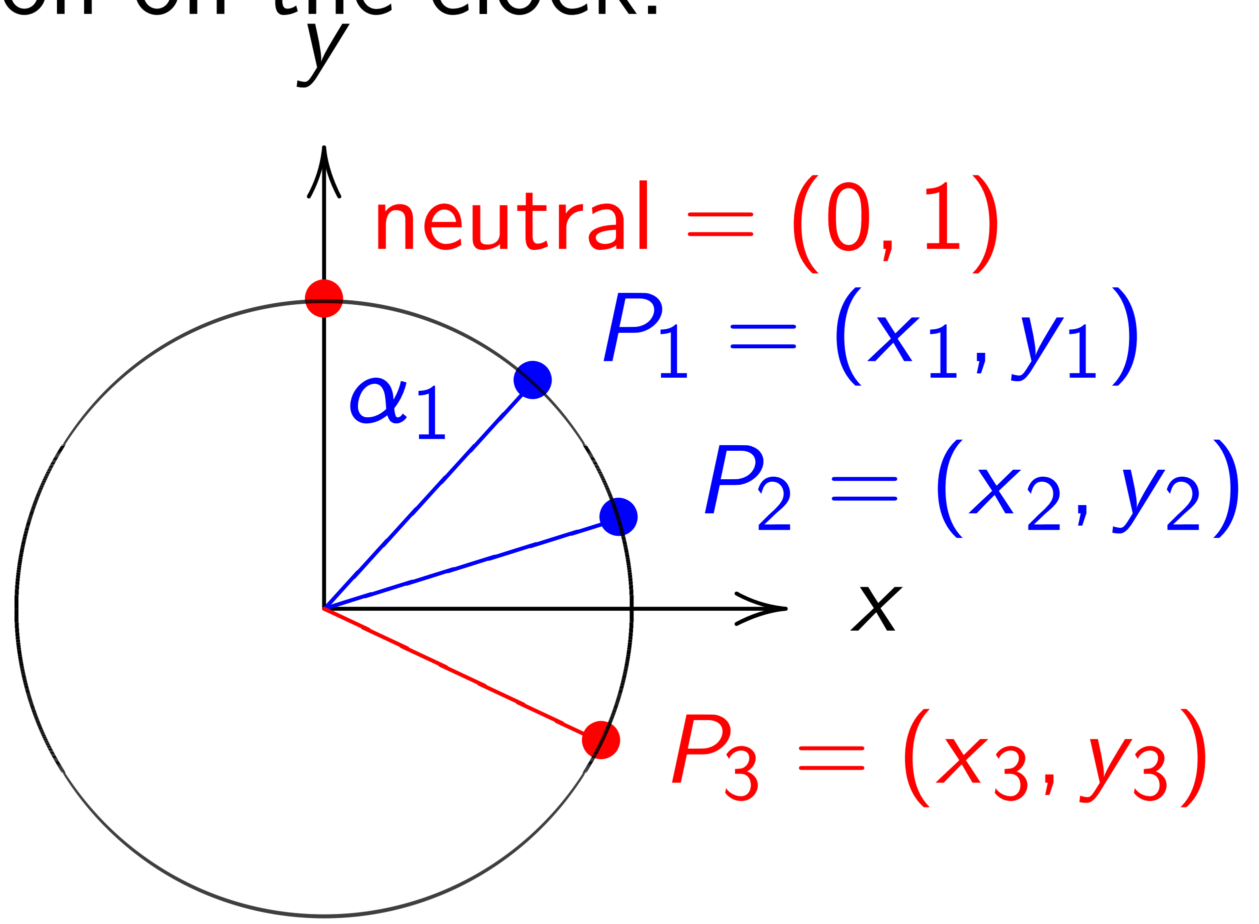
$(3/5, -4/5)$. $(-3/5, -4/5)$.

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

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

Many more.

Addition on the clock:



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

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

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

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

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".

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

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

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

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

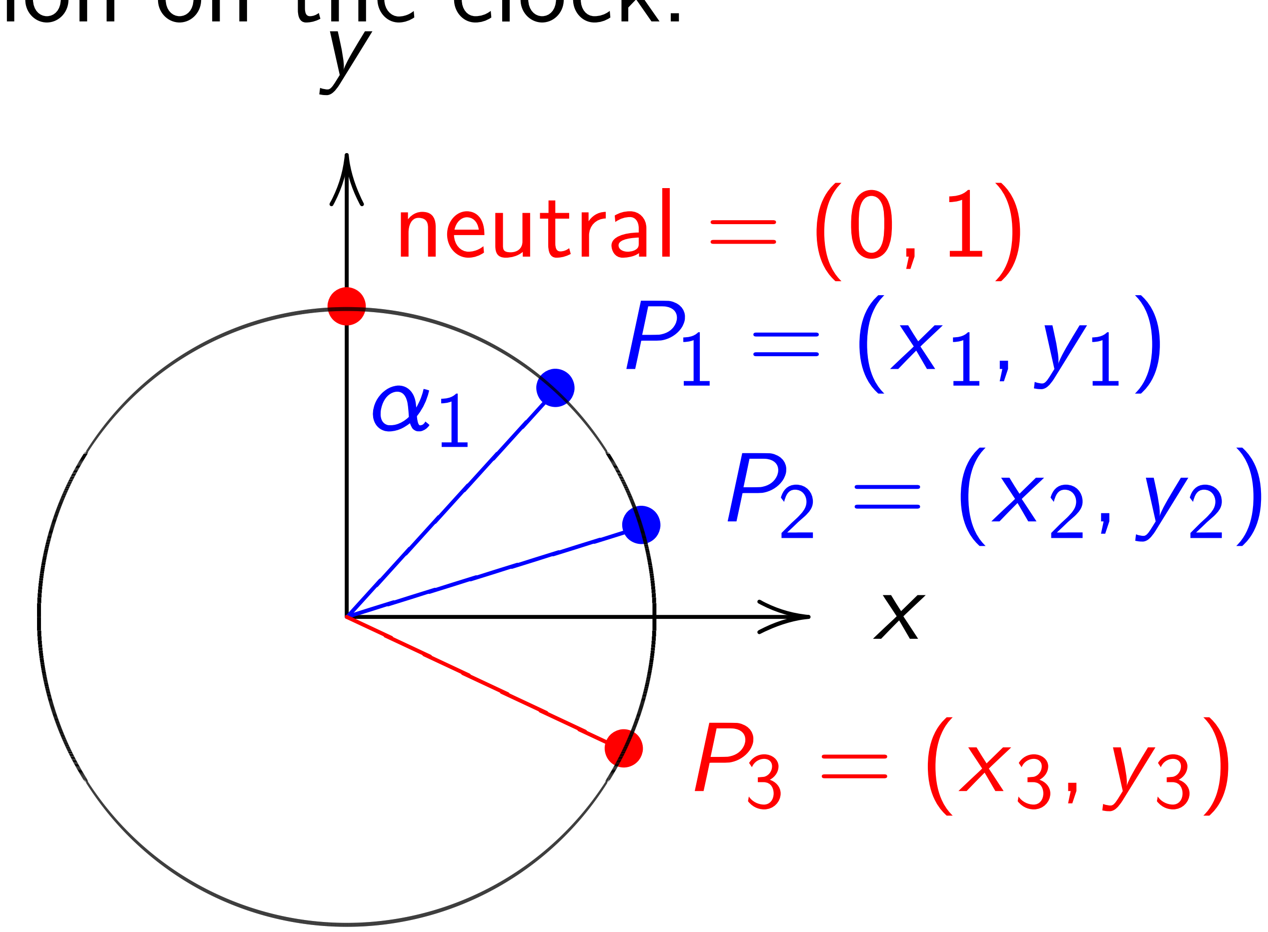
$(3/5, -4/5)$. $(-3/5, -4/5)$.

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

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

Many more.

Addition on the clock:



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

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

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

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

$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2)$.

es of points on this curve:

"12:00".

$=$ "6:00".

"3:00".

$=$ "9:00".

$1/2) =$ "2:00".

$\sqrt{3/4}) =$ "5:00".

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

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

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

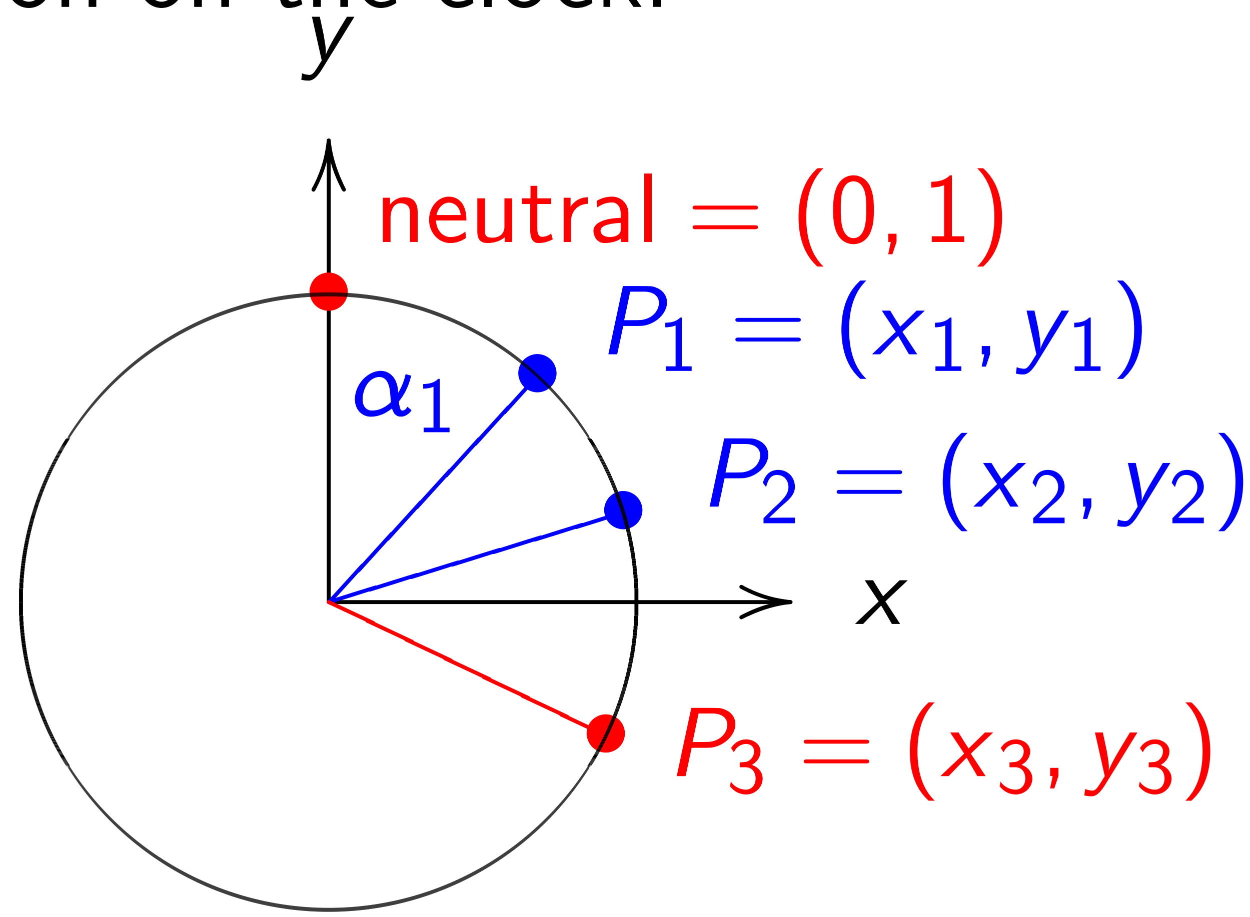
$4/5).\ (-3/5, -4/5).$

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

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

ore.

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 t

to addin

Angles r

so points

Neutral

point (0

The poir

has orde

3:00 and

Inverse o

is point

since α +

There ar

where ar

s on this curve:

:00".

"5:00".

= "7:00".

"1:30".

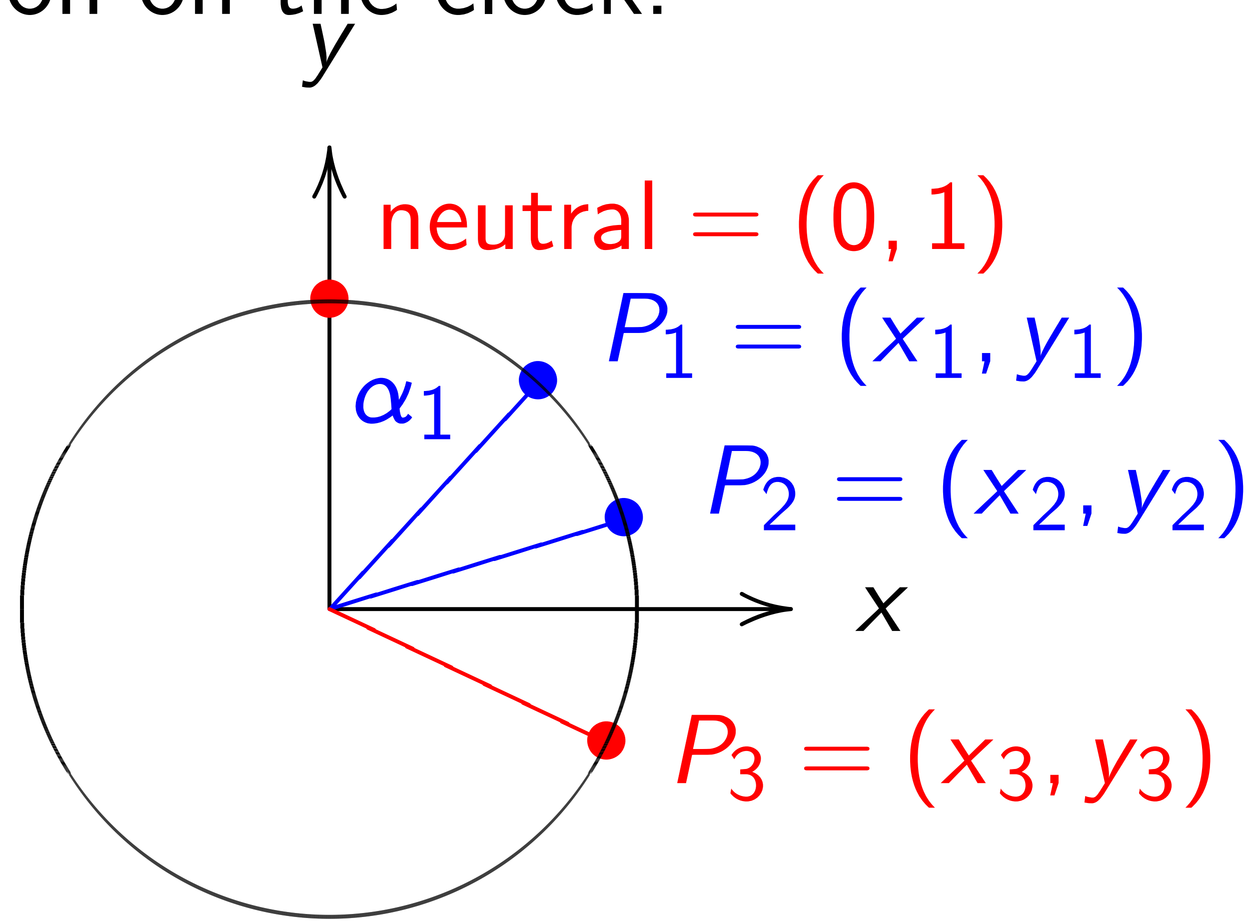
$, 4/5)$.

$/5, -4/5)$.

$, 3/5)$.

$/5, -3/5)$.

Addition on the clock:



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

Adding two points

to adding the angl

Angles modulo 36

so points on clock

Neutral element: a

point $(0, 1)$; "12:0

The point with α

has order 2 and e

3:00 and 9:00 hav

Inverse of point w

is point with $-\alpha$

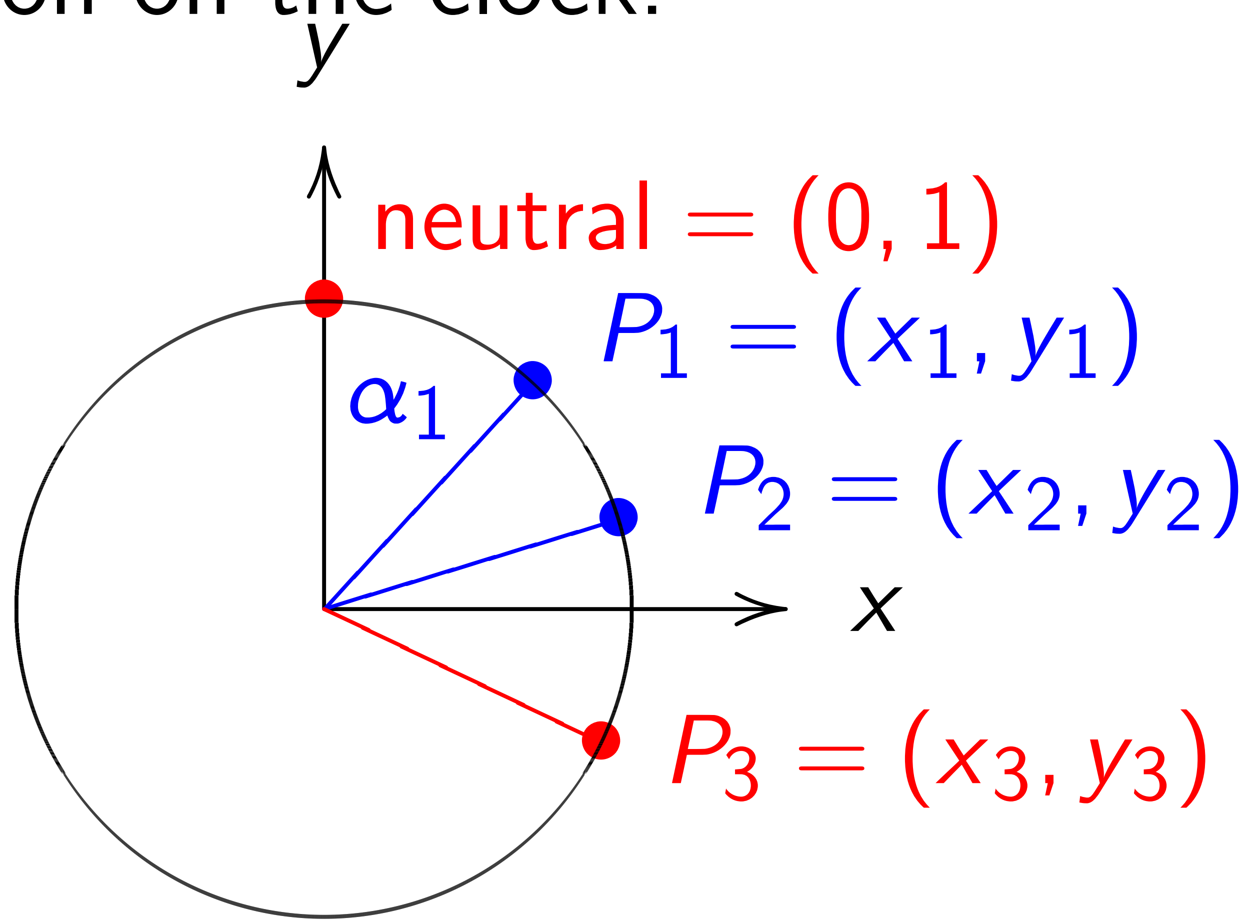
since $\alpha + (-\alpha) =$

There are many m

where angle α is r

curve:

Addition on the clock:

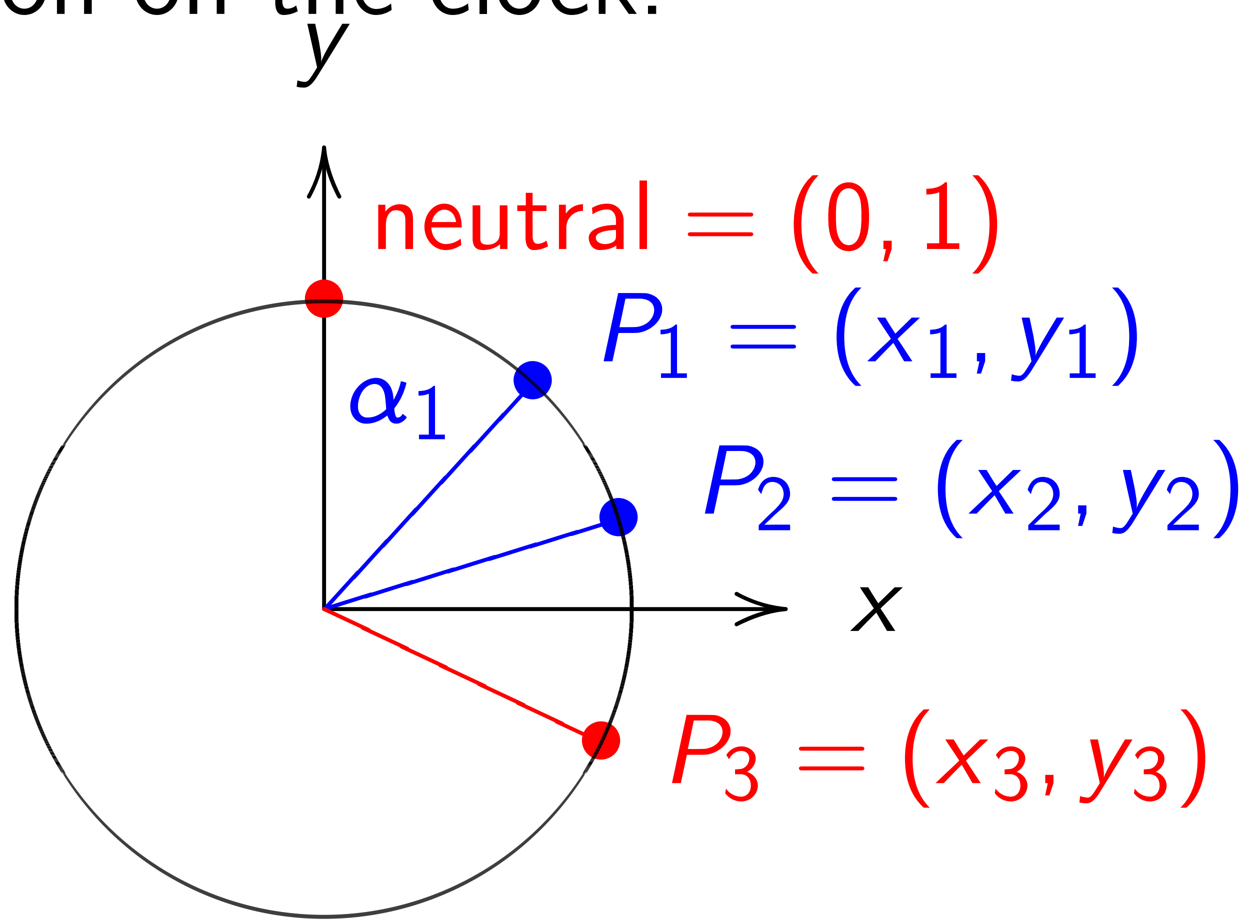


$x^2 + y^2 = 1$, parametrized by
$x = \sin\alpha$, $y = \cos\alpha$. Recall
$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$
$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2)$.

Adding two points correspon
to adding the angles α_1 and
Angles modulo 360° are a g
so points on clock are a gro

Neutral element: angle α =
point (0, 1); "12:00".
The point with $\alpha = 180°$
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."

Addition on the clock:



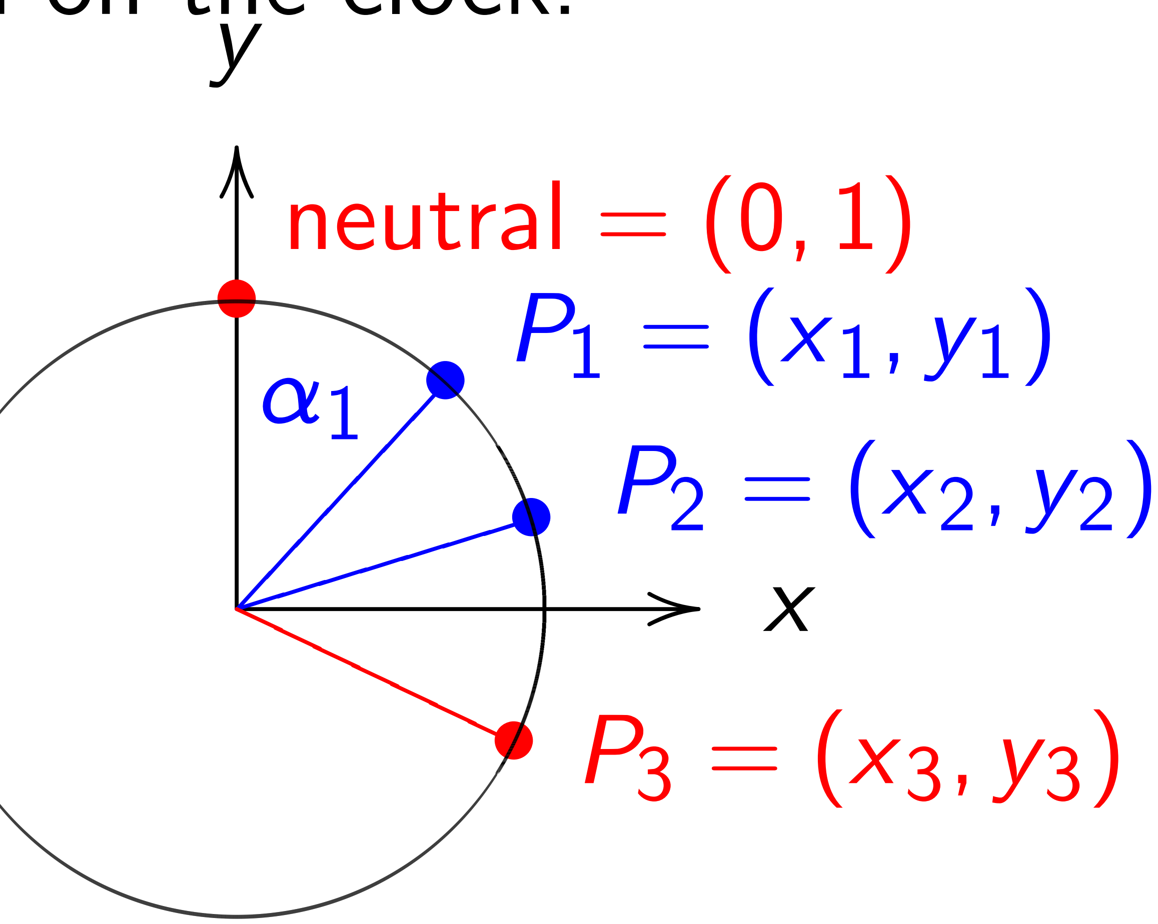
$x^2 + y^2 = 1$, parametrized by $x = \sin\alpha, \quad y = \cos\alpha$. Recall

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

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."

n on the clock:



$= 1$, parametrized by

α, $\quad y = \cos\alpha$. Recall

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

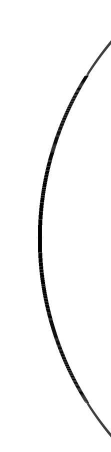
$\text{os}\,\alpha_2 + \cos\alpha_1 \sin\alpha_2,$

$\cos\alpha_2 - \sin\alpha_1 \sin\alpha_2).$

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."

Addition



$x^2 + y^2$

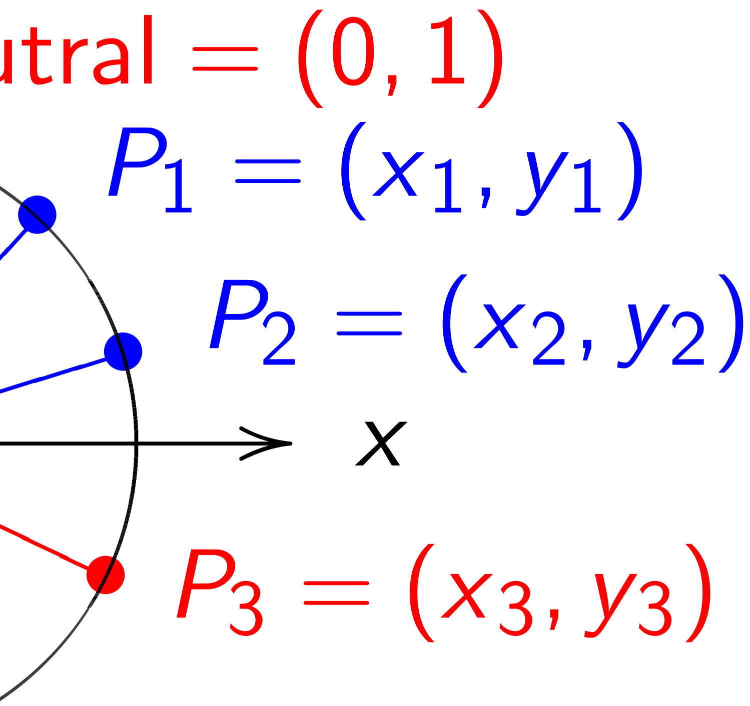
$x = \sin$

$(\sin(\alpha_1 +$

$(\sin\alpha_1\ \text{c}$

$\cos\alpha_1\ \text{c}$

ock:



ametrized by

os α. Recall

$s(\alpha_1 + \alpha_2)) =$

os $\alpha_1 \sin \alpha_2$,

n $\alpha_1 \sin \alpha_2)$.

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".

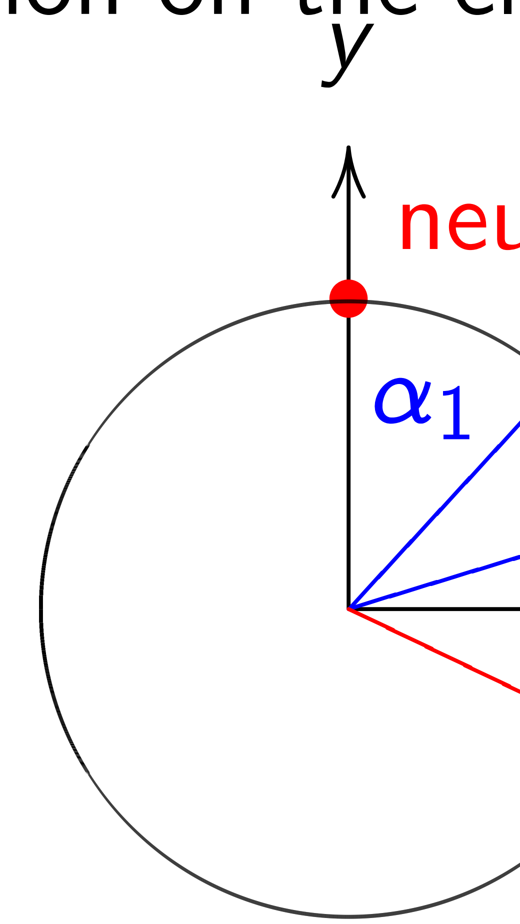
The point with $\alpha = 180^\circ$ has order 2 and equals 6:00.

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."

Addition on the cl



$x^2 + y^2 = 1$, par

$x = \sin \alpha, \quad y = $ co

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

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

$\cos \alpha_1 \cos \alpha_2 -$ si

$1)$

$x_1, y_1)$

(x_2, y_2)

(x_3, y_3)

by

ll

$)) =$

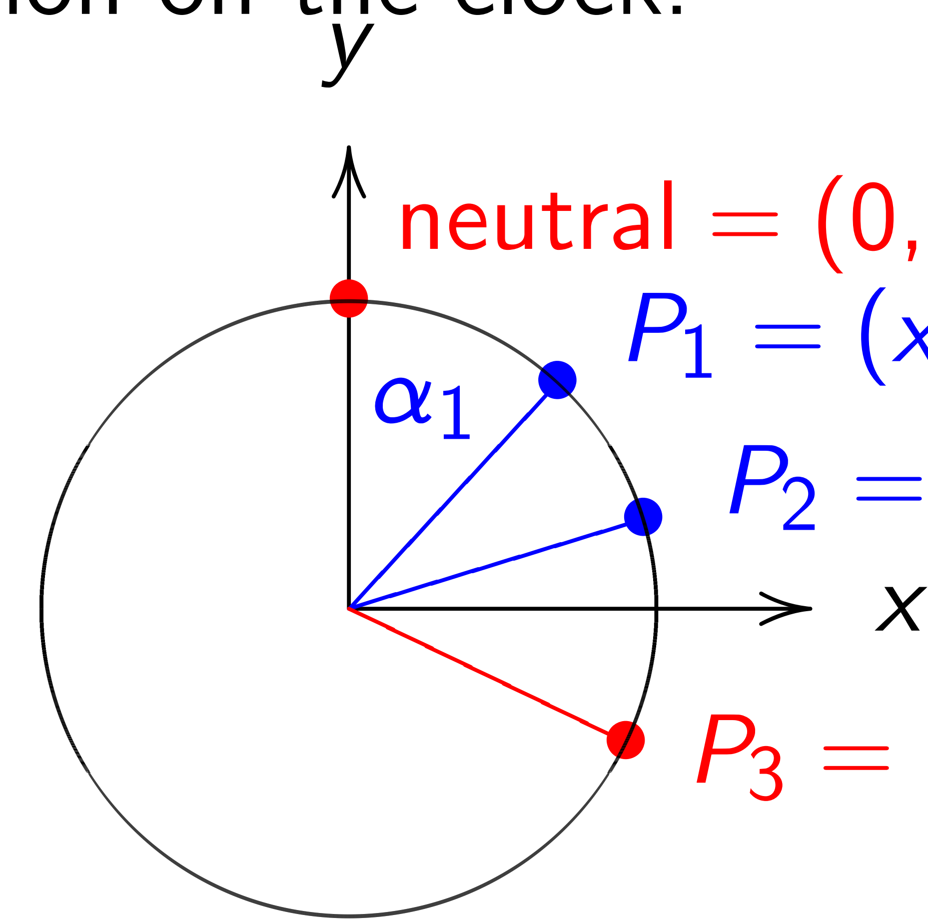
$_2,$

$_2).$

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 $\alpha = 180°$ 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."

Addition on the clock:



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

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

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

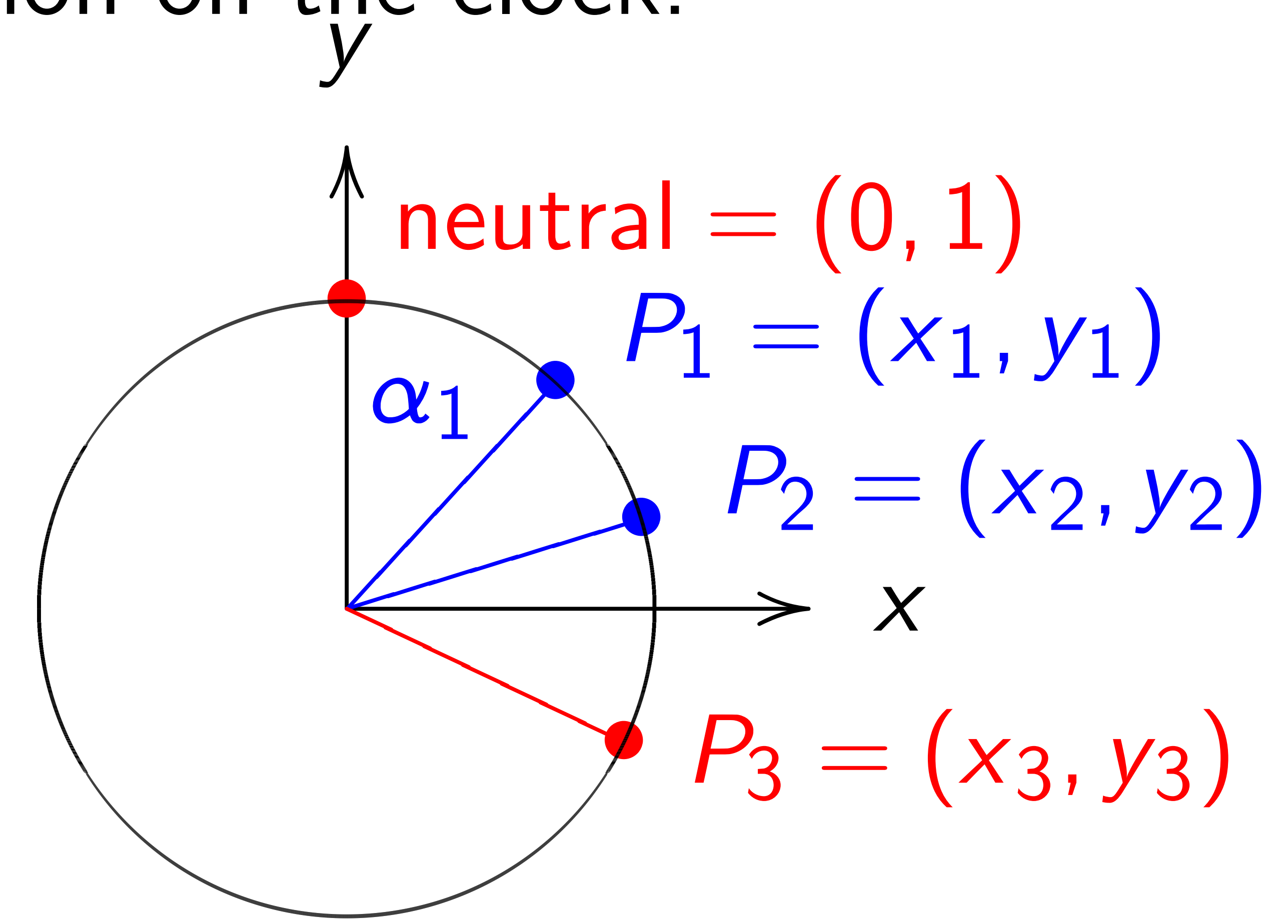
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2$

$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2$

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 $\alpha = 180°$ 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."

Addition on the clock:

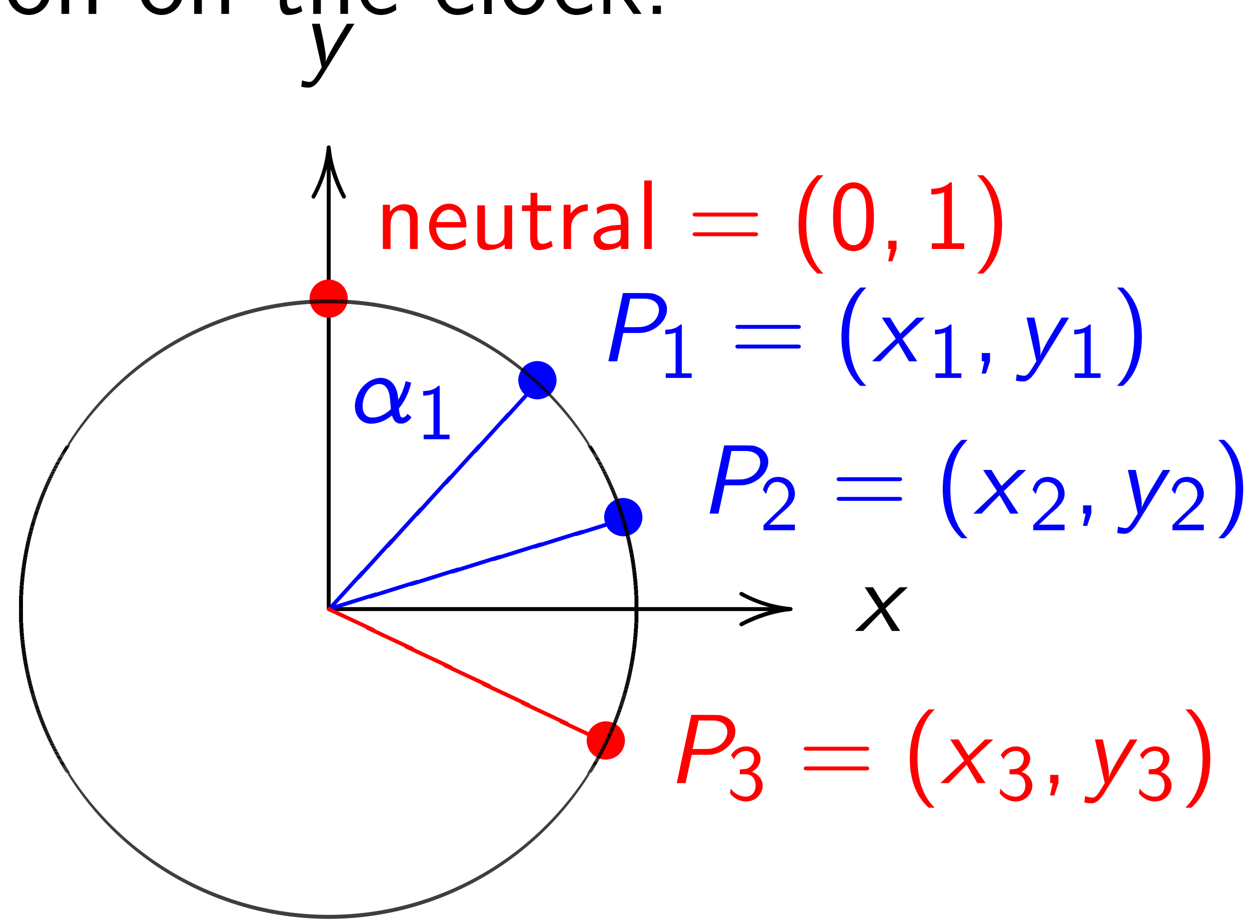


$x^2 + y^2 = 1$, parametrized by $x = \sin\alpha$, $y = \cos\alpha$. Recall
$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$
$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2)$.

two points corresponds
g the angles α_1 and α_2.
modulo 360° are a group,
s on clock are a group.

element: angle $\alpha = 0$;
, 1); "12:00".
nt with $\alpha = 180^\circ$
er 2 and equals 6:00.
d 9:00 have order 4.
of point with α
with $-\alpha$
$+ (-\alpha) = 0$.
re many more points
ngle α is not "nice."

Addition on the clock:



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

Clock ad

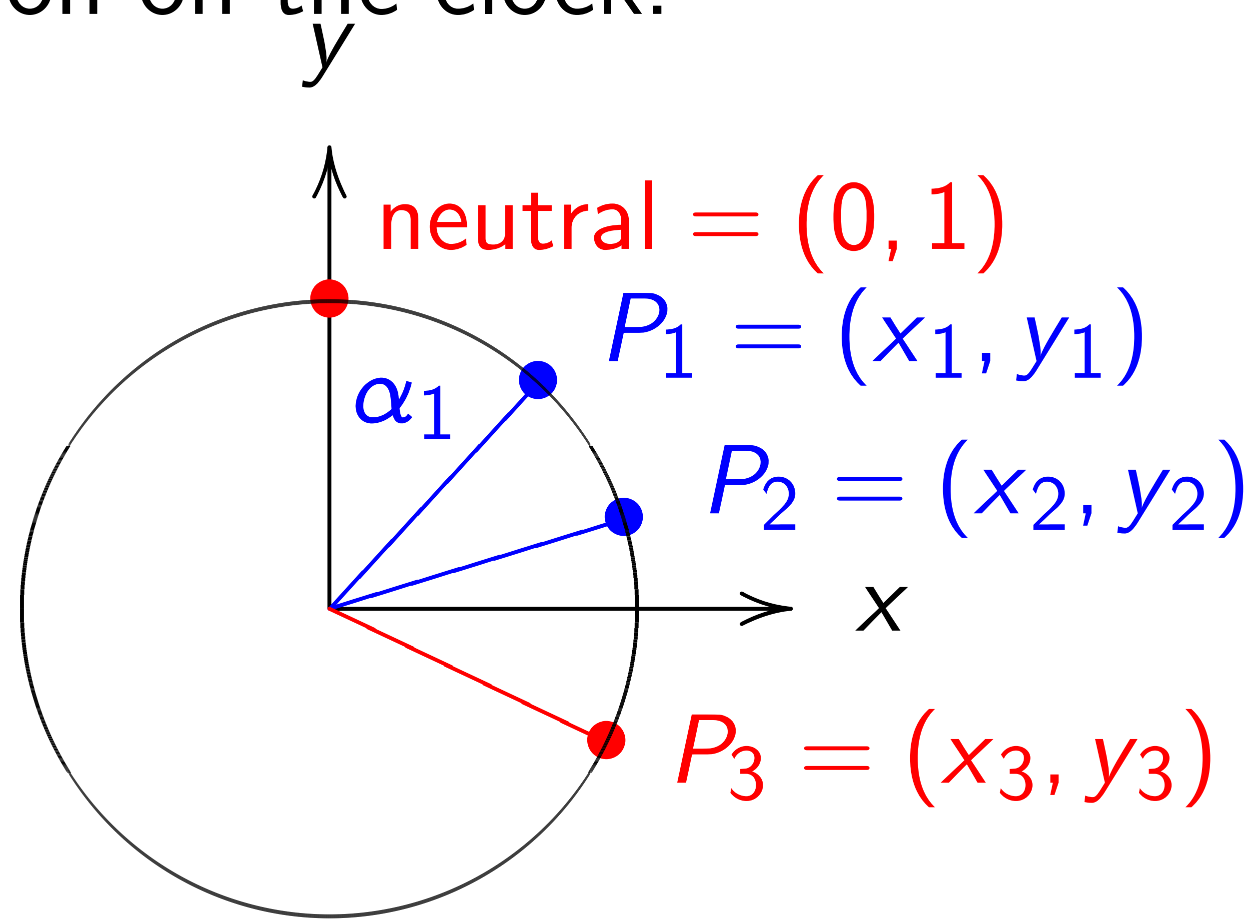
Use Cart
addition
for the c
sum $(x_1$

s corresponds
les α_1 and α_2.
0° are a group,
are a group.

angle $\alpha = 0$;
0".
$= 180°$
quals 6:00.
e order 4.
ith α

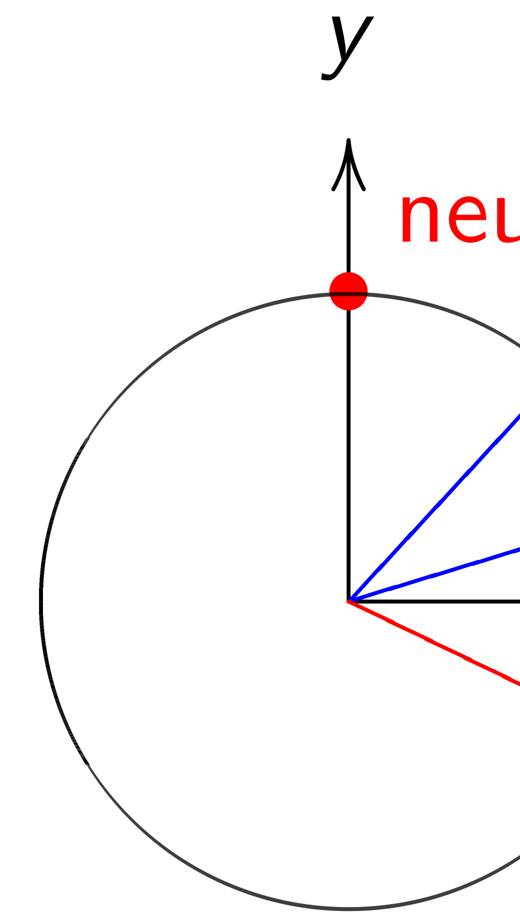
0.
nore points
not "nice."

Addition on the clock:



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

Clock addition wit



Use Cartesian coor
addition. Addition
for the clock $x^2 +$
sum $(x_1, y_1) + (x_2$

nds

α_2.

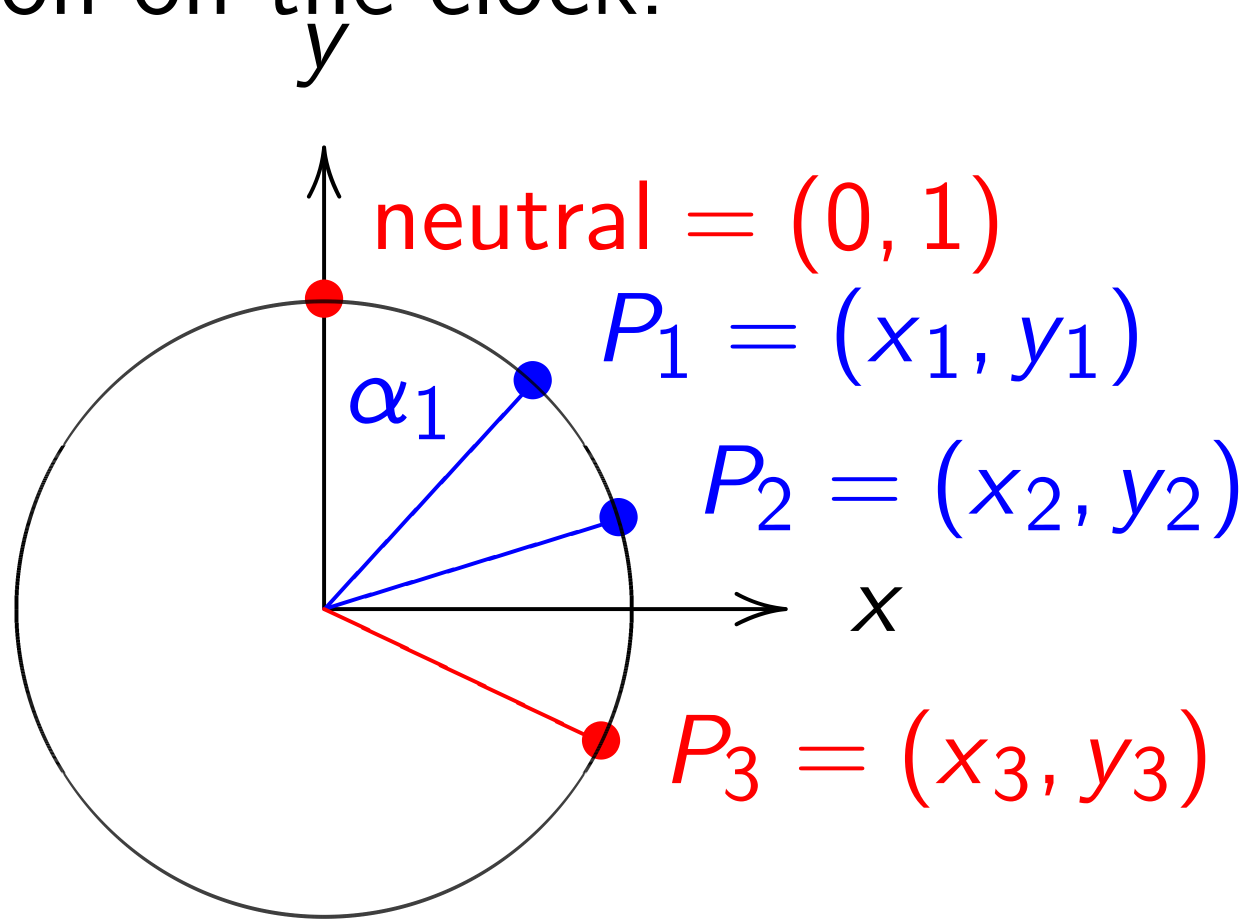
roup,

up.

$= 0$;

s

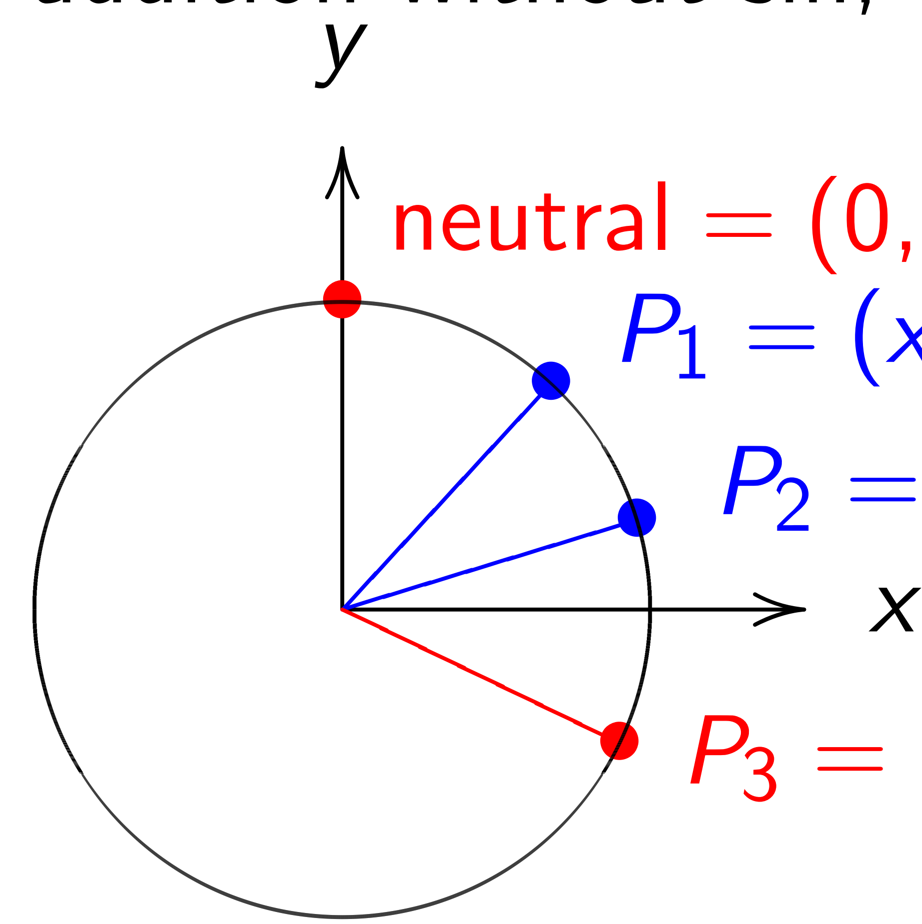
Addition on the clock:



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

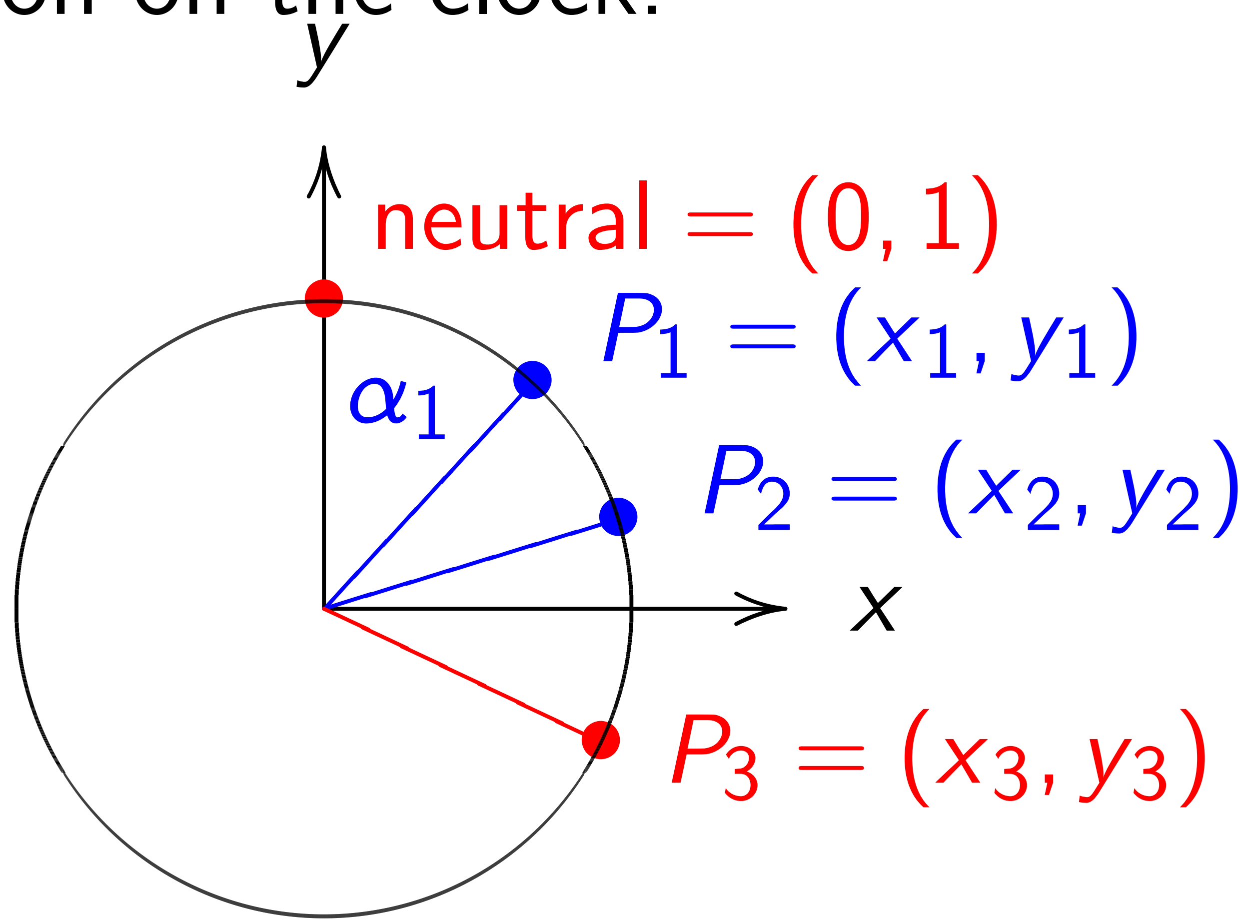
$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$
$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2)$.

Clock addition without sin, c



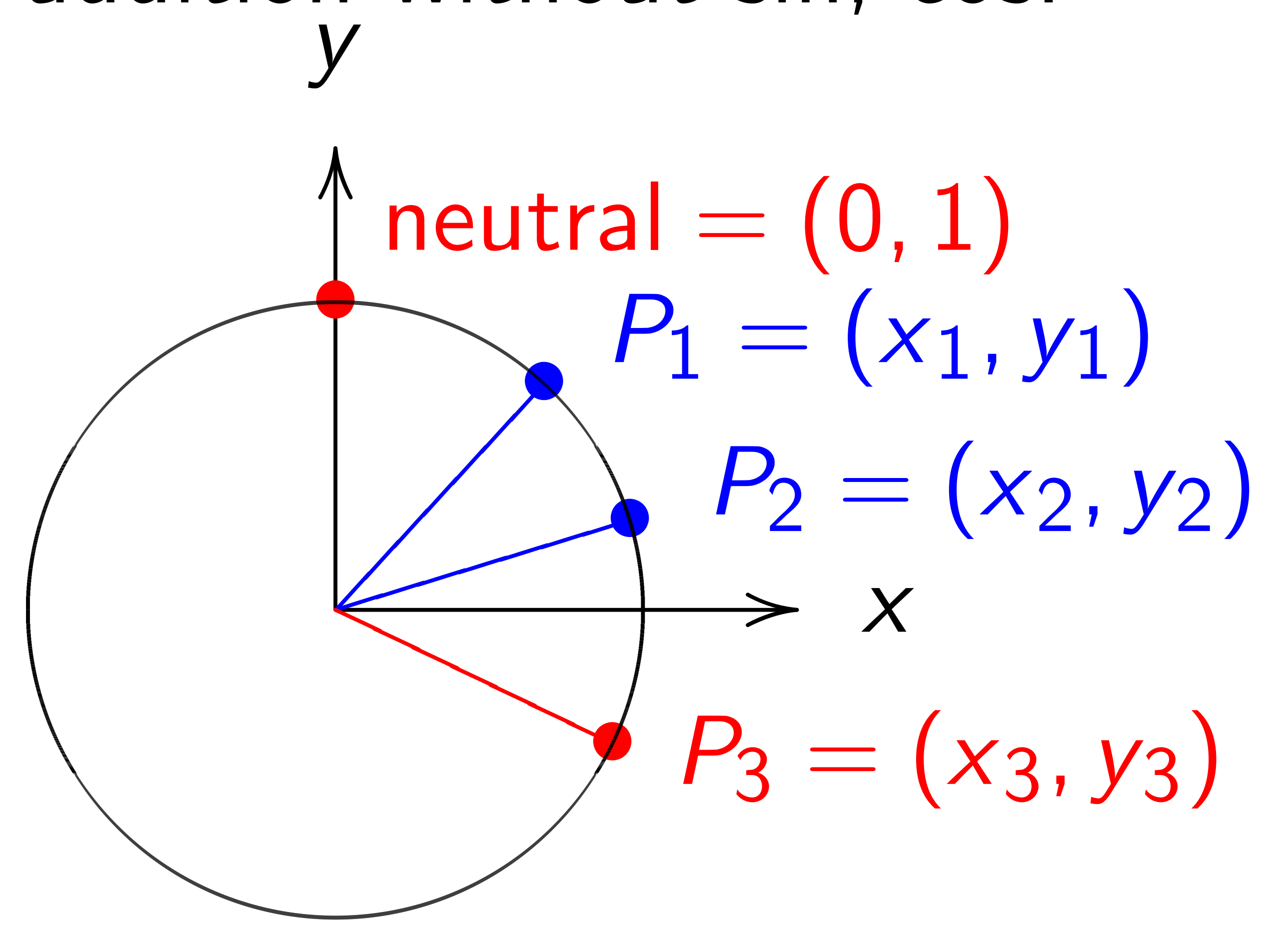
Use Cartesian coordinates fo addition. Addition formula for the clock $x^2 + y^2 = 1$: sum $(x_1, y_1) + (x_2, y_2) = (x$

Addition on the clock:



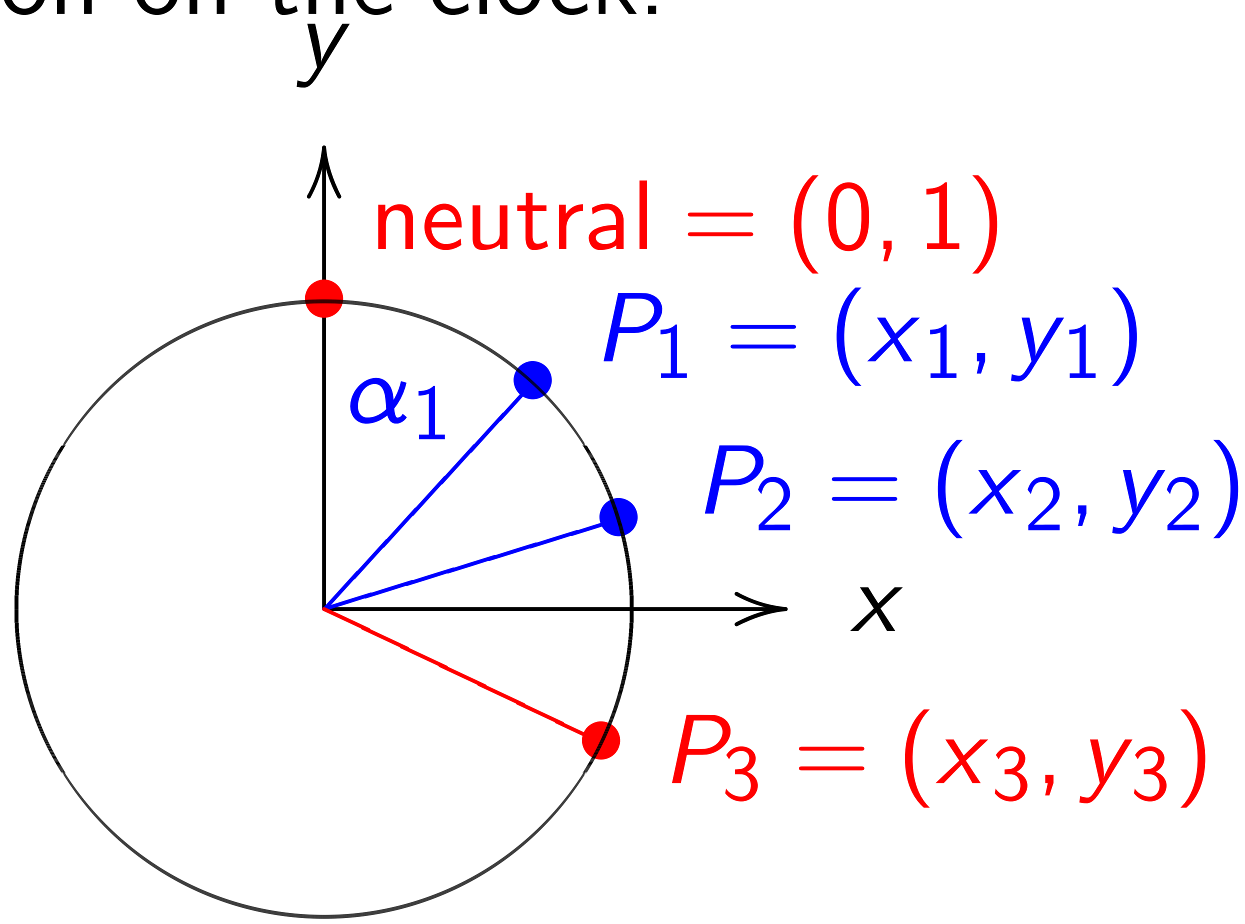
$x^2 + y^2 = 1$, parametrized by
$x = \sin\alpha, \quad y = \cos\alpha$. Recall
$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$
$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2)$.

Clock addition without sin, cos:



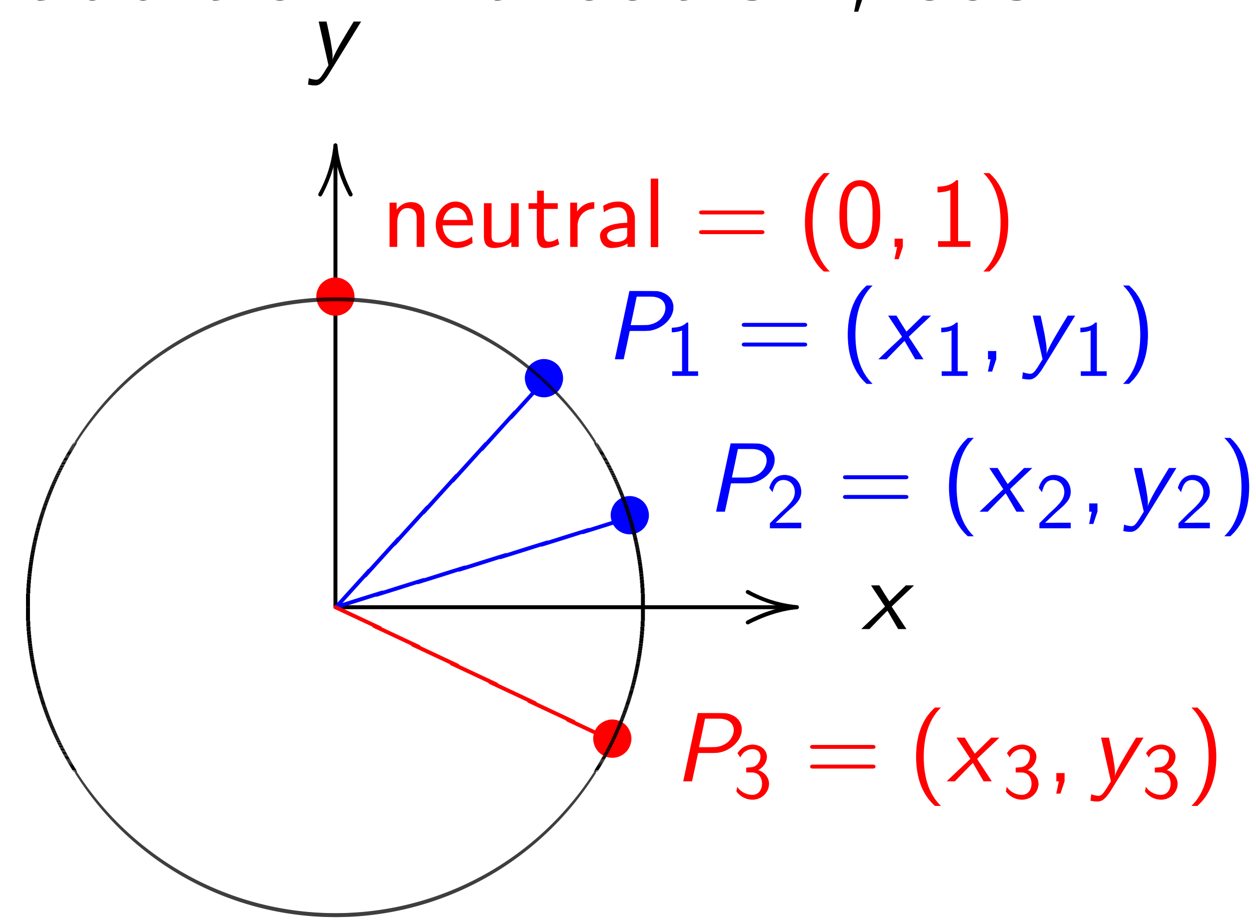
Use Cartesian coordinates for addition. Addition formula for the clock $x^2 + y^2 = 1$: sum $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$

Addition on the clock:



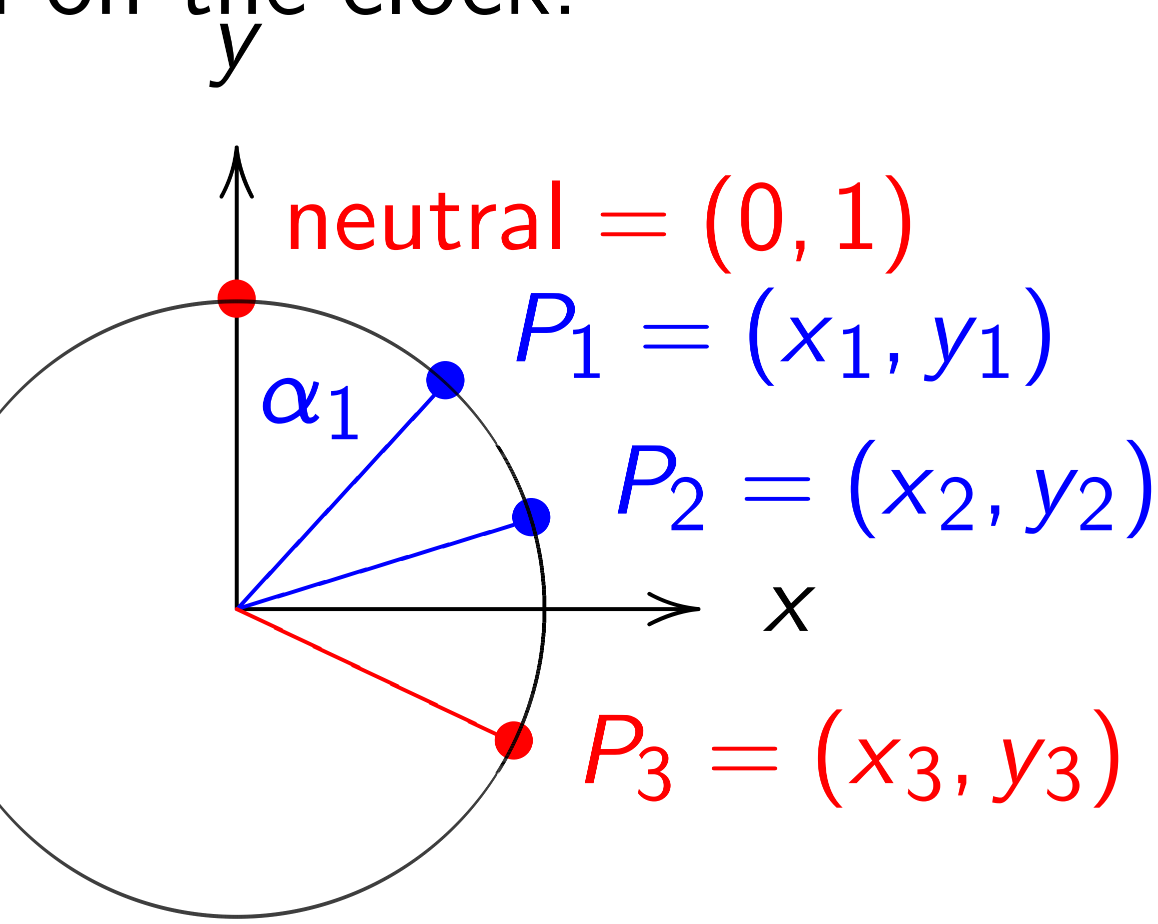
$x^2 + y^2 = 1$, parametrized by
$x = \sin\alpha, \quad y = \cos\alpha$. Recall
$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$
$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2)$.

Clock addition without sin, cos:



Use Cartesian coordinates for addition. Addition formula for the clock $x^2 + y^2 = 1$:
sum $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$
$= (x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.
Note $(x_1, y_1) + (-x_1, y_1) = (0, 1)$.
$kP = \underbrace{P + P + \cdots + P}_{k \text{ copies}}$ for $k \geq 0$.

n on the clock:



$= 1$, parametrized by

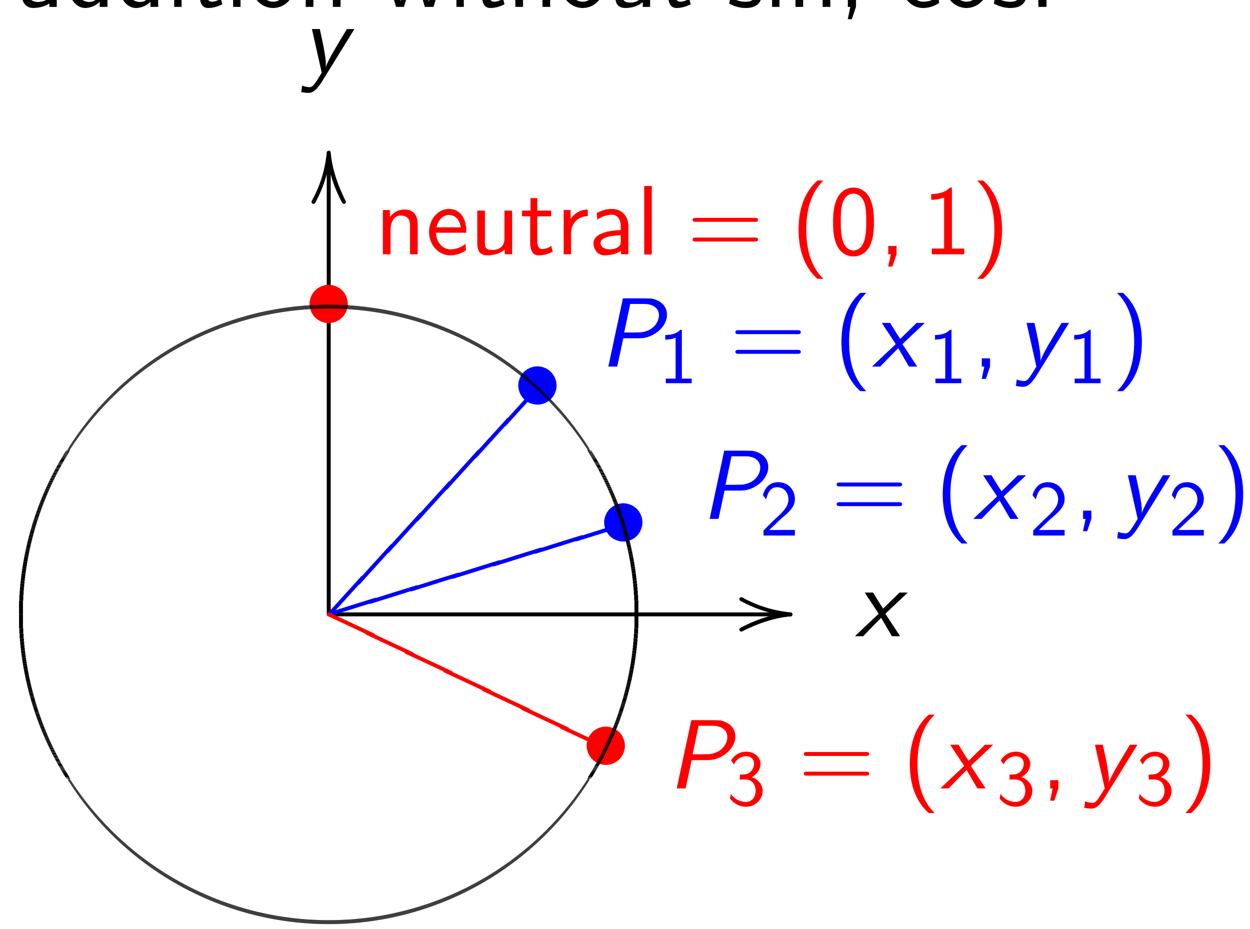
$\alpha, \quad y = \cos\alpha$. Recall

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

$\text{os}\,\alpha_2 + \cos\alpha_1 \sin\alpha_2,$

$\text{cos}\,\alpha_2 - \sin\alpha_1 \sin\alpha_2).$

Clock addition without sin, cos:



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

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

$= (x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

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

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

Example

"2:00" +

$= (\sqrt{3}/$

$= (-1/2$

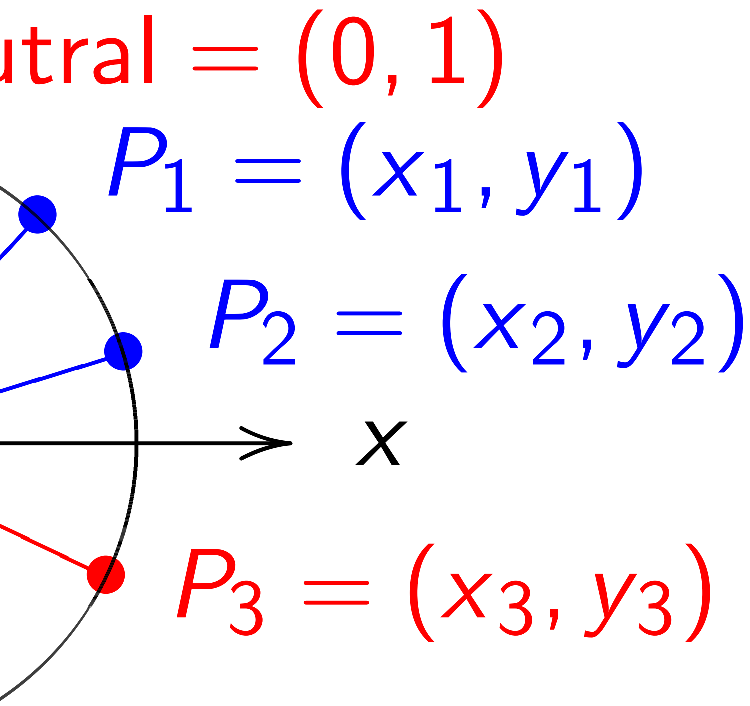
"5:00" +

$= (1/2,$

$= (\sqrt{3}/$

$2\left(\frac{3}{5}, \frac{4}{5}\right)$

ock:



ametrized by

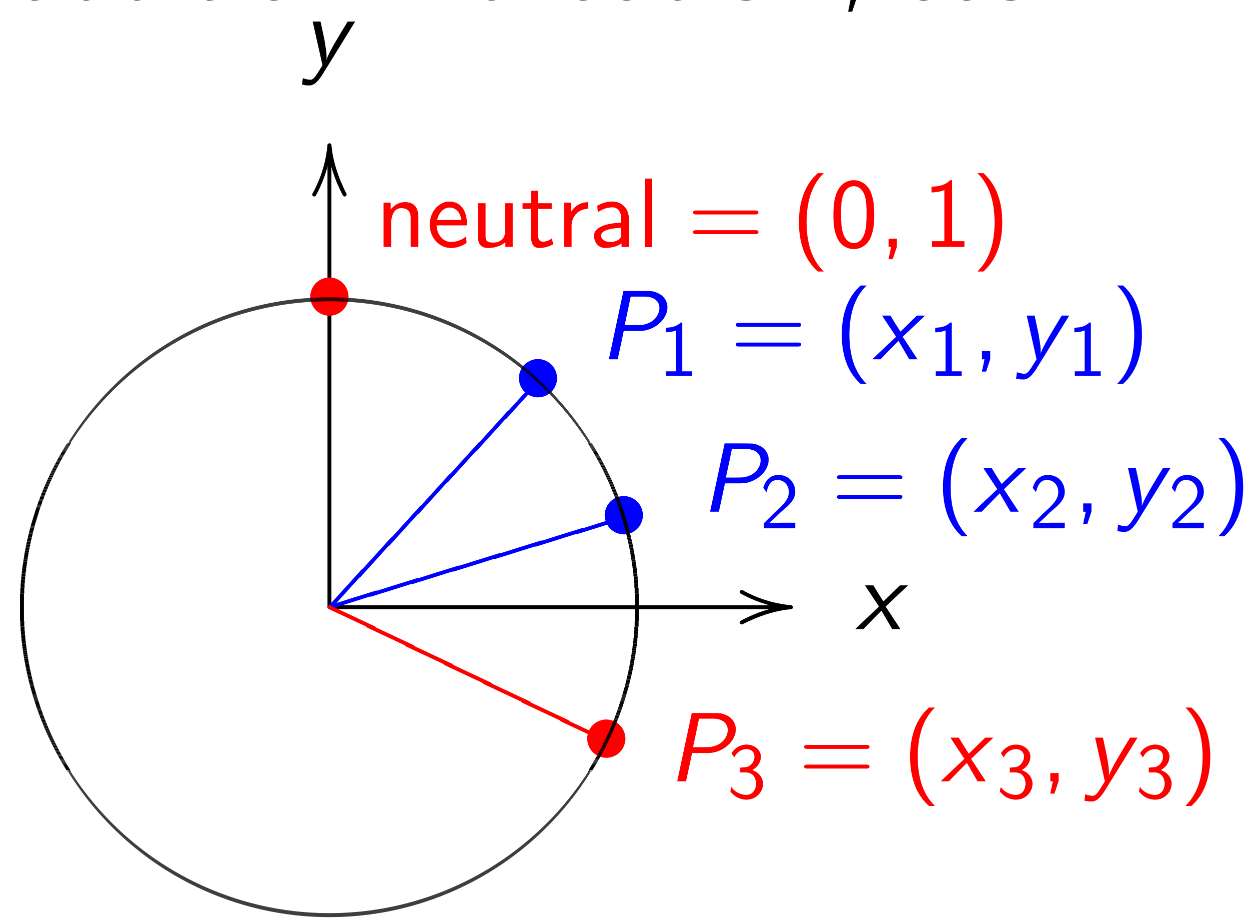
os α. Recall

$(\alpha_1 + \alpha_2)) =$

os $\alpha_1 \sin \alpha_2$,

n $\alpha_1 \sin \alpha_2)$.

Clock addition without sin, cos:



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

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

$= (x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

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

"2:00" + "5:00"

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

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

"5:00" + "9:00"

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

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

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

$1)$

$x_1, y_1)$

(x_2, y_2)

(x_3, y_3)

by

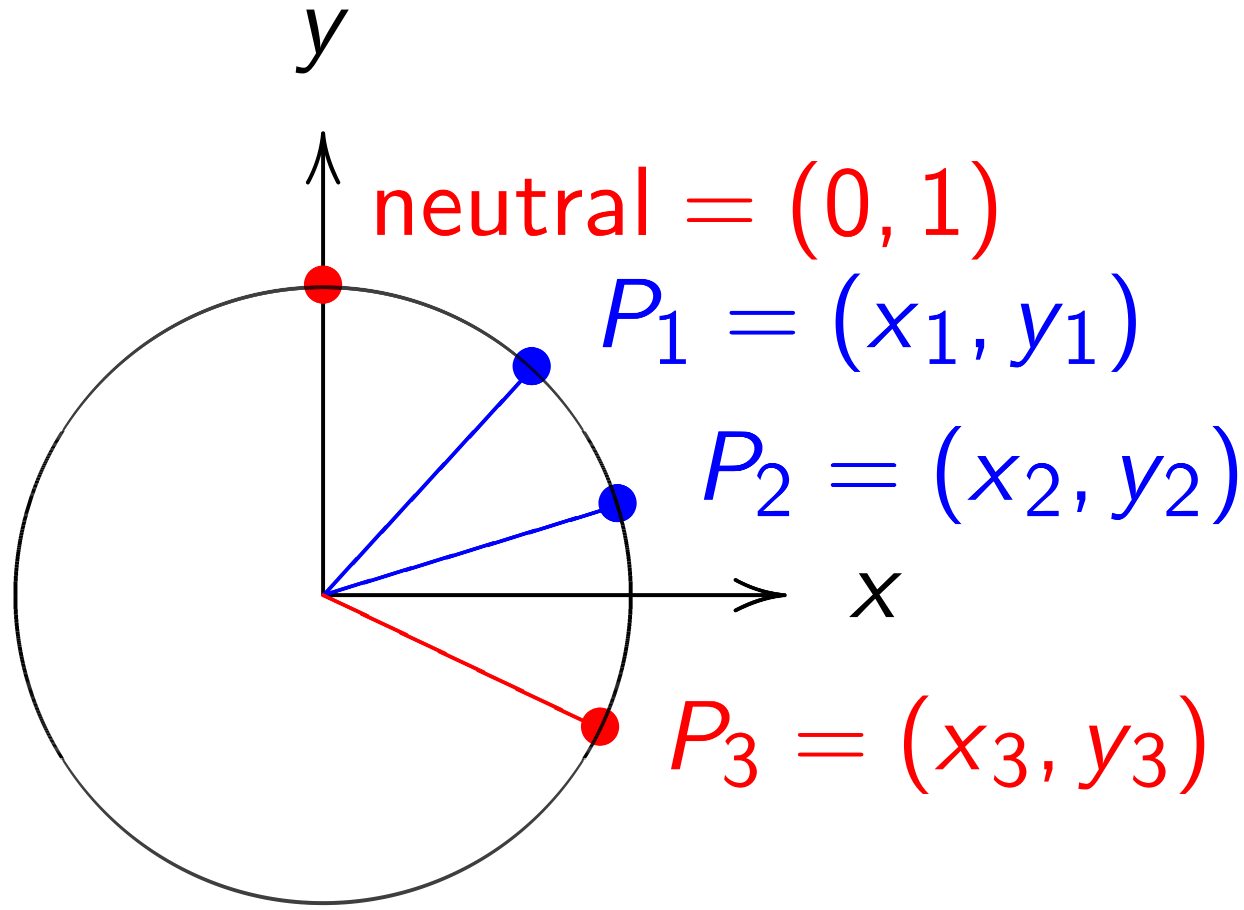
ll

$)) =$

$_2,$

$_2).$

Clock addition without sin, cos:



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

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

$= (x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

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"

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

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

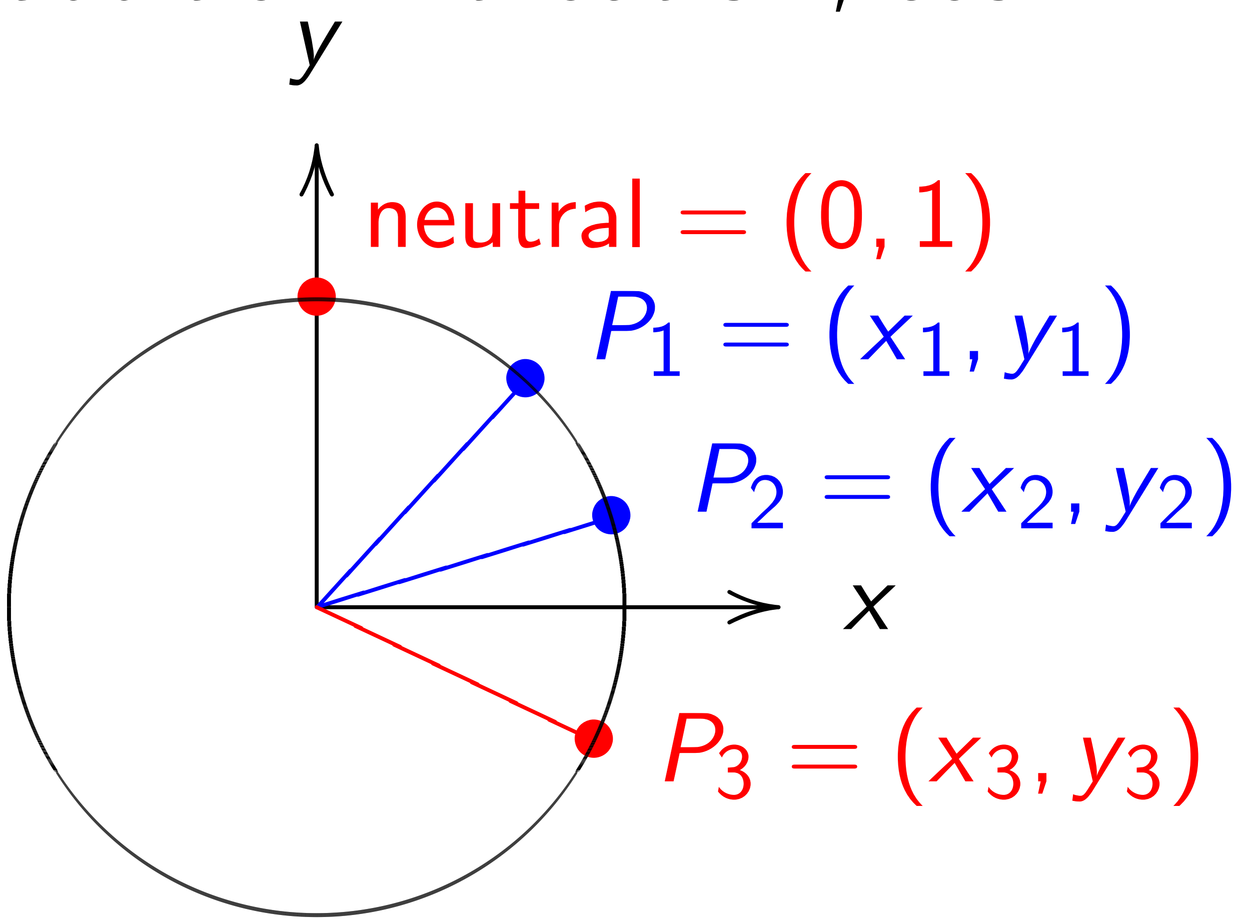
"5:00" + "9:00"

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

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

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

Clock addition without sin, cos:



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

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

$= (x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

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"

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

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

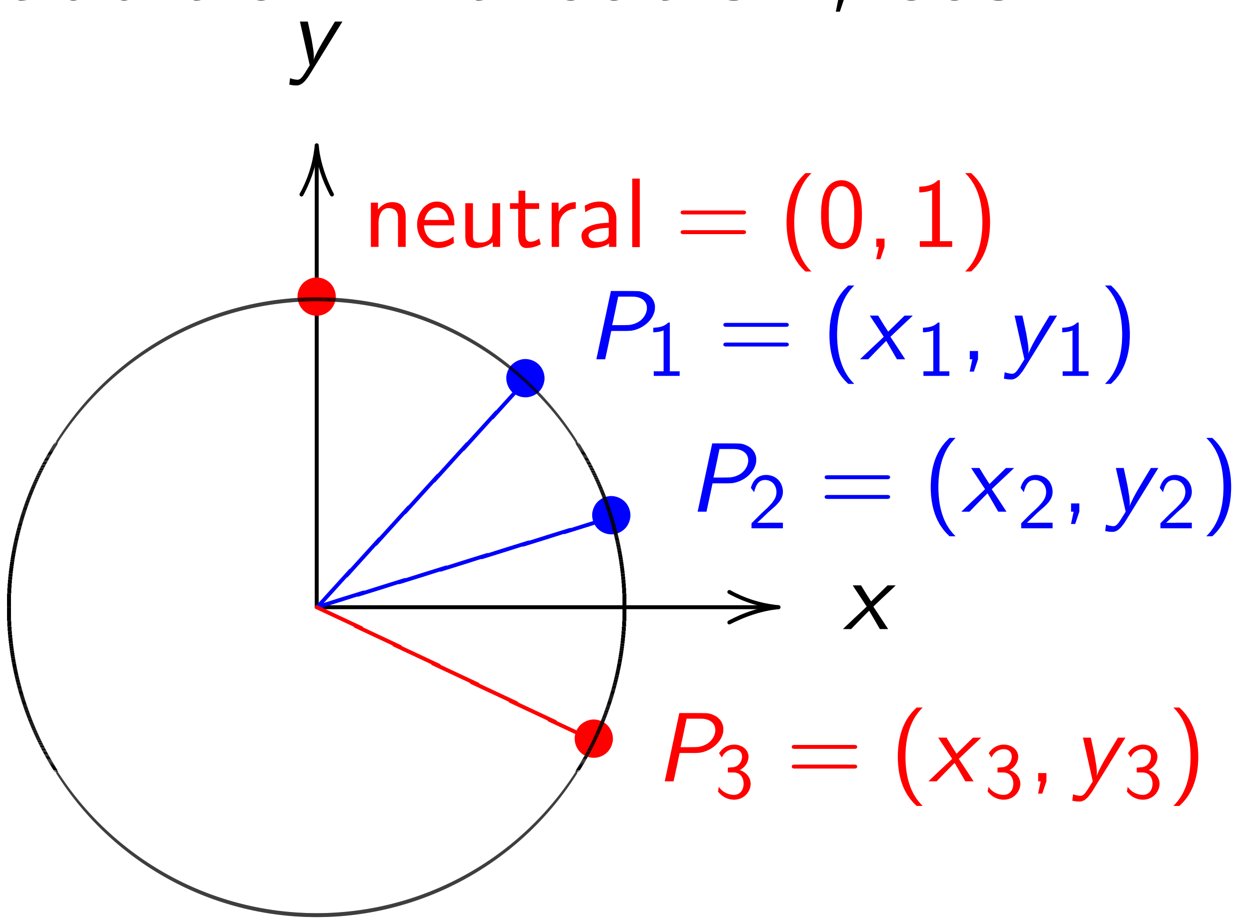
"5:00" + "9:00"

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

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

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

Clock addition without sin, cos:



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

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

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:

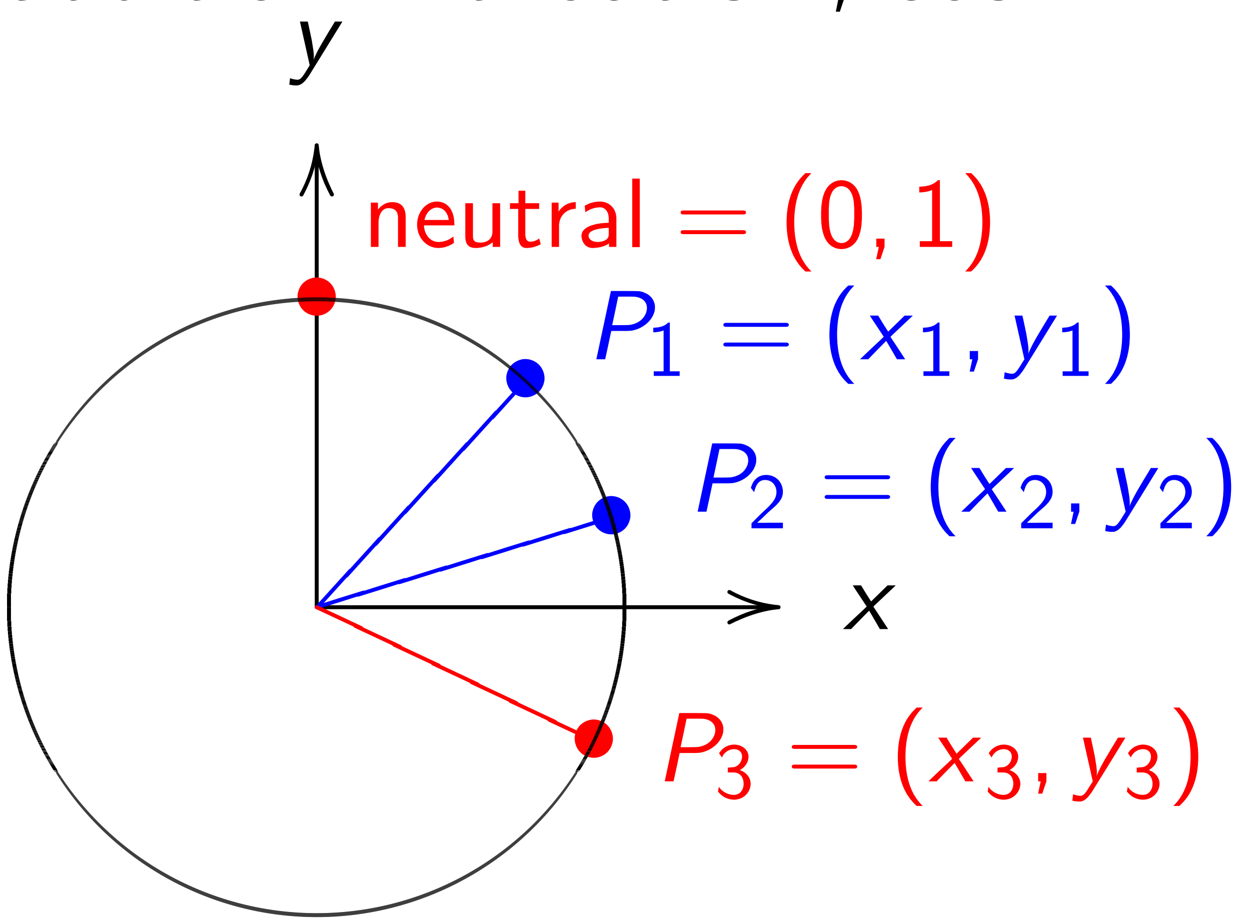
"2:00" + "5:00"
$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$
$= (-1/2, -\sqrt{3/4}) =$ "7:00".

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

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

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

Clock addition without sin, cos:



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

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

$= (x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

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

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

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"

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

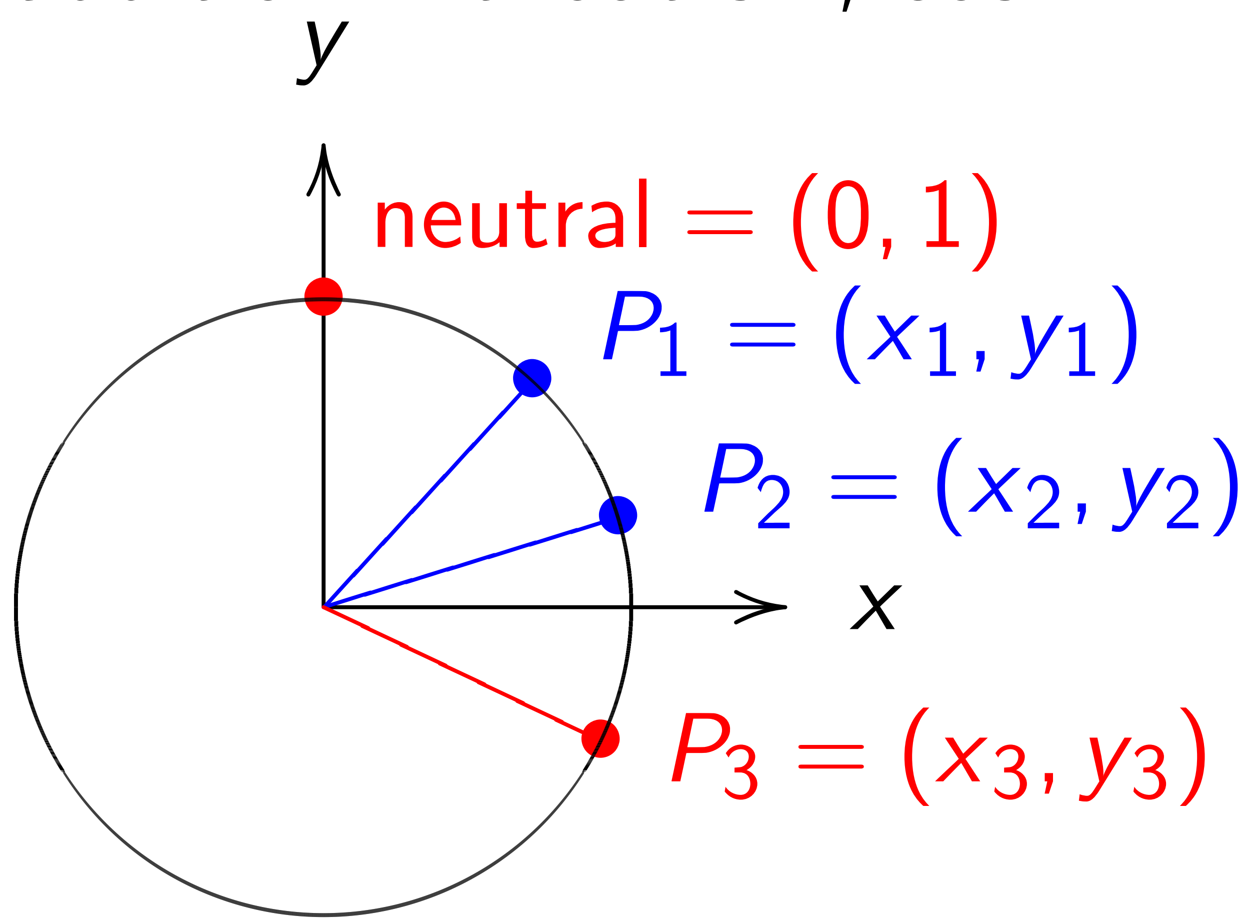
$= (\sqrt{3/4}, 1/2) =$ "2:00".

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

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

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

Clock addition without sin, cos:



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

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

$= (x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

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"

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

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

"5:00" + "9:00"

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

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

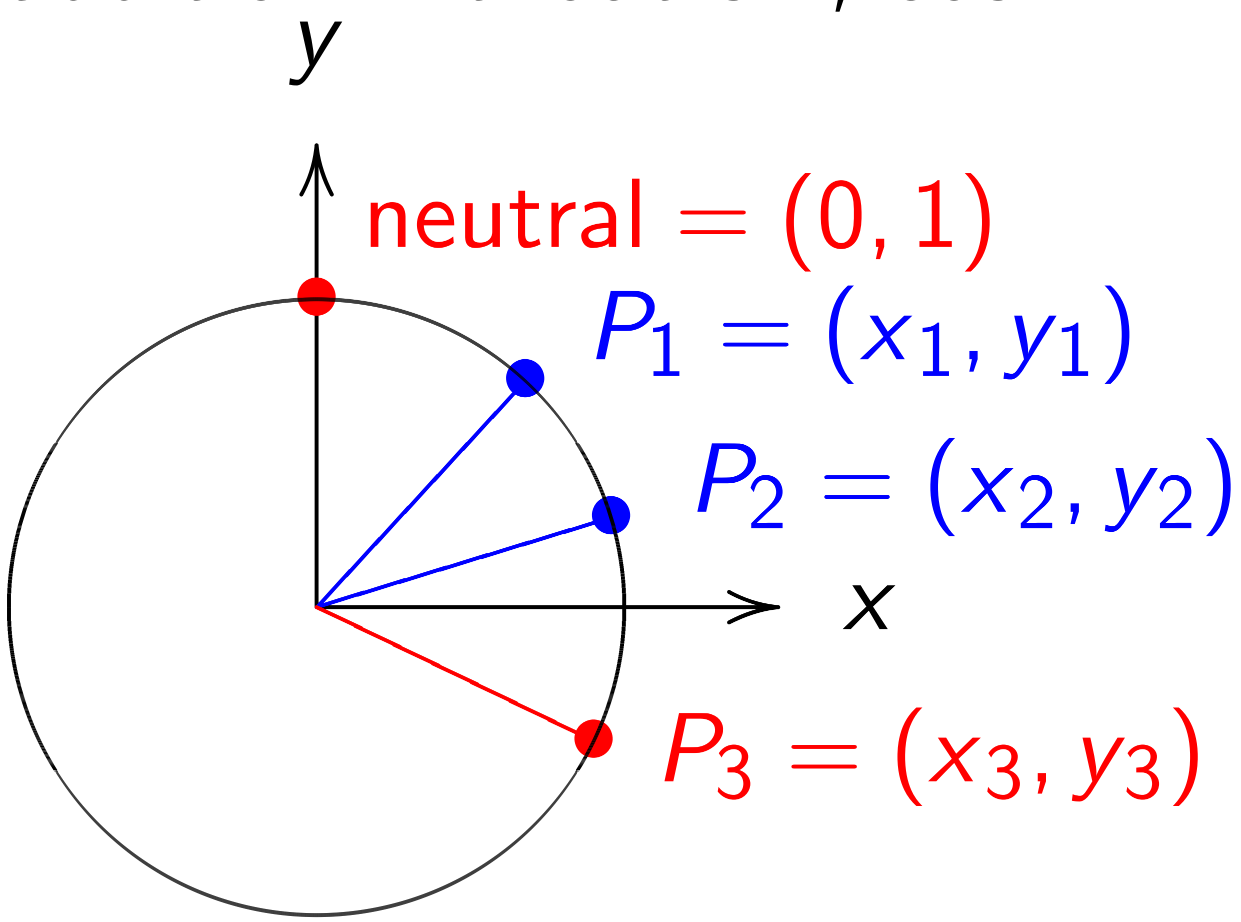
$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

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

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

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

Clock addition without sin, cos:



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

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

$= (x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

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"

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

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

"5:00" + "9:00"

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

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

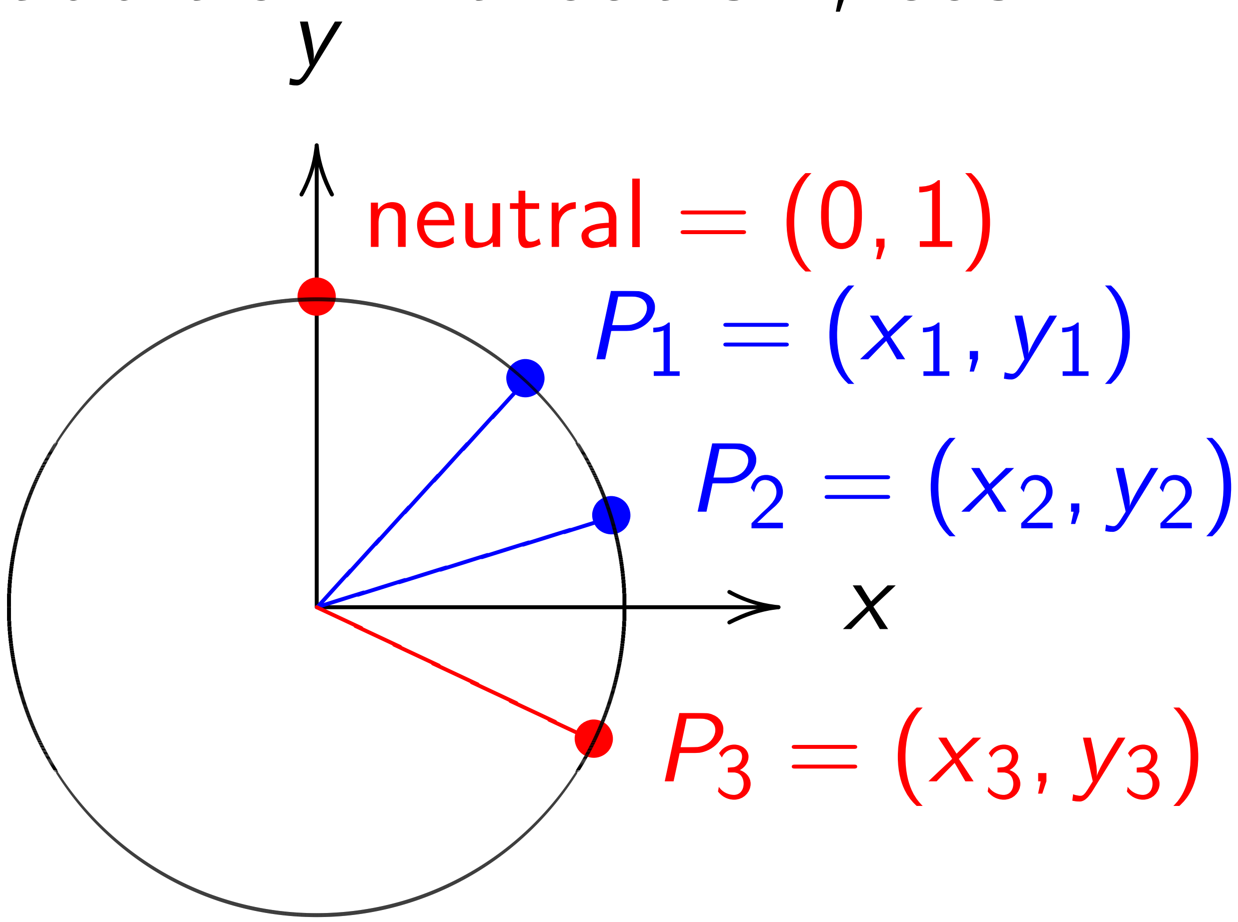
$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

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

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

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

Clock addition without sin, cos:



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

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

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

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

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

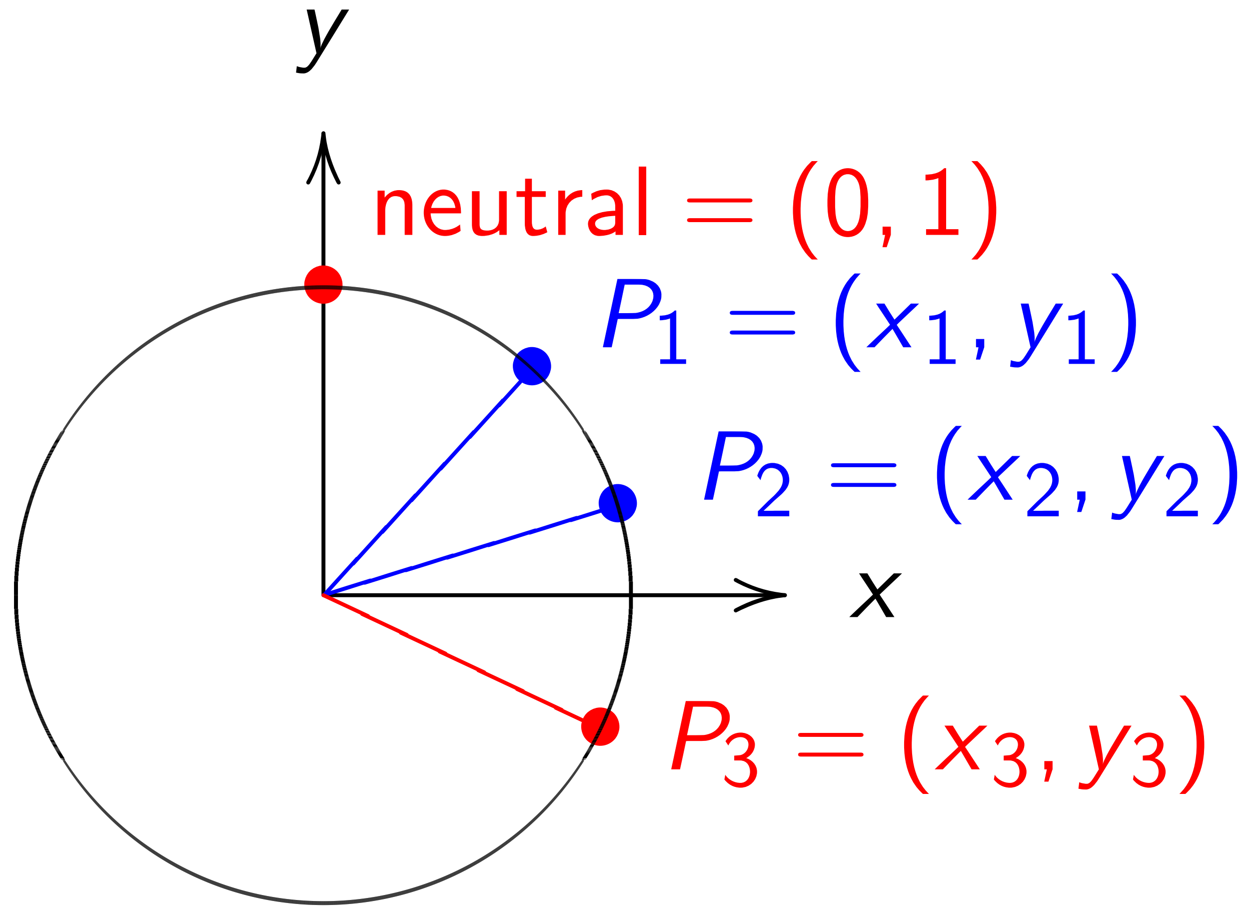
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

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

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

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

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"

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

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

"5:00" + "9:00"

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

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

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

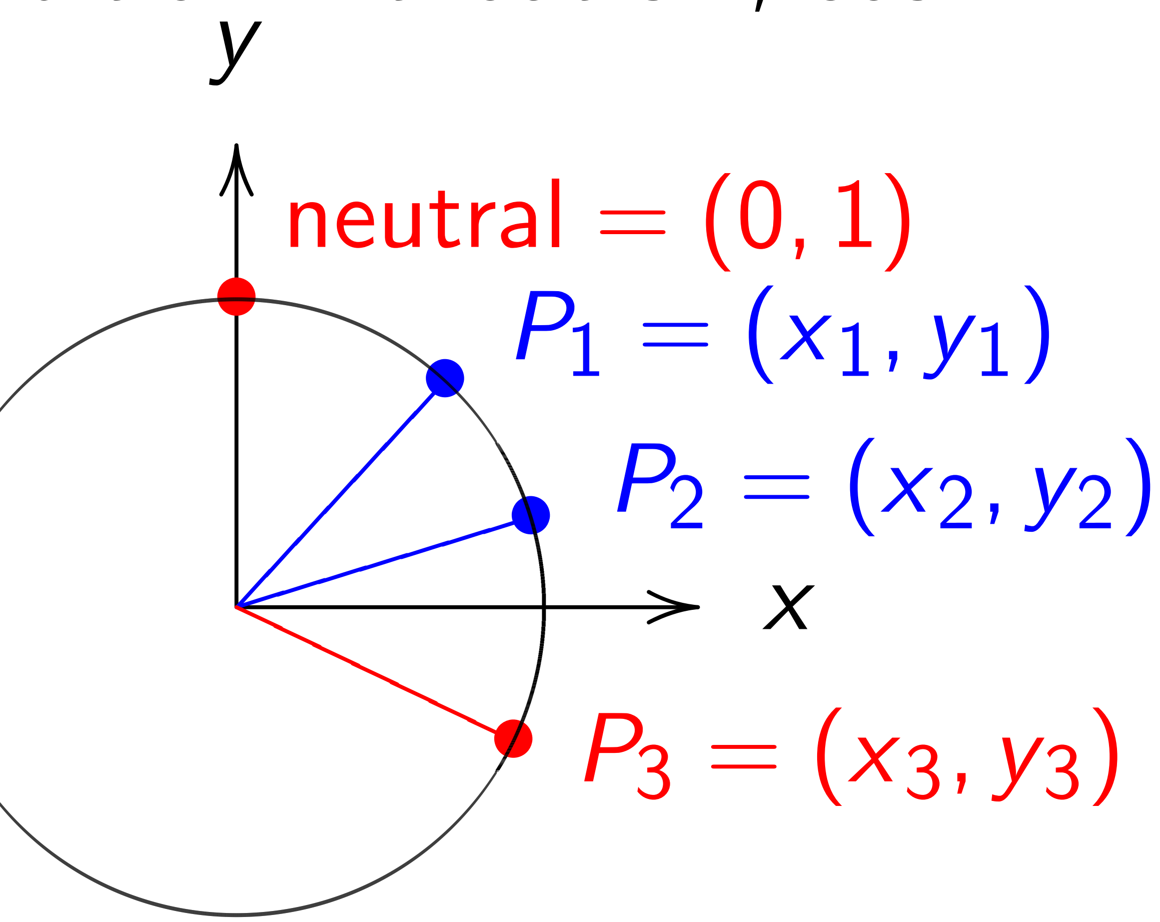
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

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

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

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

ddition without sin, cos:



tesian coordinates for

. Addition formula

clock $x^2 + y^2 = 1$:

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

$+ y_1 x_2, y_1 y_2 - x_1 x_2)$.

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

$+ P + \cdots + P$ for $k \ge 0$.

k copies

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 o

Clock($\mathbf{F}$

$\{(x, y) \in$

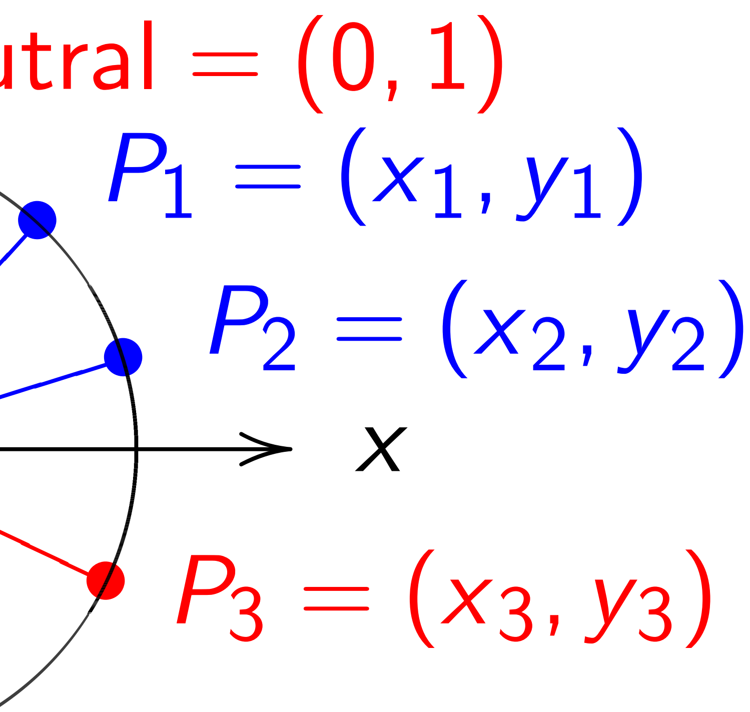
Here $\mathbf{F}_7$

$= \{0, 1,$

with $+,$

E.g. $2 \cdot$

thout sin, cos:



rdinates for

n formula

$y^2 = 1$:

$_2, y_2) = (x_3, y_3)$

$_1 y_2 - x_1 x_2)$.

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

$+ P$ for $k \geq 0$.

Examples of clock addition:

"2:00" + "5:00"

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

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

"5:00" + "9:00"

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

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

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

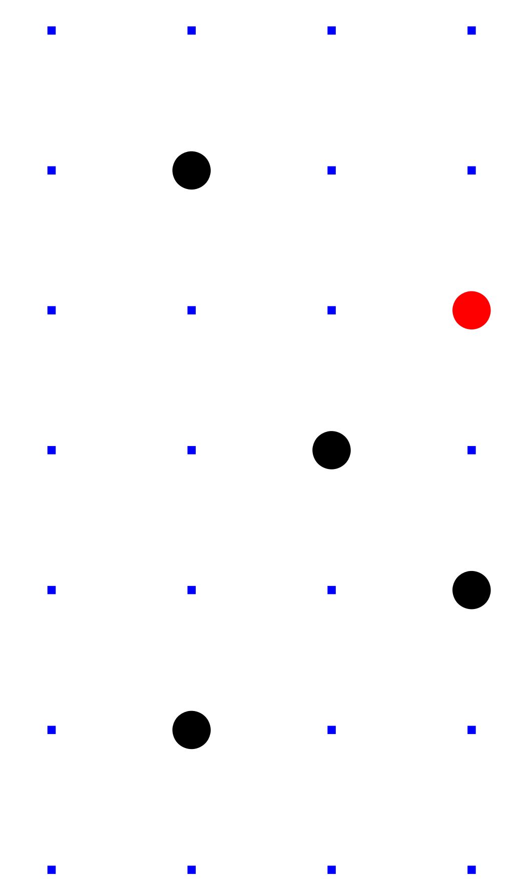
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

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

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

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

Clocks over finite



$\text{Clock}(\mathbf{F}_7) =$

$\{(x, y) \in \mathbf{F}_7 \times \mathbf{F}_7$

Here $\mathbf{F}_7 = \{0, 1, 2$

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

with $+, -, \times$ mod

E.g. $2 \cdot 5 = 3$ and

cos:

$1)$

$x_1, y_1)$

(x_2, y_2)

(x_3, y_3)

or

$x_3, y_3)$

$y_2)$.

$(0, 1)$.

$k \geq 0$.

Examples of clock addition:

"2:00" + "5:00"

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

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

"5:00" + "9:00"

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

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

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

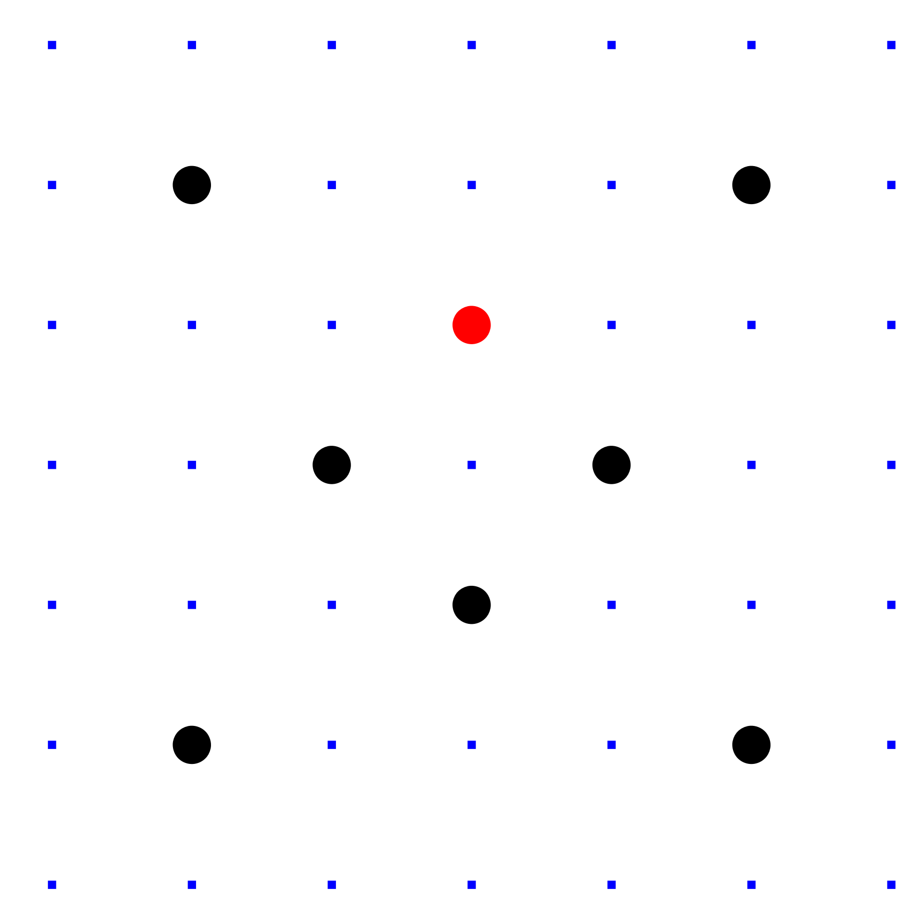
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

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

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

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

Clocks over finite fields



$\text{Clock}(\mathbf{F}_7) =$

$\{(x, y) \in \mathbf{F}_7 \times \mathbf{F}_7 : x^2 + y^2$

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

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

with $+, -, \times$ modulo 7.

E.g. $2 \cdot 5 = 3$ and $3/2 = 5$ i

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".

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

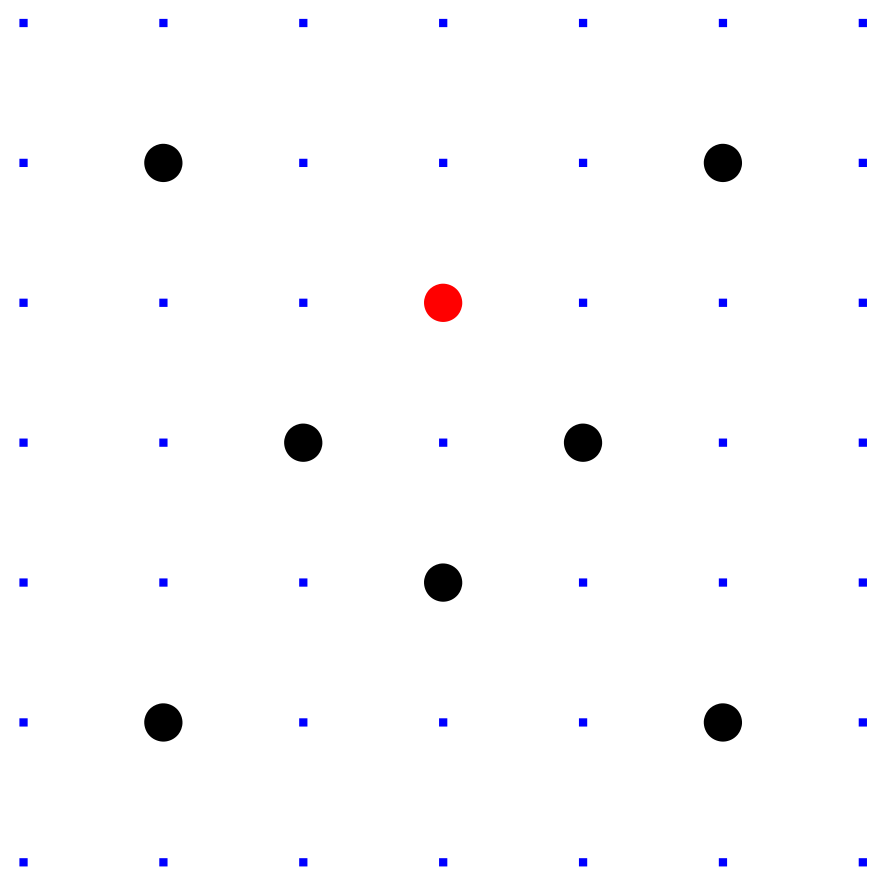
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

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

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

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

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.

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

es of clock addition:

– “5:00”

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

$2, -\sqrt{3/4}) =$ “7:00”.

– “9:00”

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

$\overline{4}, 1/2) =$ “2:00”.

$) = \left(\frac{24}{25}, \frac{7}{25}\right).$

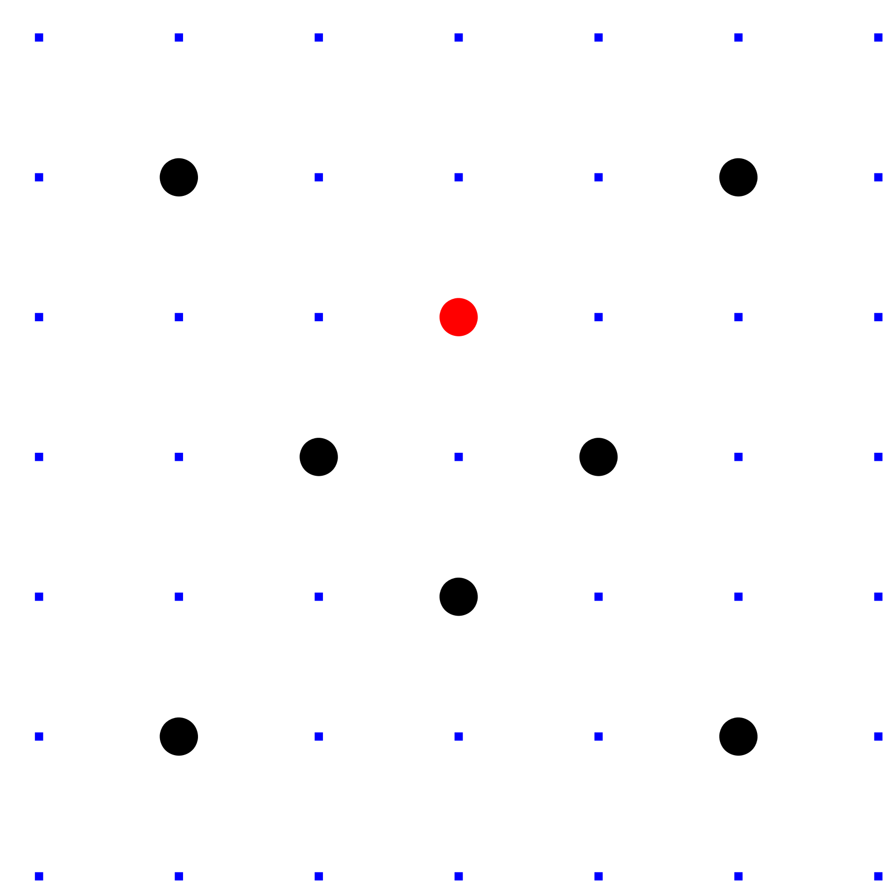
$) = \left(\frac{117}{125}, \frac{-44}{125}\right).$

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

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

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

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 $+, -, \times$ modulo 7.
E.g. $2 \cdot 5 = 3$ and $3/2 = 5$ in $\mathbf{F}_7$.

```
>>> for i
...   for
...
...
...
(0, 1)
(0, 6)
(1, 0)
(2, 2)
(2, 5)
(5, 2)
(5, 5)
(6, 0)
>>>
```

addition:

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

$) =$ “7:00”.

$+ (-1, 0)$

“2:00”.

$\frac{7}{25}\Big)$.

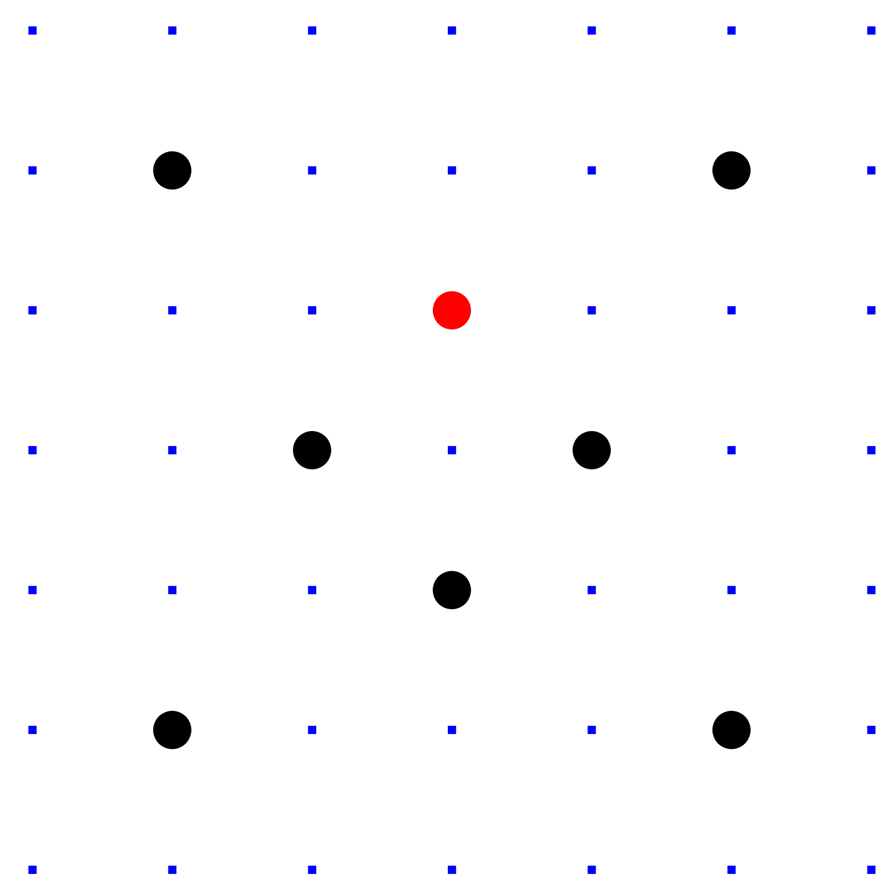
$, \frac{-44}{125}\Big)$.

$, \frac{-527}{625}\Big)$.

$= (x_1, y_1)$.

$_1) = (0, 1)$.

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

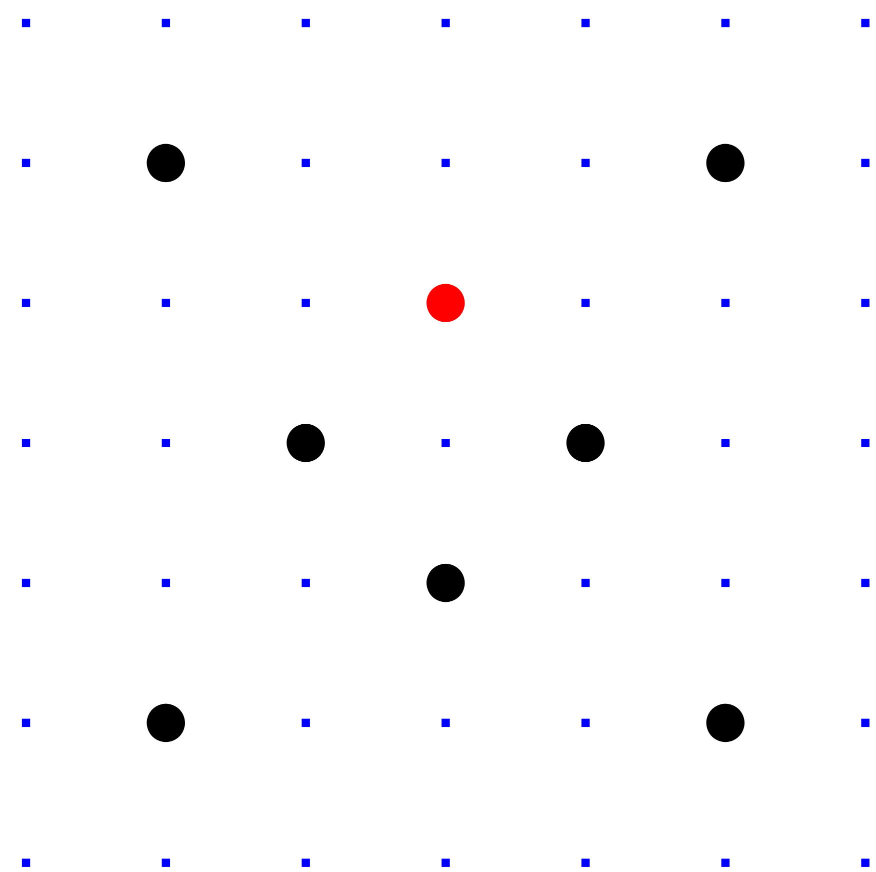
```
>>> for x in range
...   for y in ran
...     if (x*x+y*
...       print (x
...
(0, 1)
(0, 6)
(1, 0)
(2, 2)
(2, 5)
(5, 2)
(5, 5)
(6, 0)
>>>
```

3/4)

.

).

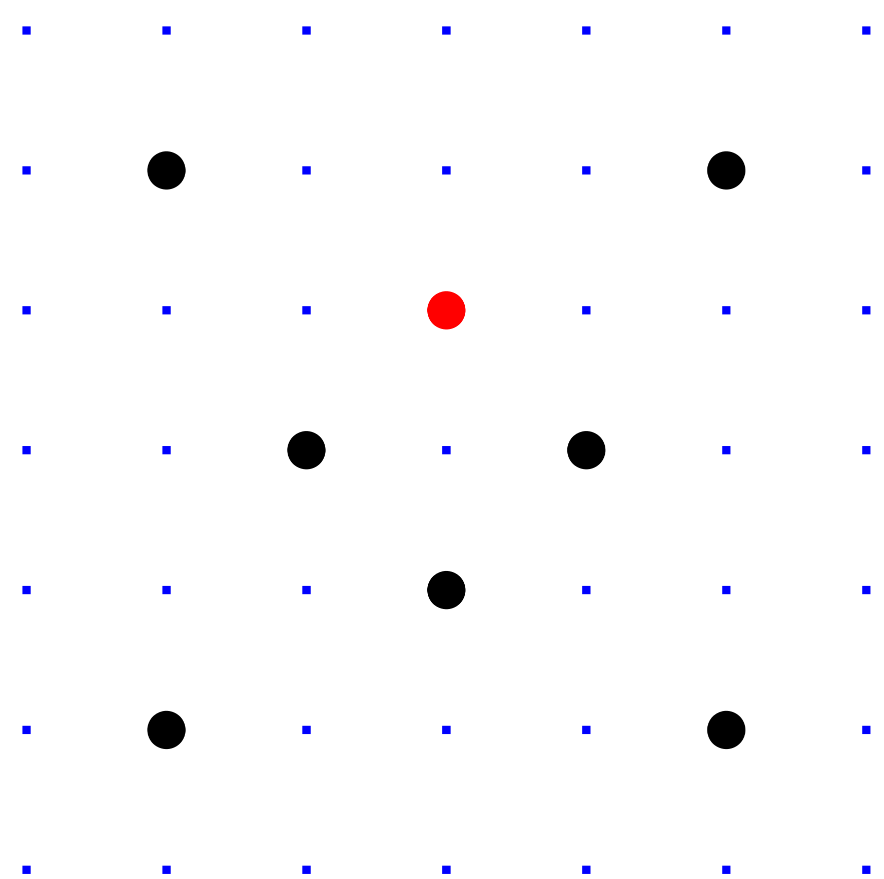
Clocks over finite fields



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

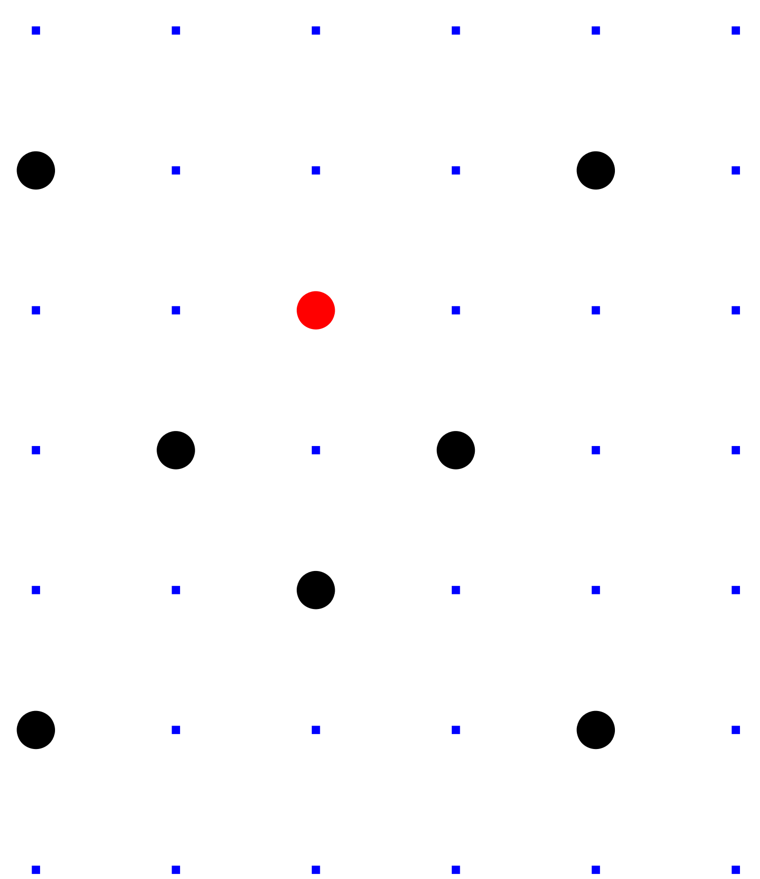
Clocks over finite fields



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

over finite fields



$_7) =$

$\in \mathbf{F}_7 \times \mathbf{F}_7 : x^2 + y^2 = 1\}.$

$= \{0, 1, 2, 3, 4, 5, 6\}$

$2, 3, -3, -2, -1\}$

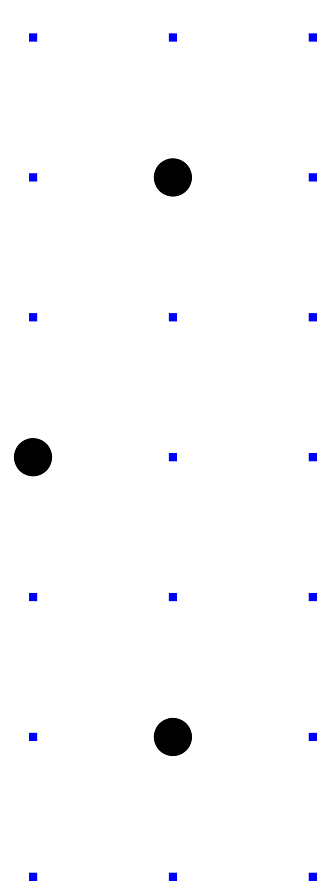
$-, \times$ modulo 7.

$5 = 3$ 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)
>>>
```

```
>>> clas
...   de
...     
...   de
...     
...   __
...
>>> prin
2
>>> prin
6
>>> prin
0
>>> prin
3
```

fields



$: x^2 + y^2 = 1\}.$

$, 3, 4, 5, 6\}$

$2, -1\}$

ulo 7.

$3/2 = 5$ 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.int =
...   def __str__(
...     return str
...   __repr__ = _
...
>>> print F7(2)
2
>>> print F7(6)
6
>>> print F7(7)
0
>>> print F7(10)
3
```

$= 1\}$.

n $\mathbf{F}_7$.

```
>>> 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
```

```
>>> 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
```

```
x in range(7):
r y in range(7):
if (x*x+y*y) % 7 == 1:
  print (x,y)
```

```
>>> 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._
...   a..
>>>
>>> prin
True
>>> prin
True
>>> prin
True
>>> prin
False
>>> prin
False
>>> prin
False
```

```
(7):
ge(7):
y) % 7 == 1:
,y)
```

```
>>> 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__ = la
...     a.int == b.i
>>>
>>> print F7(7) ==
True
>>> print F7(10) =
True
>>> print F7(-3) =
True
>>> print F7(0) ==
False
>>> print F7(0) ==
False
>>> print F7(0) ==
False
```

1:

```
>>> 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
```

```
>>> 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
```

```
s F7:
f __init__(self,x):
self.int = x % 7
f __str__(self):
return str(self.int)
repr__ = __str__

t F7(2)

t F7(6)

t F7(7)

t F7(10)
```

```
>>> 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._
...   F7
>>> F7._
...   F7
>>> F7._
...   F7
>>>
>>> prin
0
>>> prin
4
>>> prin
3
>>>
```

```
(self,x):
 x % 7
self):
(self.int)
_str__
```

```
>>> 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__ = l
...   F7(a.int + b
>>> F7.__sub__ = l
...   F7(a.int - b
>>> F7.__mul__ = l
...   F7(a.int * b
>>>
>>> print F7(2) +
0
>>> print F7(2) -
4
>>> print F7(2) *
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
>>>
```

```
>>> 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
>>>
```

```
_eq__ = lambda a,b: \
int == b.int

t F7(7) == F7(0)

t F7(10) == F7(3)

t F7(-3) == F7(4)

t F7(0) == F7(1)

t F7(0) == F7(2)

t F7(0) == F7(3)
```

```
>>> 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 e

```
p = 1000
class Fp
  ...

def cloc
  x1,y1 =
  x2,y2 =
  x3 = x
  y3 = y
  return
```

```
mbda a,b: \
nt

 F7(0)

= F7(3)

= F7(4)

 F7(1)

 F7(2)

 F7(3)
```

```
>>> 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: C

```
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
```

\

```
>>> 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($\mathbf{F}_{100}$

```
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
```

```
>>> 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
```

```
_add__ = lambda a,b: \
(a.int + b.int)
_sub__ = lambda a,b: \
(a.int - b.int)
_mul__ = lambda a,b: \
(a.int * b.int)

t F7(2) + F7(5)

t F7(2) - F7(5)

t F7(2) * F7(5)
```

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 =
>>> P2 =
>>> prin
(4000, 7
>>> P3 =
>>> prin
(15000,
>>> P4 =
>>> P5 =
>>> P6 =
>>> prin
(780000,
>>> prin
(780000,
>>>
```

```
ambda a,b: \
.int)
ambda a,b: \
.int)
ambda a,b: \
.int)

F7(5)

F7(5)

F7(5)
```

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

\

\

\

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)
>>>
```

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)
>>>
```

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

```
003
:

kadd(P1,P2):
= P1
= P2
1*y2+y1*x2
1*y2-x1*x2
 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
...   if
...   re
...   if
...   Q
...   Q
...   if
...   re
...
>>> n =
>>> scal
(947472,
>>>
```

Can you

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

):

```
>>> 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
...   if n == 0: \
...   return (Fp(0
...   if n == 1: r
...   Q = scalarmu
...   Q = clockadd
...   if n % 2: Q
...   return Q
...
>>> n = oursixdigi
>>> scalarmult(n,P
(947472, 736284)
>>>
```

Can you figure out

0003).

```
>>> 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
...   return Q
...
>>> n = oursixdigitsecret
>>> scalarmult(n,P)
(947472, 736284)
>>>
```

Can you figure out our secre

```
>>> 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?

```
(Fp(1000),Fp(2))
 clockadd(P,P)
t P2
)
 clockadd(P2,P)
t P3
26)
 clockadd(P3,P)
 clockadd(P4,P)
 clockadd(P5,P)
t P6
 1351)
t clockadd(P3,P3)
 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 cr

The "Cl
protocol

Standard
base po

Alice ch
compute

Bob cho
compute

Alice co
Bob com
They us
to encry

```
Fp(2))
P,P)



P2,P)



P3,P)
P4,P)
P5,P)



(P3,P3)
```

```
>>> 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 cryptograph

The "Clock Diffie-
protocol":

Standardize large
base point (x, y)

Alice chooses big s
computes her publ

Bob chooses big se
computes his publ

Alice computes $a($
Bob computes $b(a$
They use this shar
to encrypt with AE

```
>>> 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 large prime p &
base point $(x, y) \in$ Clock(**F**

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

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

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

```
>>> 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 large prime p & **base point** $(x, y) \in \text{Clock}(\mathbf{F}_p)$.

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

```
scalarmult(n,P):
 n == 0: \
turn (Fp(0),Fp(1))
 n == 1: return P
= scalarmult(n//2,P)
= clockadd(Q,Q)
 n % 2: Q = clockadd(P,Q)
turn Q

oursixdigitsecret
armult(n,P)
 736284)
```

figure out our secret n?

Clock cryptography

The "Clock Diffie–Hellman protocol":

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

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

Alic
secret

Alic
publi
$a(X$

{Alice,
shared
$ab(X$

```
(n,P):

),Fp(1))
eturn P
lt(n//2,P)
(Q,Q)
= clockadd(P,Q)

tsecret
)
```

t our secret n?

Clock cryptography

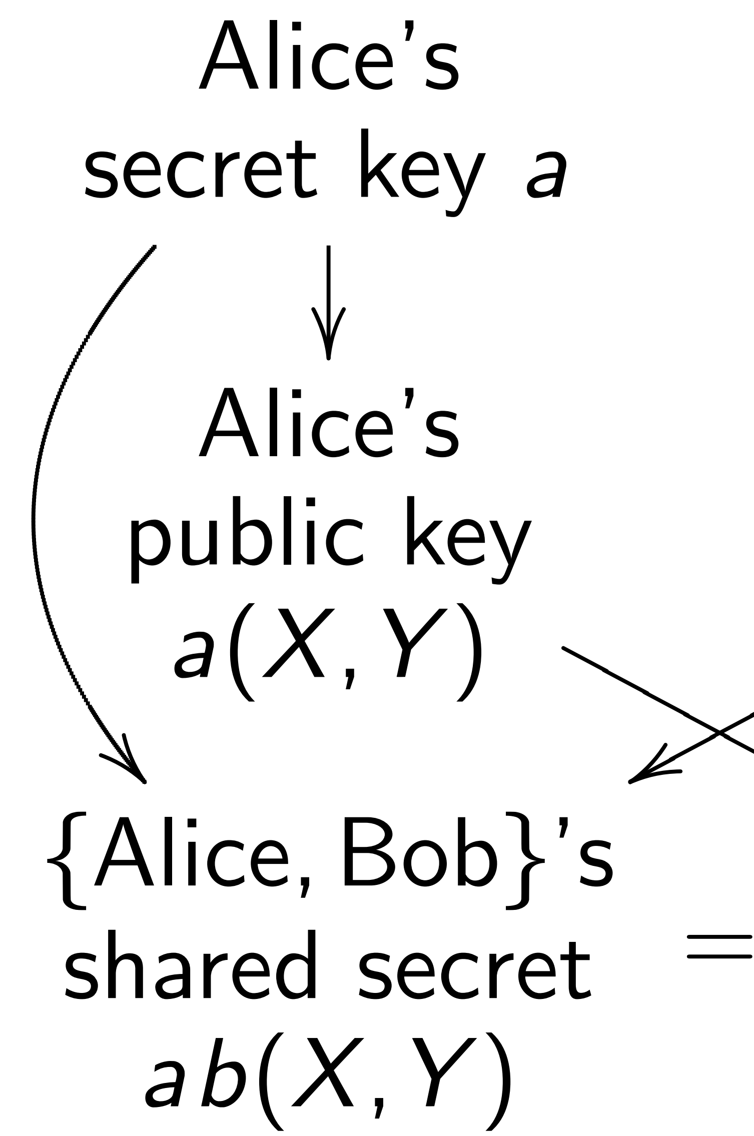
The "Clock Diffie–Hellman protocol":

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

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



(P,Q)

et n?

Clock cryptography

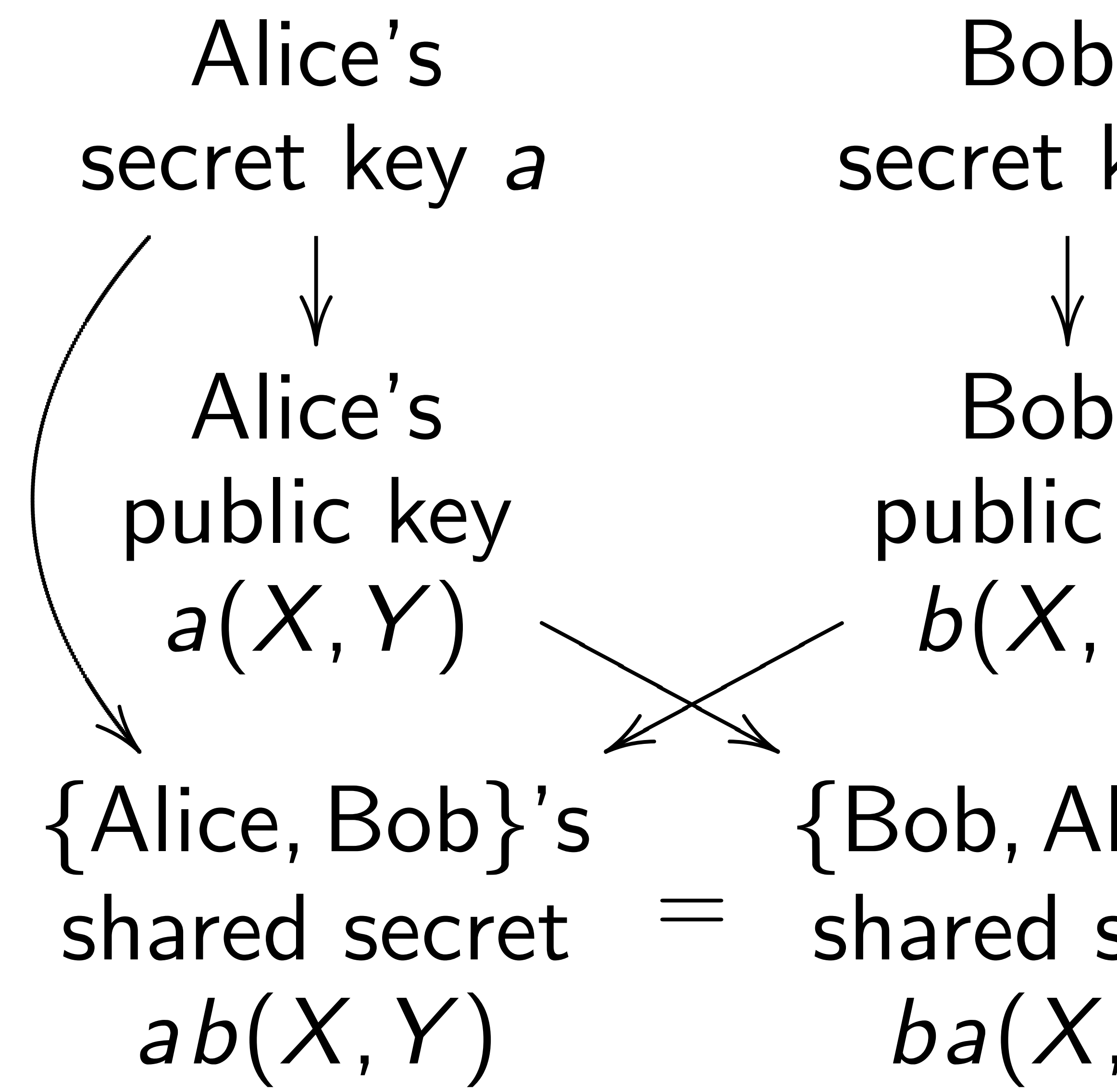
The "Clock Diffie–Hellman protocol":

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

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



Clock cryptography

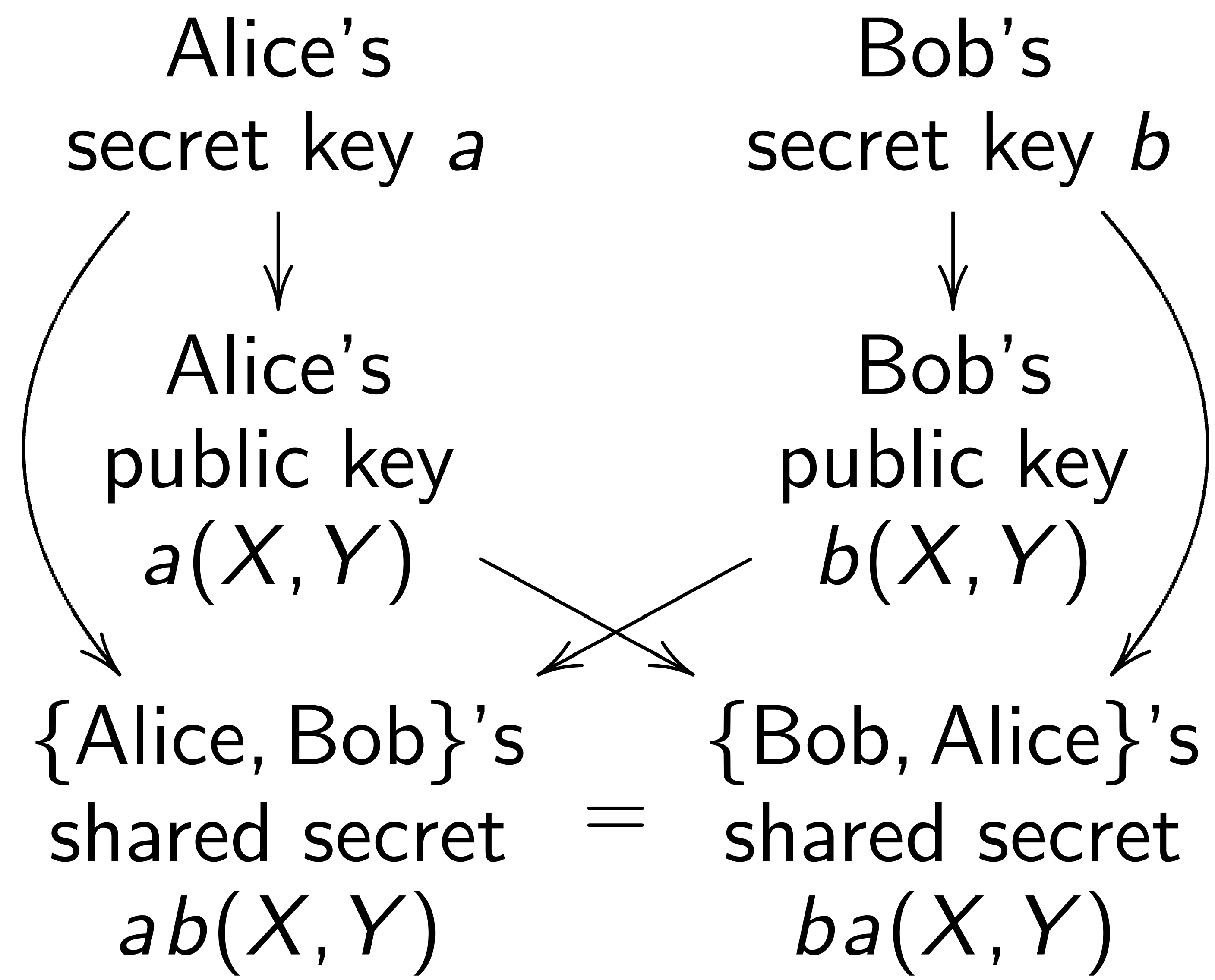
The "Clock Diffie–Hellman protocol":

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

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



Clock cryptography

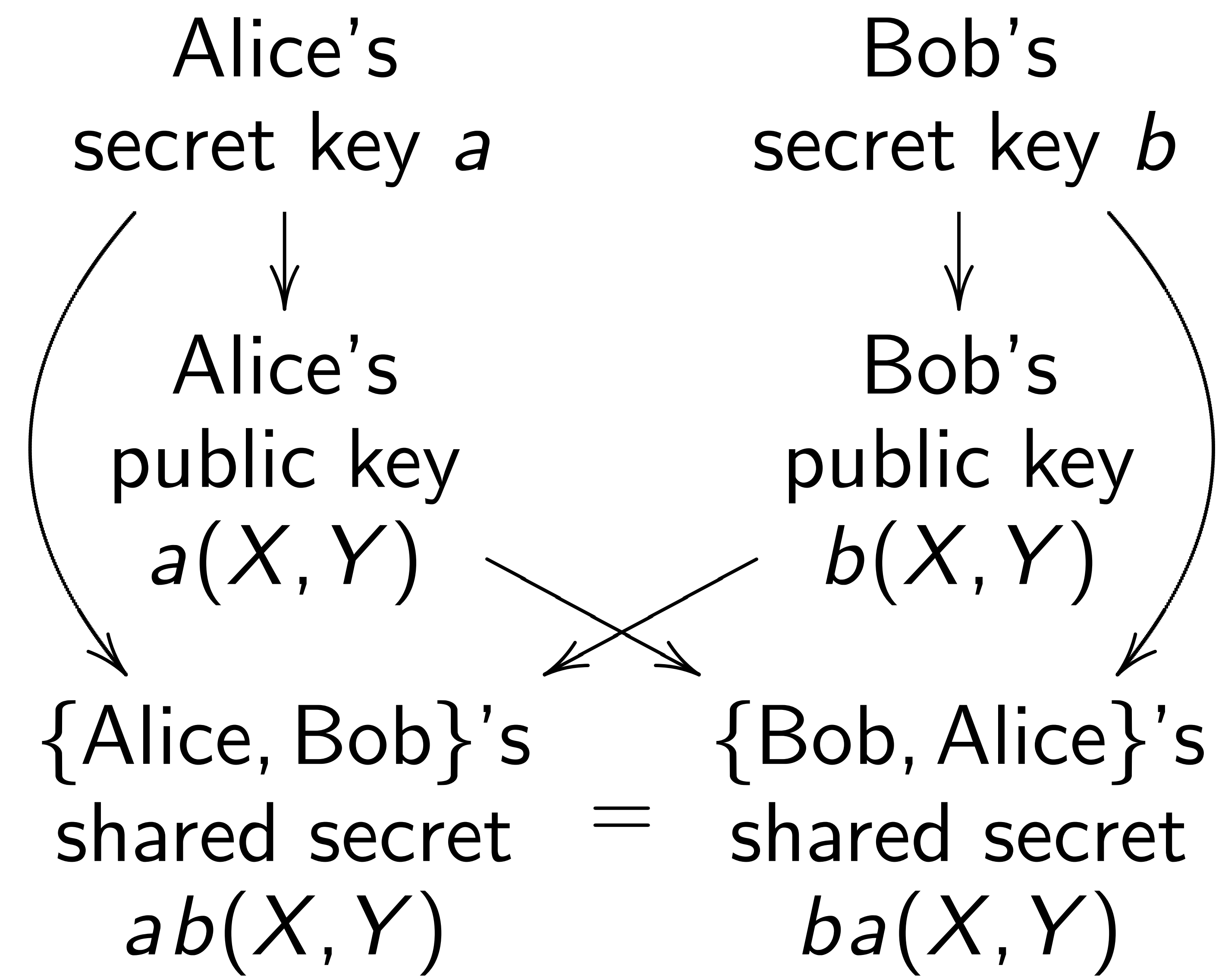
The "Clock Diffie–Hellman protocol":

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

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



Warning #1:
Many p are unsafe!

Warning #2:
Clocks aren't elliptic!
To match RSA-3072 security need $p \approx 2^{1536}$.

yptography

ock Diffie–Hellman

”:

dize large prime p &

oint $(x, y) \in \text{Clock}(\mathbf{F}_p)$.

ooses big secret a,

es her public key $a(x, y)$.

oses big secret b,

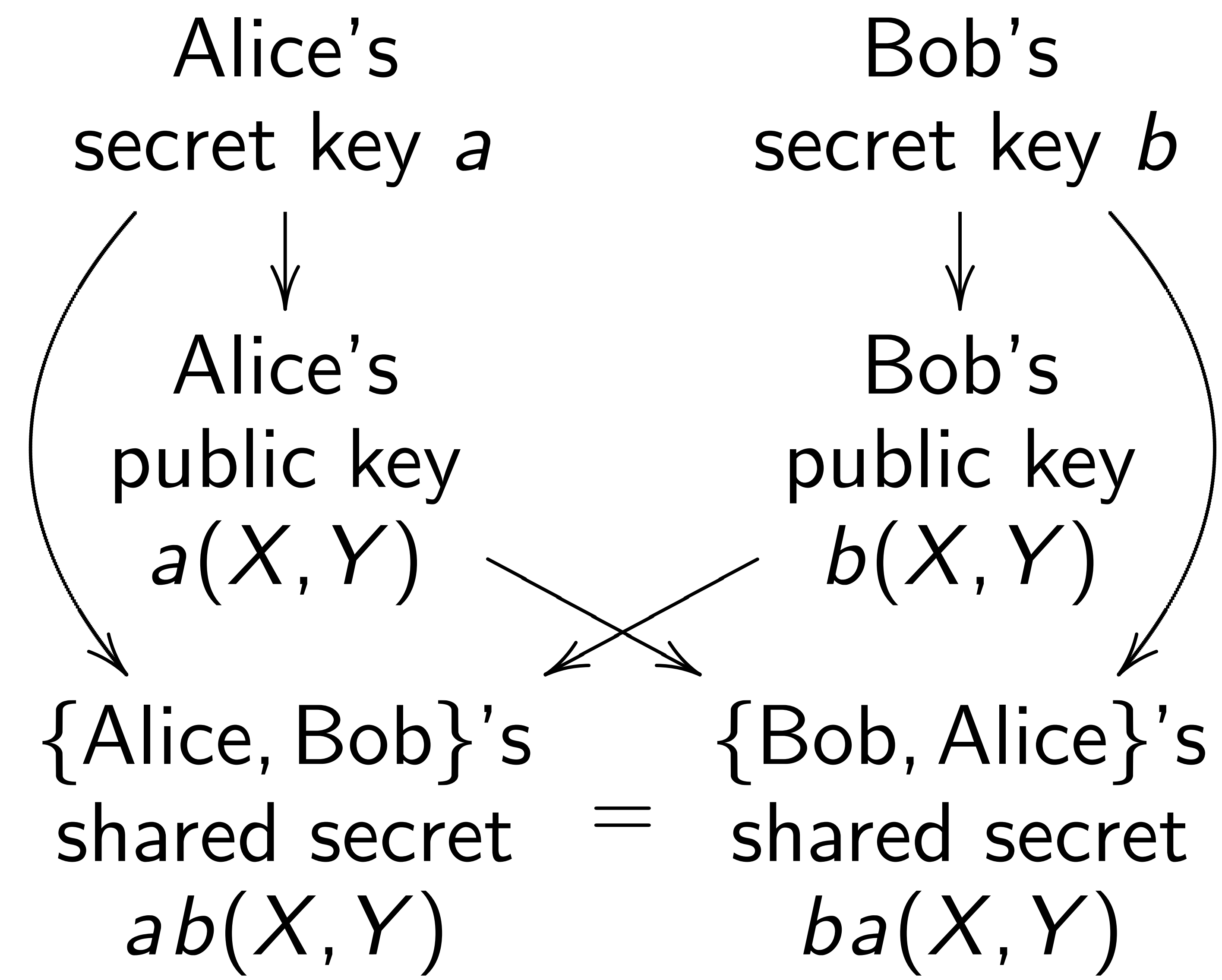
es his public key $b(x, y)$.

mputes $a(b(x, y))$.

nputes $b(a(x, y))$.

e this shared secret

pt with AES-GCM etc.



Warning #1:

Many p are unsafe!

Warning #2:

Clocks aren't elliptic!

To match RSA-3072 security

need $p \approx 2^{1536}$.

Warning

Attacker

public ke

Attacker

Alice use

Often at

each ope

Alice, no

This rev

Break by

2011 Br

y

-Hellman

prime p &

$\in \text{Clock}(\mathbf{F}_p)$.

secret a,

lic key $a(x, y)$.

ecret b,

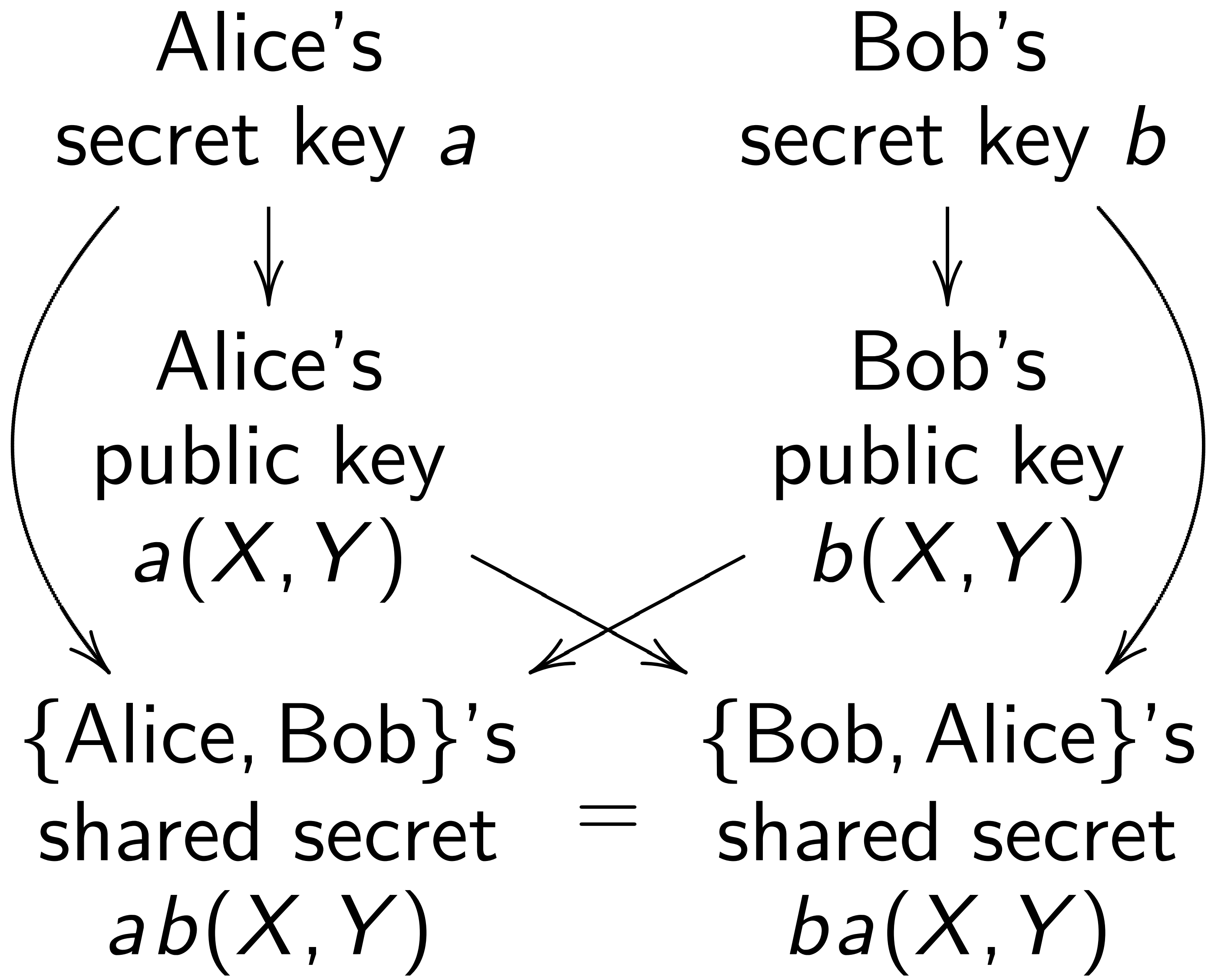
ic key $b(x, y)$.

$b(x, y))$.

$a(x, y))$.

red secret

ES-GCM etc.



Warning #1:

Many p are unsafe!

Warning #2:

Clocks aren't elliptic!

To match RSA-3072 security

need $p \approx 2^{1536}$.

Warning #3:

Attacker sees mor

public keys $a(x, y$

Attacker sees how

Alice uses to comp

Often attacker ca

each operation pe

Alice, not just tot

This reveals secre

Break by timing a

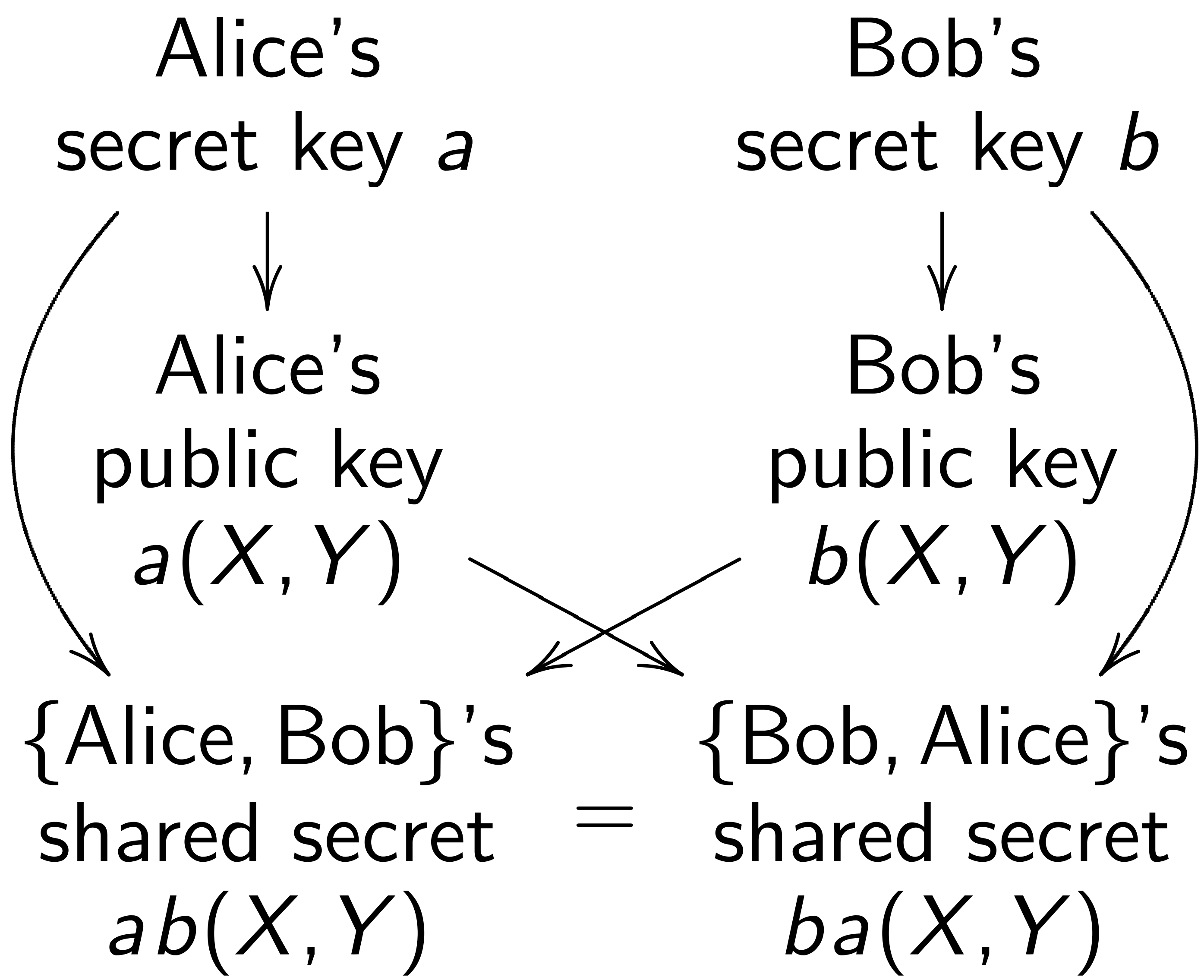
2011 Brumley–Tu

$\mathbf{F}_p$).

x, y).

$, y$).

etc.



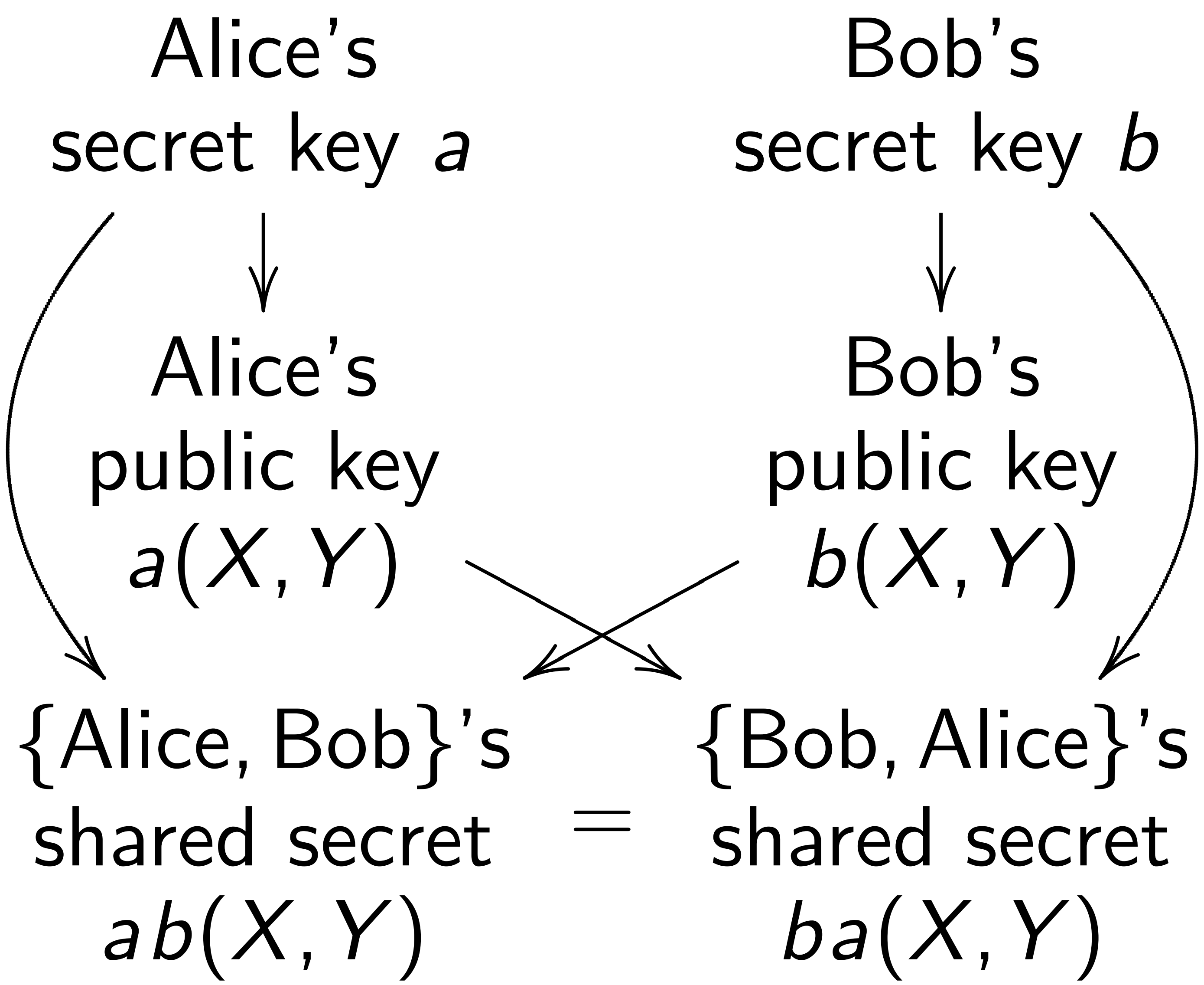
Warning #1:
Many p are unsafe!

Warning #2:
Clocks aren't elliptic!
To match RSA-3072 security need $p \approx 2^{1536}$.

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

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

Break by timing attacks, e.g
2011 Brumley–Tuveri.



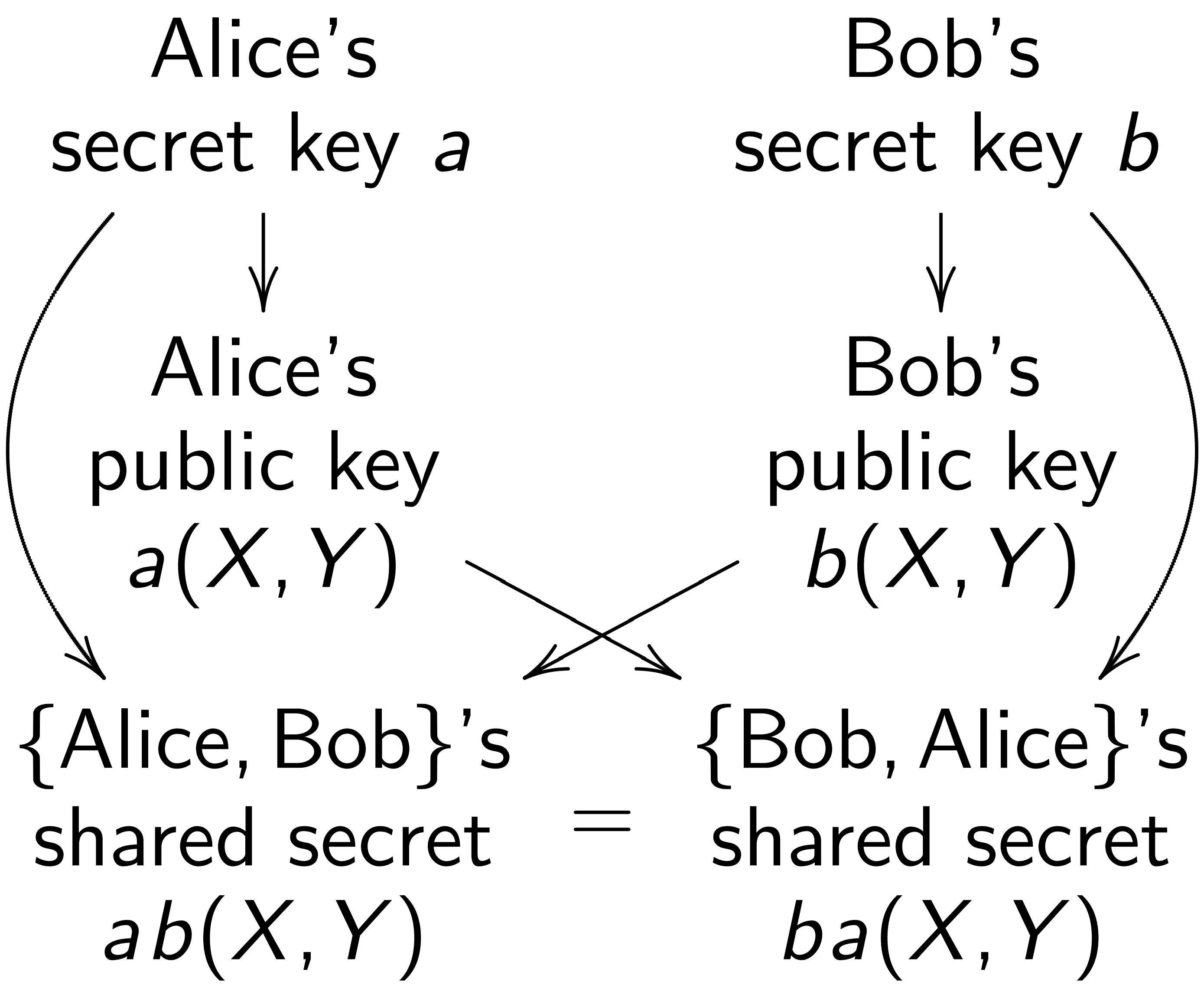
Warning #1:
Many p are unsafe!

Warning #2:
Clocks aren't elliptic!
To match RSA-3072 security
need $p \approx 2^{1536}$.

Warning #3:
Attacker sees more than
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.

Break by timing attacks, e.g.,
2011 Brumley–Tuveri.



Warning #1:
Many p are unsafe!

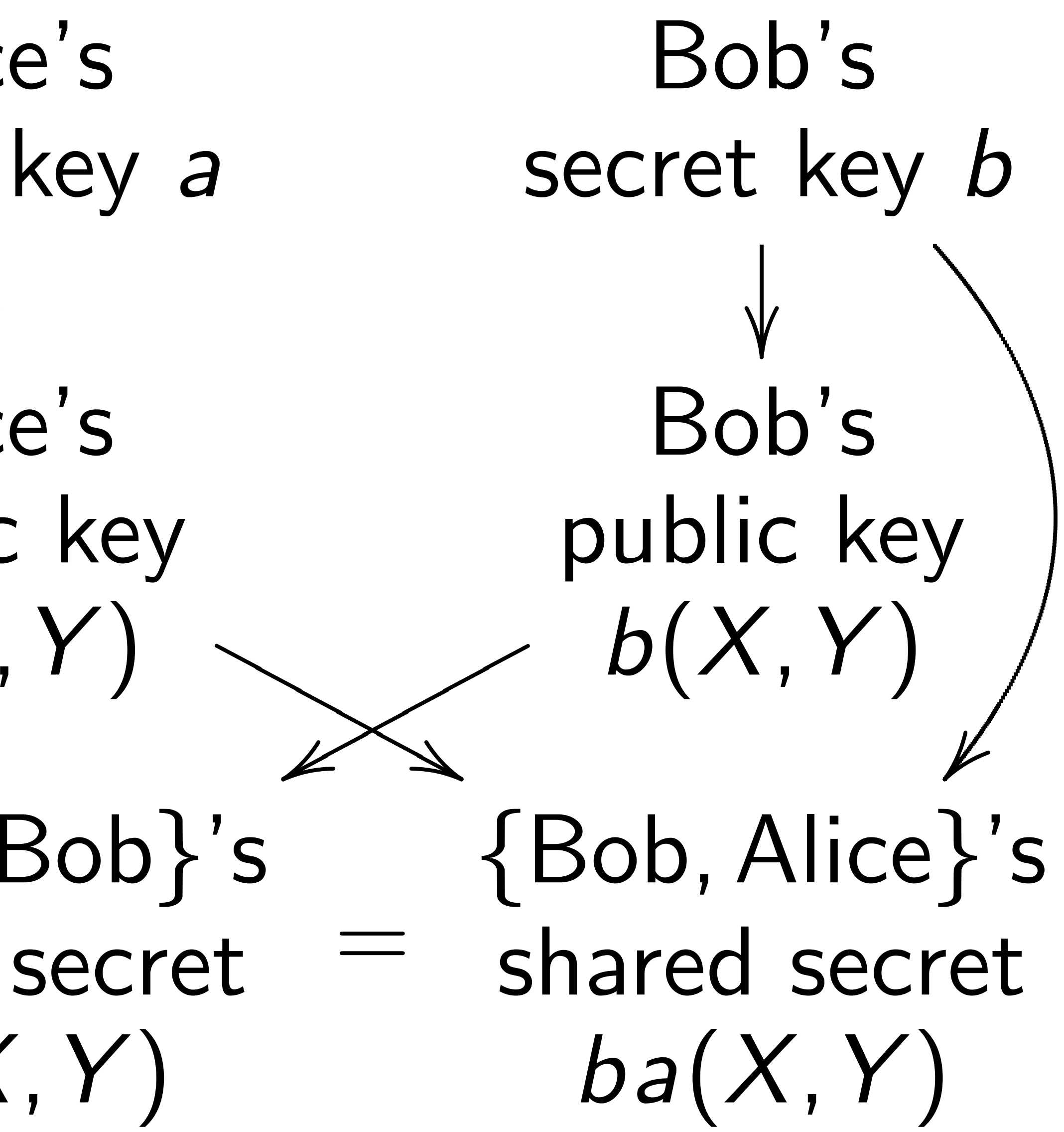
Warning #2:
Clocks aren't elliptic!
To match RSA-3072 security need $p \approx 2^{1536}$.

Warning #3:
Attacker sees more than 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.

Break by timing attacks, e.g., 2011 Brumley–Tuveri.

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



; #1:
are unsafe!

; #2:
aren't elliptic!

th RSA-3072 security
$\approx 2^{1536}$.

Warning #3:
Attacker sees more than 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.

Break by timing attacks, e.g., 2011 Brumley–Tuveri.

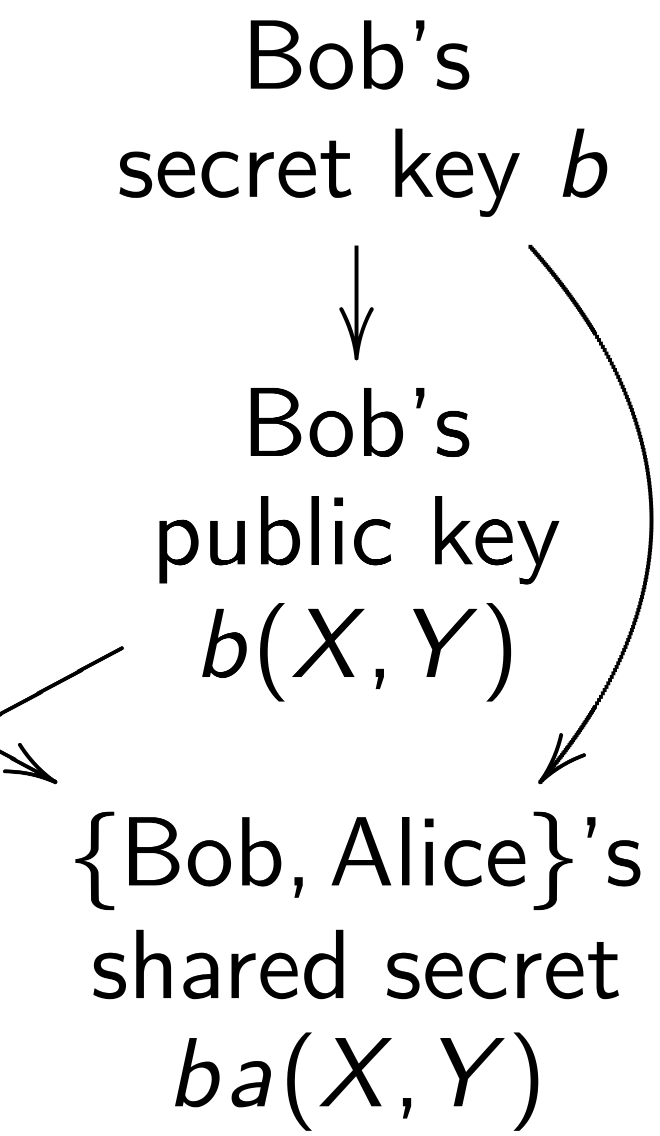
Fix: **constant-time** code, performing same operations no matter what scalar is.

Exercise

How ma
do you r
$(x_1 y_2 +$

How ma
do you r
i.e. to c
$(x_1 y_1 +$

How car
computa
cheaper
Assume



e!

tic!

72 security

Warning #3:
Attacker sees more than 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.

Break by timing attacks, e.g., 2011 Brumley–Tuveri.

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

Exercise

How many multipl
do you need to co
$(x_1y_2 + y_1x_2, y_1y$

How many multipl
do you need to do
i.e. to compute
$(x_1y_1 + y_1x_1, y_1y$

How can you optir
computation if squ
cheaper than mult
Assume $\mathbf{S} < \mathbf{M} <$

's
key b

's
key
$Y)$

lice}'s
secret
$, Y)$

y

Warning #3:
Attacker sees more than 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.

Break by timing attacks, e.g., 2011 Brumley–Tuveri.

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

<u>Exercise</u>

How many multiplications do you need to compute $(x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)$

How many multiplications do you need to double a poi i.e. to compute $(x_1y_1 + y_1x_1, y_1y_1 - x_1x_1)$

How can you optimize the computation if squarings are cheaper than multiplications Assume $\mathbf{S} < \mathbf{M} < 2\mathbf{S}$.

Warning #3:
Attacker sees more than 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.

Break by timing attacks, e.g., 2011 Brumley–Tuveri.

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

<u>Exercise</u>

How many multiplications do you need to compute $(x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)$?

How many multiplications do you need to double a point, i.e. to compute $(x_1y_1 + y_1x_1, y_1y_1 - x_1x_1)$?

How can you optimize the computation if squarings are cheaper than multiplications? Assume $\mathbf{S} < \mathbf{M} < 2\mathbf{S}$.

; #3:
r sees more than
eys $a(x, y)$ and $b(x, y)$.

r sees how much *time*
es to compute $a(b(x, y))$.
ttacker can see time for
eration performed by
ot just total time.
eals secret scalar a.

y timing attacks, e.g.,
umley–Tuveri.

nstant-time code,
ng same operations
er what scalar is.

Exercise

How many multiplications
do you need to compute
$(x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)$?

How many multiplications
do you need to double a point,
i.e. to compute
$(x_1y_1 + y_1x_1, y_1y_1 - x_1x_1)$?

How can you optimize the
computation if squarings are
cheaper than multiplications?
Assume $\mathbf{S} < \mathbf{M} < 2\mathbf{S}$.

Addition

Change
and Bob

$x^2 + y^2$
Sum of
$((x_1y_2+$
$(y_1y_2-$

e than
) and $b(x, y)$.

much *time*
pute $a(b(x, y))$.
n see time for
rformed by
al time.
t scalar a.

ttacks, e.g.,
veri.

ne code,
operations
calar is.

Exercise

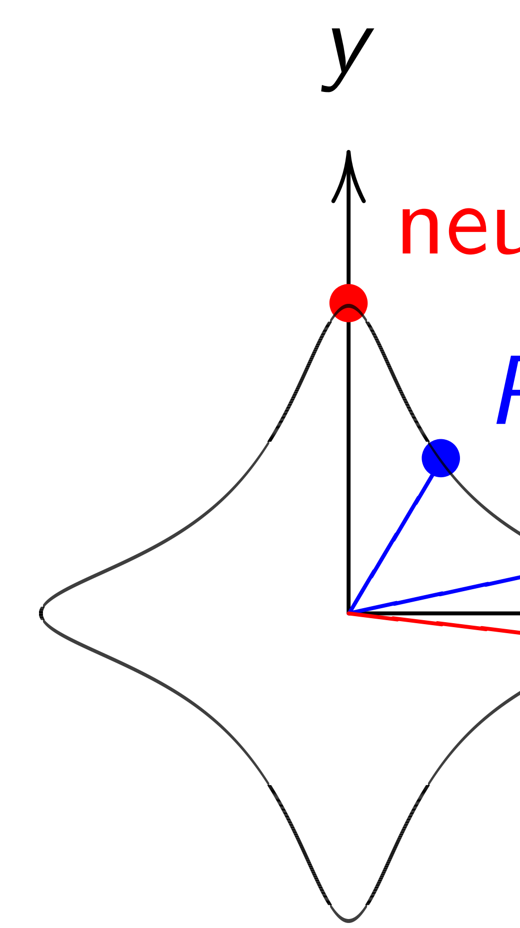
How many multiplications do you need to compute $(x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)$?

How many multiplications do you need to double a point, i.e. to compute $(x_1y_1 + y_1x_1, y_1y_1 - x_1x_1)$?

How can you optimize the computation if squarings are cheaper than multiplications? Assume $\mathbf{S} < \mathbf{M} < 2\mathbf{S}$.

Addition on an Ed

Change the curve
and Bob work.



$x^2 + y^2 = 1 - 30x$
Sum of (x_1, y_1) ar
$((x_1y_2+y_1x_2)/(1-$
$(y_1y_2-x_1x_2)/(1-$

y).

ne

x, y)).

for

y

s.,

Exercise

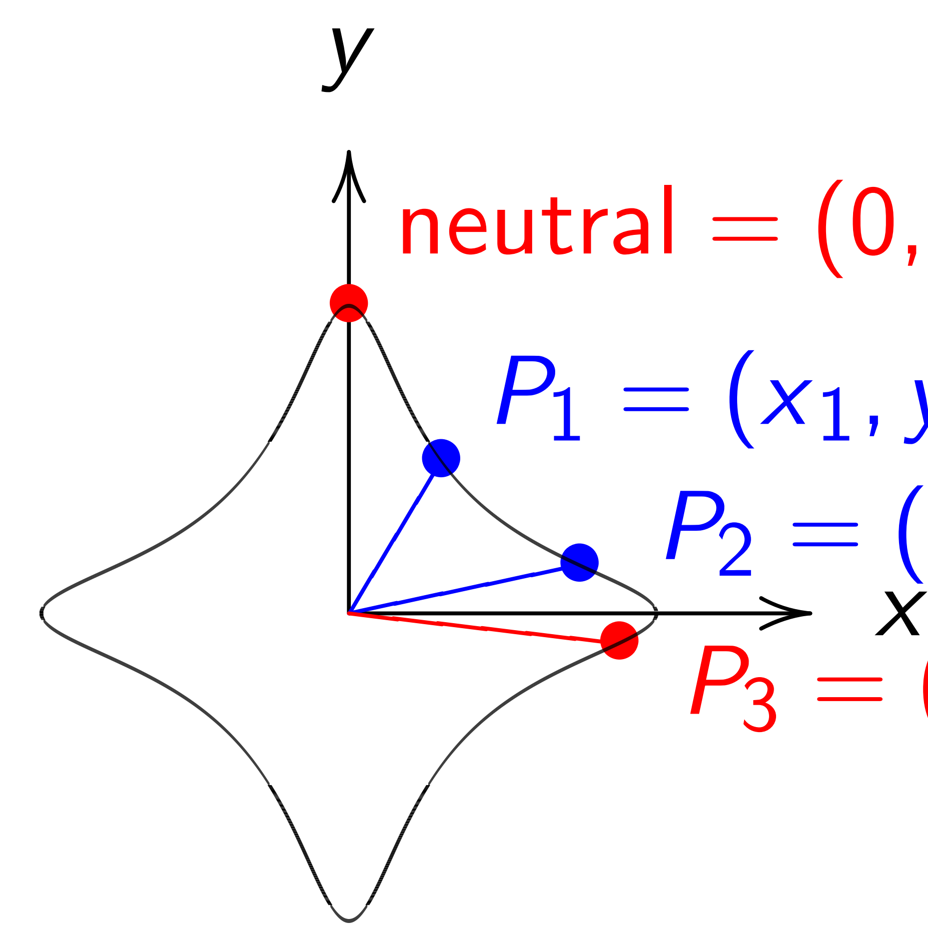
How many multiplications do you need to compute $(x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)$?

How many multiplications do you need to double a point, i.e. to compute $(x_1y_1 + y_1x_1, y_1y_1 - x_1x_1)$?

How can you optimize the computation if squarings are cheaper than multiplications? Assume $\mathbf{S} < \mathbf{M} < 2\mathbf{S}$.

Addition on an Edwards cur

Change the curve on which and Bob work.



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

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

$((x_1y_2+y_1x_2)/(1-30x_1x_2y$

$(y_1y_2-x_1x_2)/(1+30x_1x_2y$

Exercise

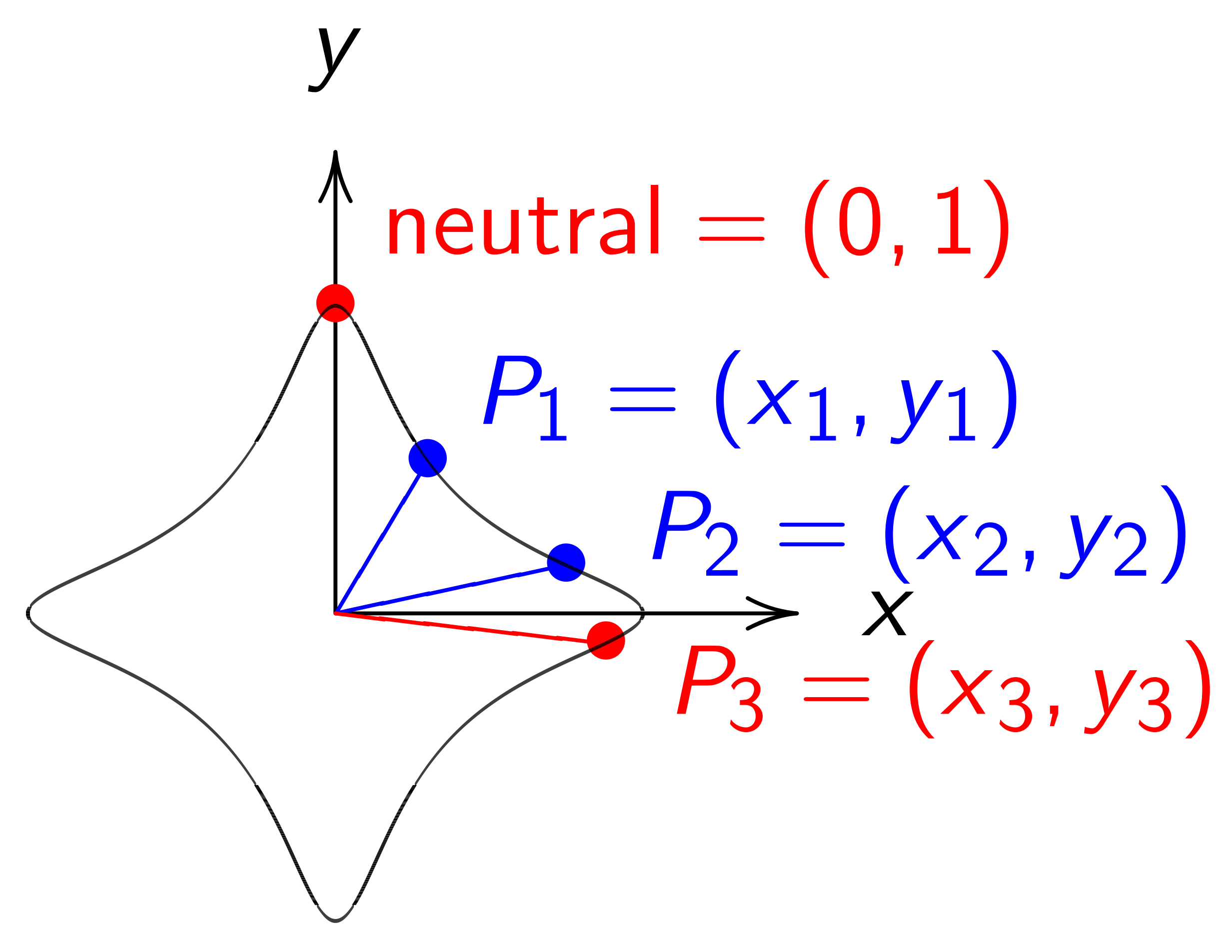
How many multiplications do you need to compute $(x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)$?

How many multiplications do you need to double a point, i.e. to compute $(x_1y_1 + y_1x_1, y_1y_1 - x_1x_1)$?

How can you optimize the computation if squarings are cheaper than multiplications? Assume $\mathbf{S} < \mathbf{M} < 2\mathbf{S}$.

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

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

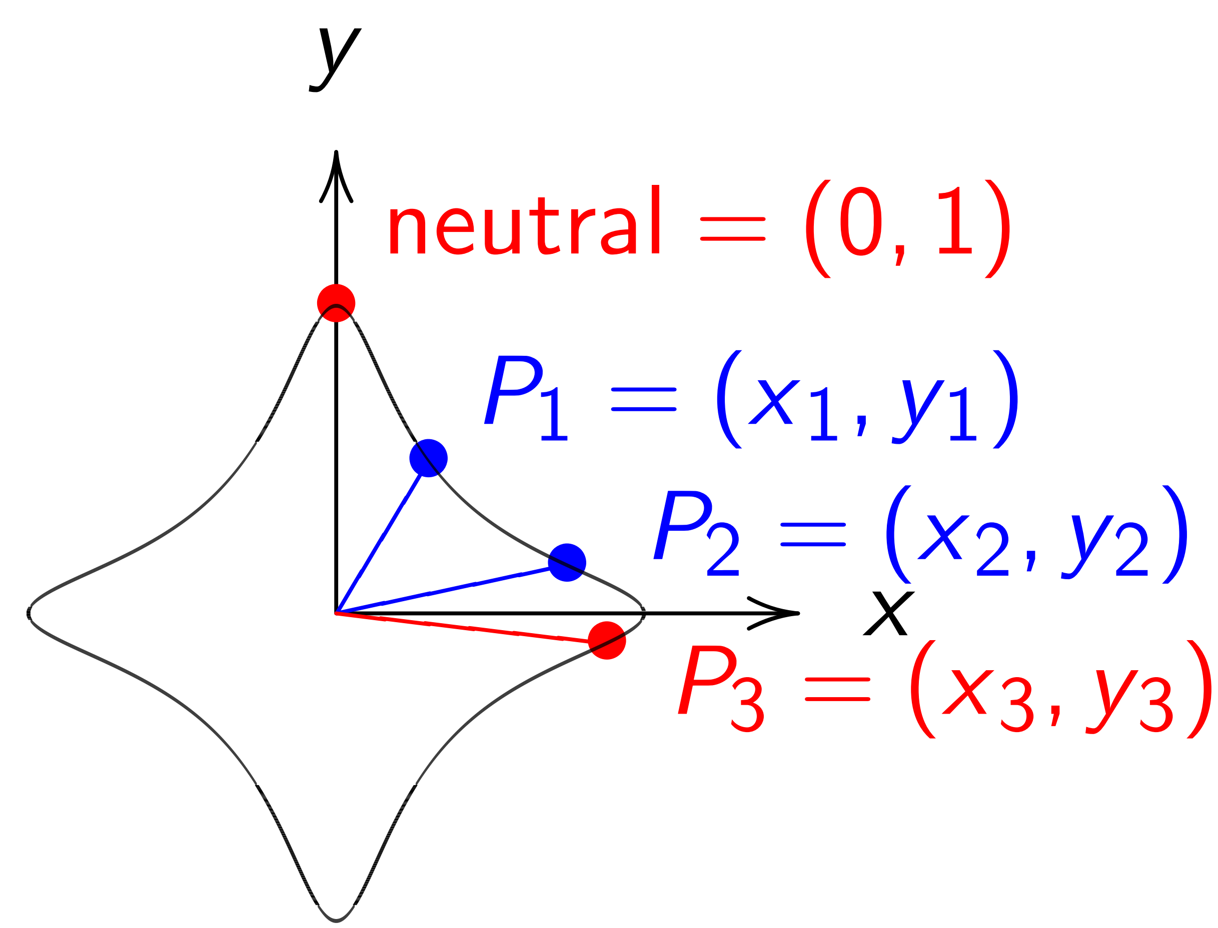
ny multiplications
need to compute
$y_1x_2, y_1y_2 - x_1x_2$)?

ny multiplications
need to double a point,
ompute
$y_1x_1, y_1y_1 - x_1x_1$)?

n you optimize the
ation if squarings are
than multiplications?
$\mathbf{S} < \mathbf{M} < 2\mathbf{S}$.

Addition on an Edwards curve

Change the curve on which Alice and Bob work.

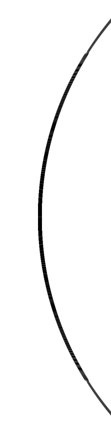


$x^2 + y^2 = 1 - 30x^2y^2$.

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 cloc



$x^2 + y^2$
Sum of (
$(x_1y_2 +$
$y_1y_2 -$

lications
mpute
$_2 - x_1x_2)$?

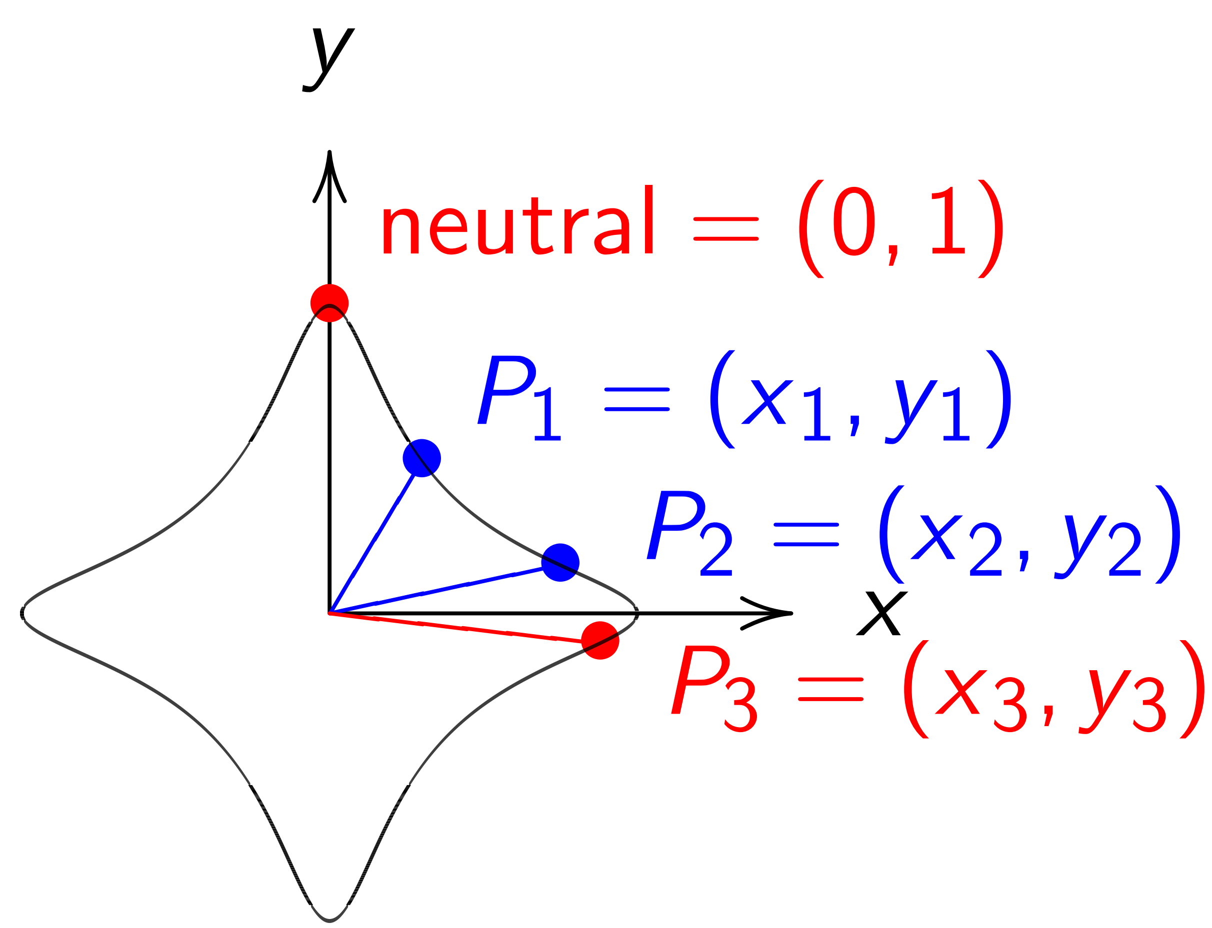
lications
uble a point,

$_1 - x_1x_1)$?

mize the
uarings are
tiplications?
2**S**.

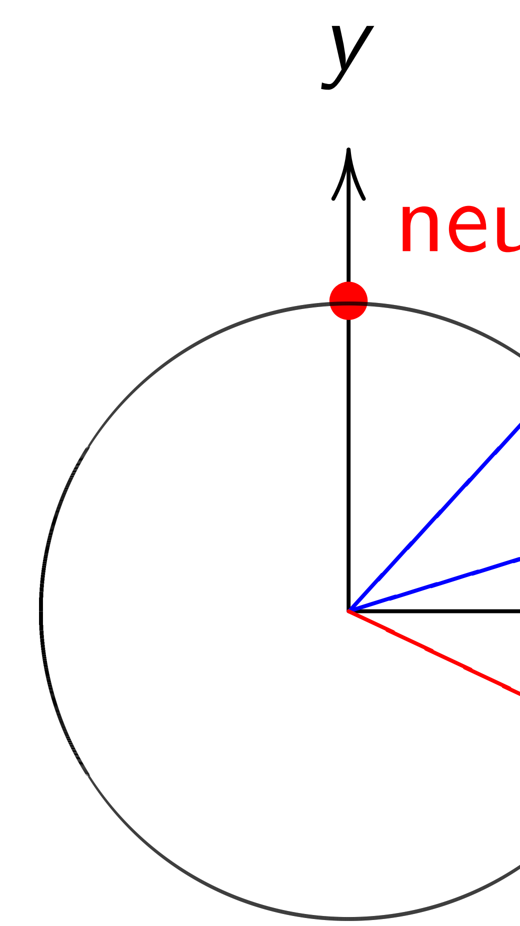
Addition on an Edwards curve

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, f



$x^2 + y^2 = 1$.
Sum of (x_1, y_1) ar
$(x_1y_2 + y_1x_2,$
$y_1y_2 - x_1x_2)$.

?

nt,

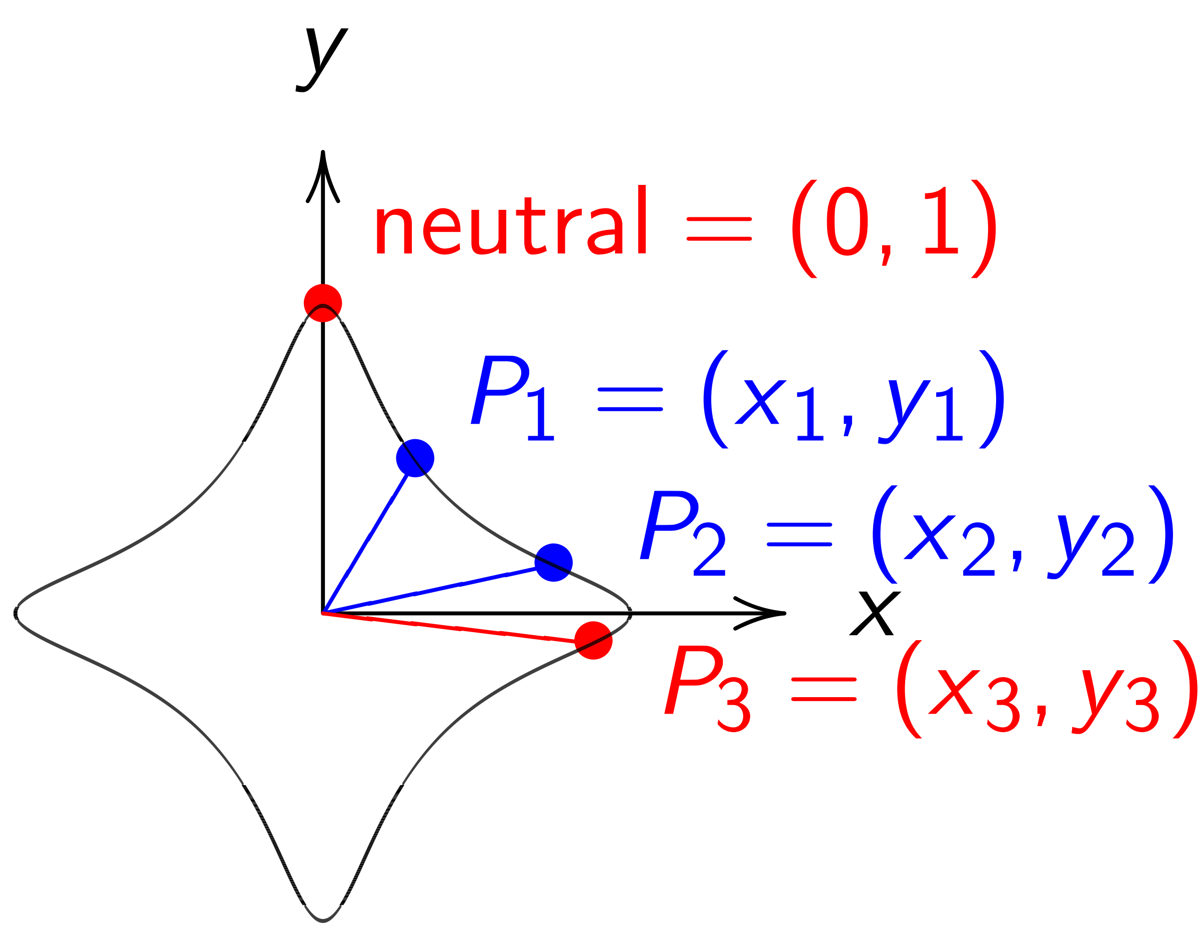
?

e

s?

Addition on an Edwards curve

Change the curve on which Alice and Bob work.



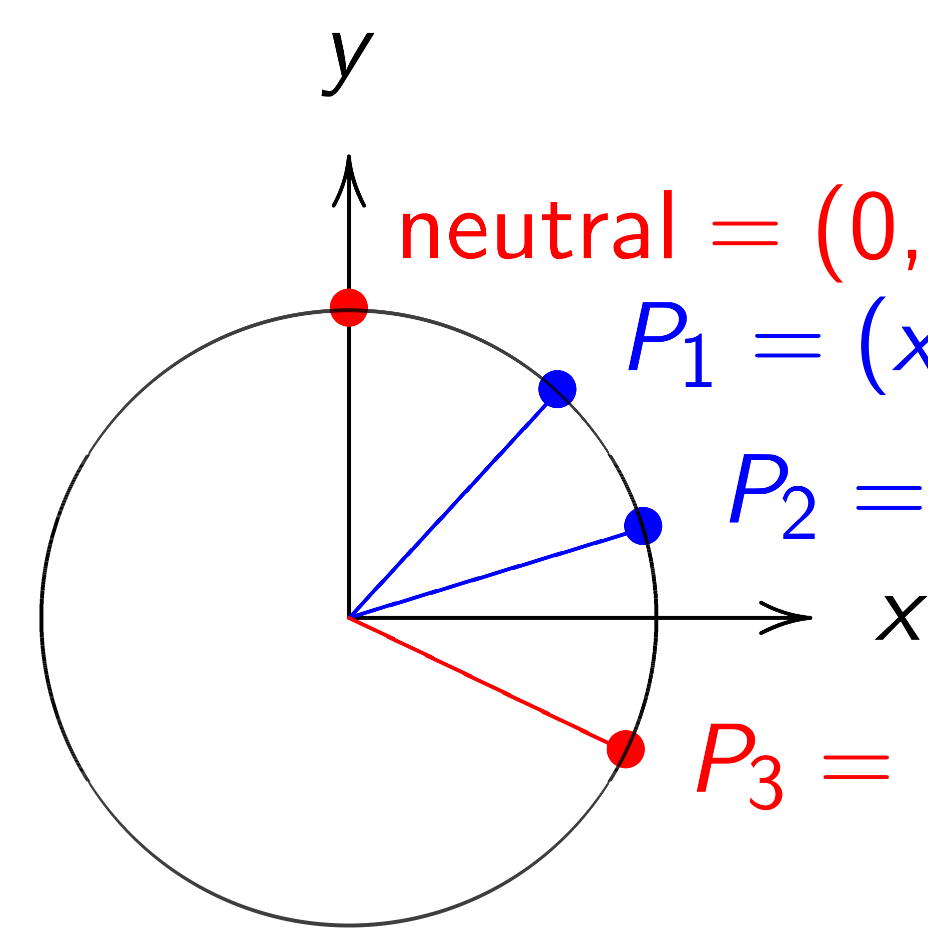
$x^2 + y^2 = 1 - 30x^2y^2$.

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 compar



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

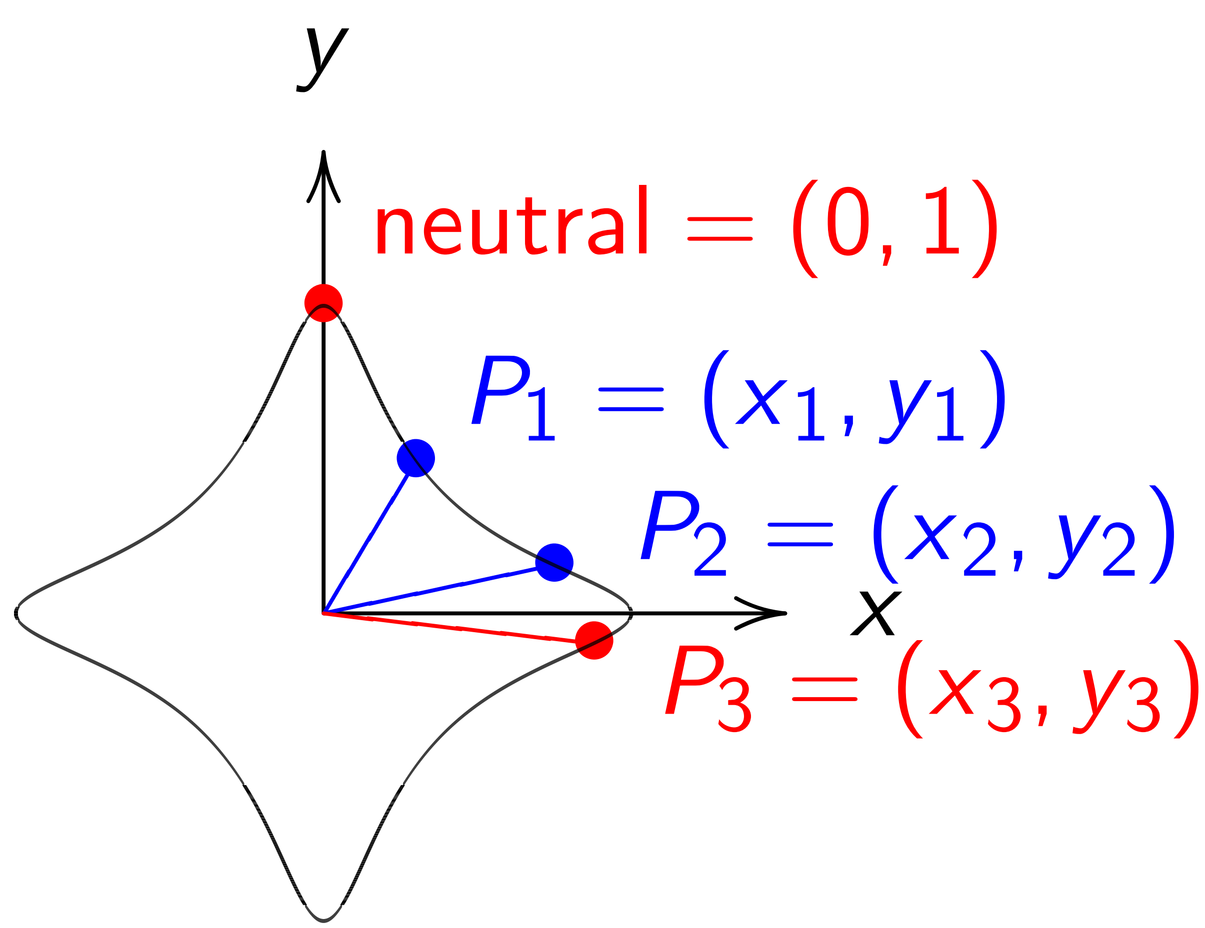
Sum of (x_1, y_1) and (x_2, y_2)

$(x_1y_2 + y_1x_2,$

$y_1y_2 - x_1x_2)$.

Addition on an Edwards curve

Change the curve on which Alice and Bob work.



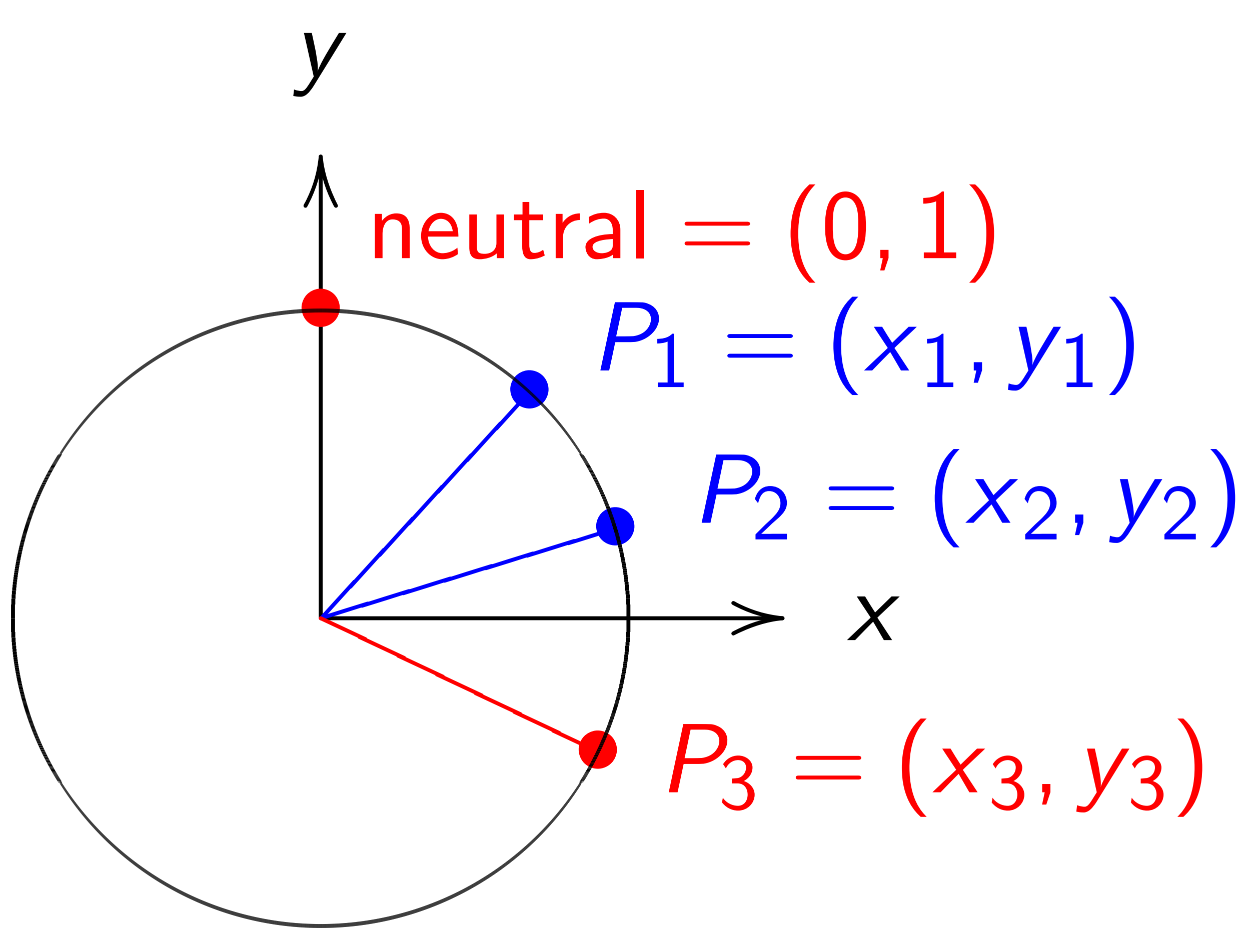
$x^2 + y^2 = 1 - 30x^2y^2$.

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

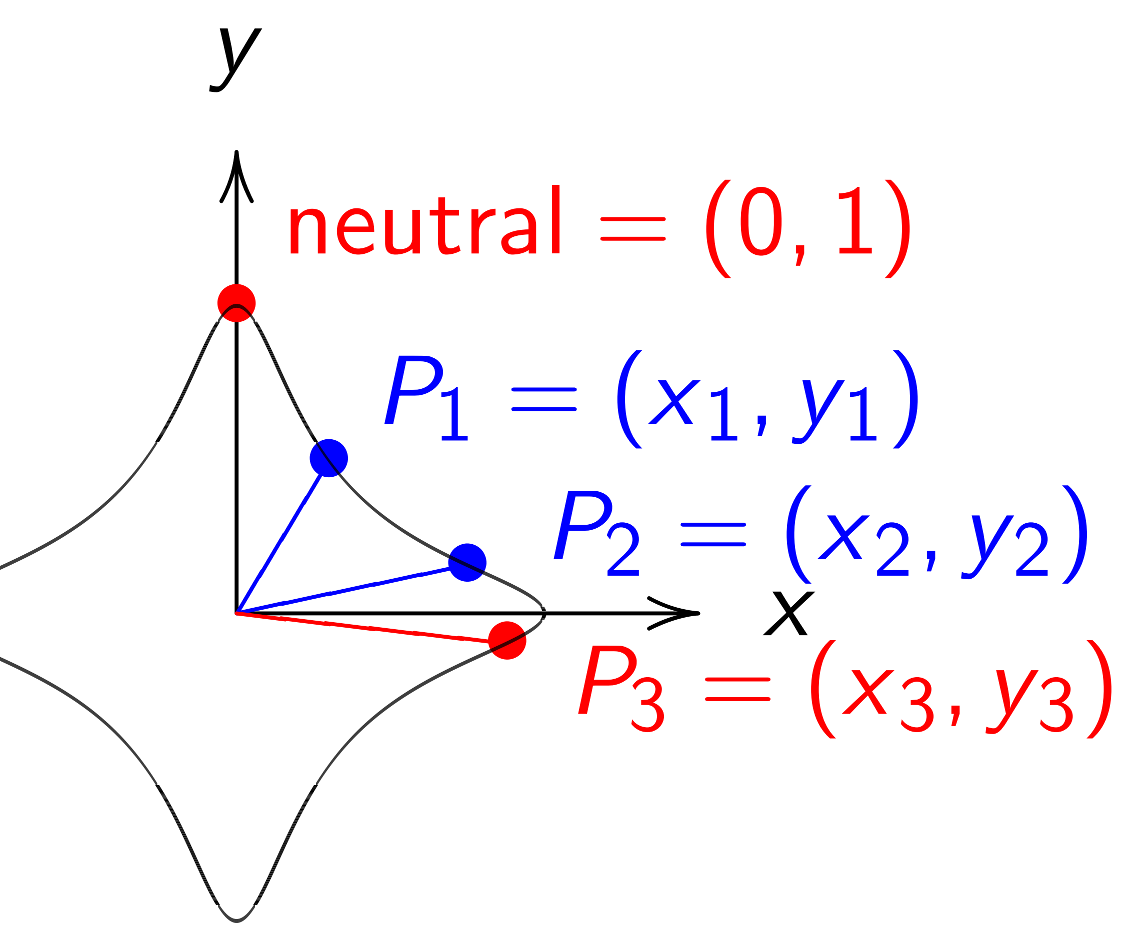
$(x_1y_2 + y_1x_2,$

$y_1y_2 - x_1x_2)$.

n on an Edwards curve

the curve on which Alice

o work.



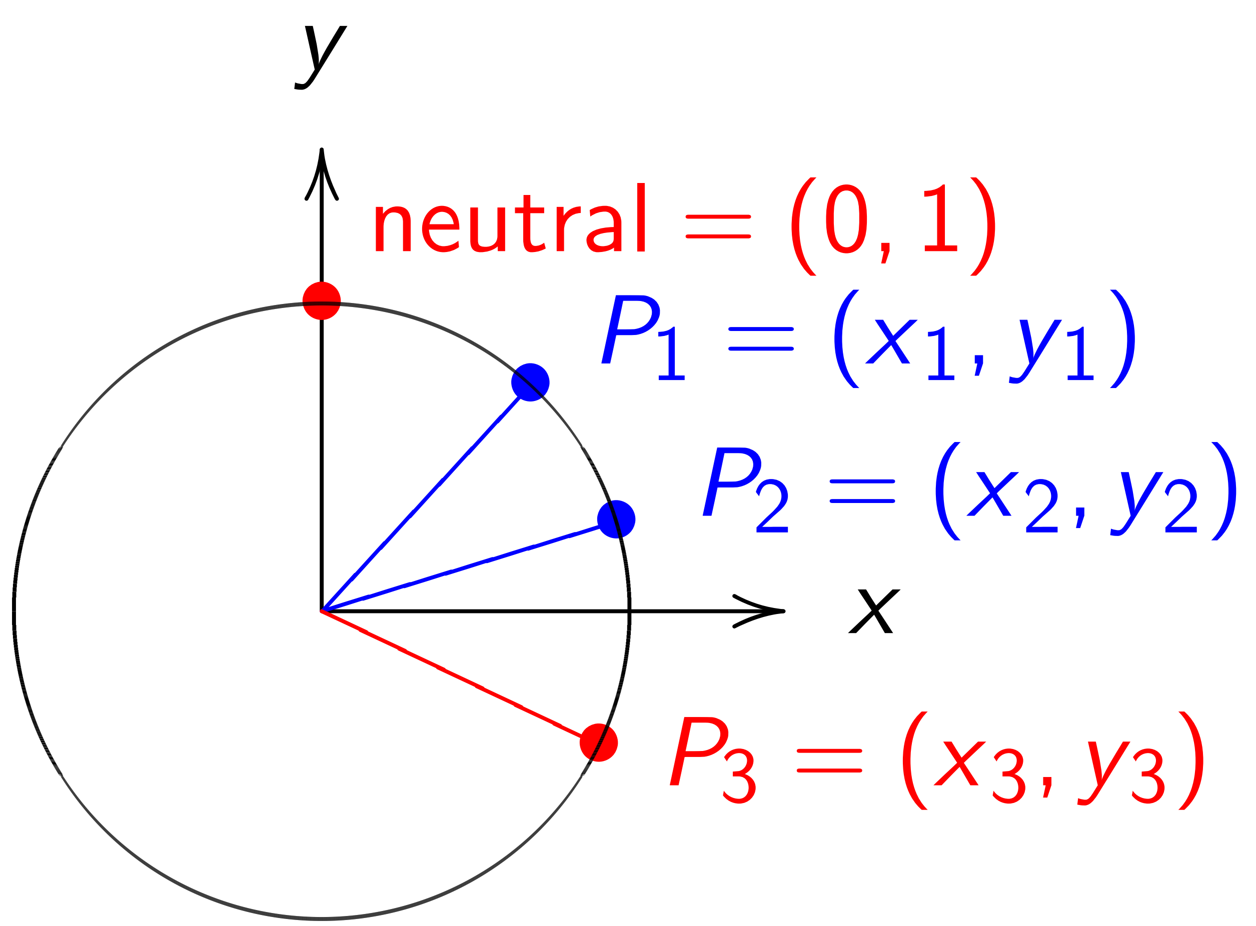
$= 1 - 30x^2y^2$.

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

$\cdot y_1x_2)/(1{-}30x_1x_2y_1y_2)$,

$x_1x_2)/(1{+}30x_1x_2y_1y_2))$.

The clock again, for comparison:



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

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

$(x_1y_2 + y_1x_2,$

$y_1y_2 - x_1x_2)$.

“Hey, th

in the Ec

What if

Answer:

If $x_i = 0$

$1 \pm 30x_1$

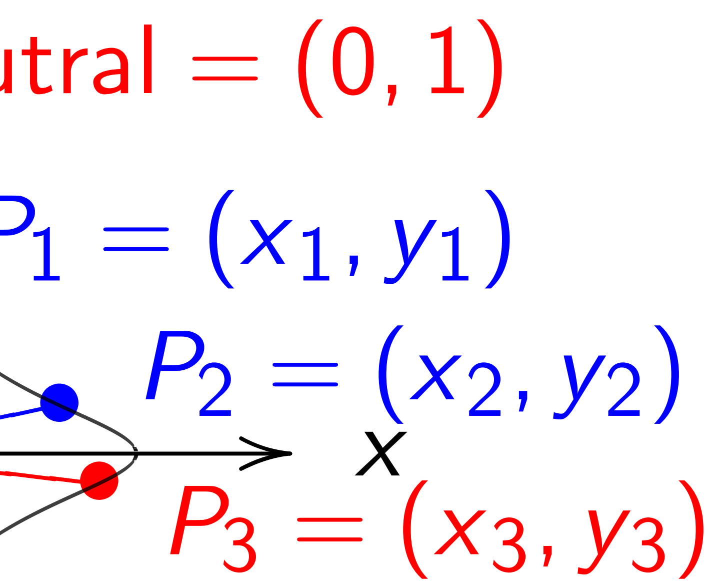
If $x^2 + y$

then $30x$

so $\sqrt{30}$

dwards curve

on which Alice



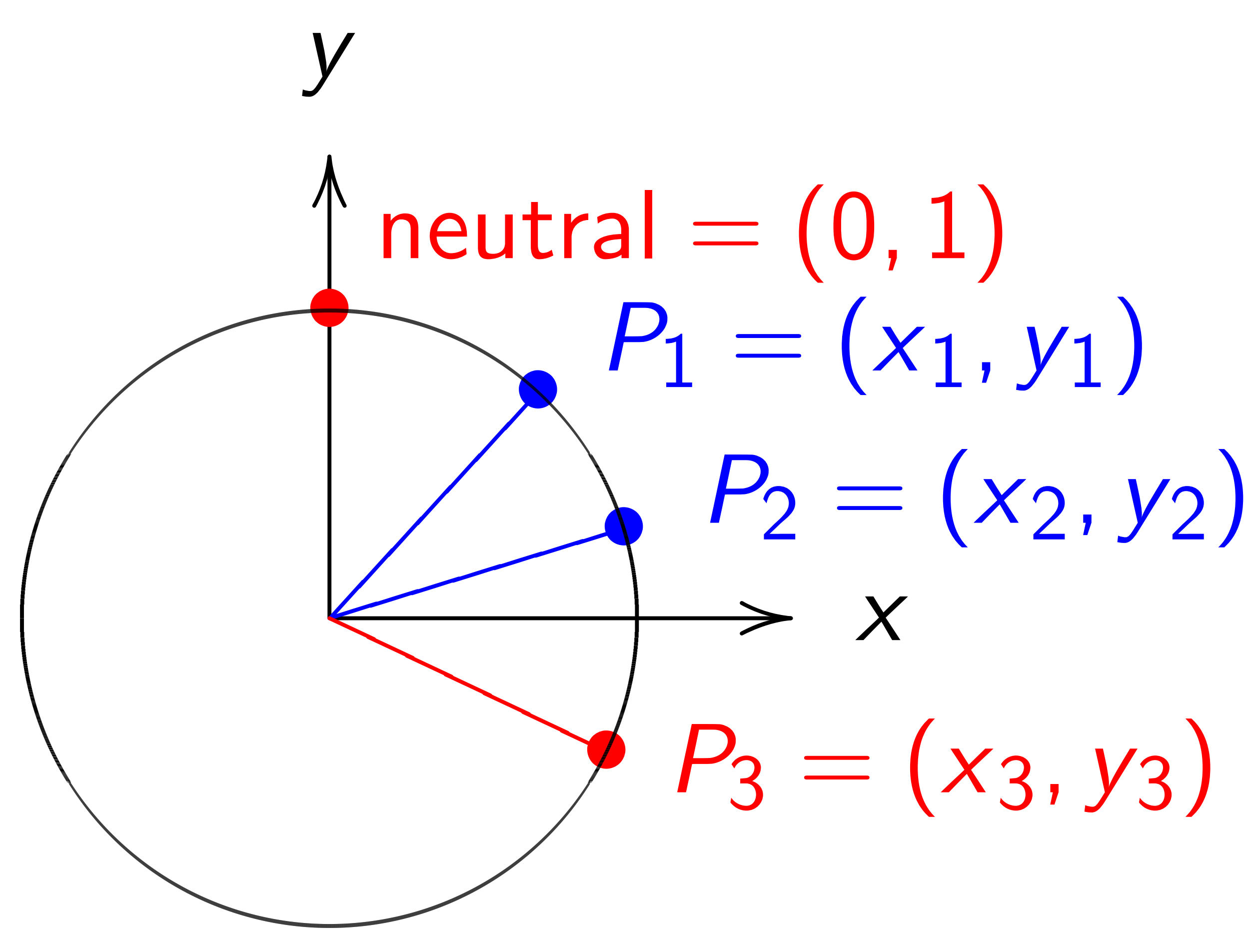
x^2y^2.

nd (x_2, y_2) is

$-30x_1x_2y_1y_2)$,

$+30x_1x_2y_1y_2))$.

The clock again, for comparison:



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

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

$(x_1y_2 + y_1x_2,$

$y_1y_2 - x_1x_2)$.

"Hey, there were d

in the Edwards ad

What if the denom

Answer: They are

If $x_i = 0$ or $y_i = 0$

$1 \pm 30x_1x_2y_1y_2 =$

If $x^2 + y^2 = 1 - 3$

then $30x^2y^2 < 1$

so $\sqrt{30}\,|xy| < 1$.

ve

Alice

1)

$y_1)$

$x_2, y_2)$

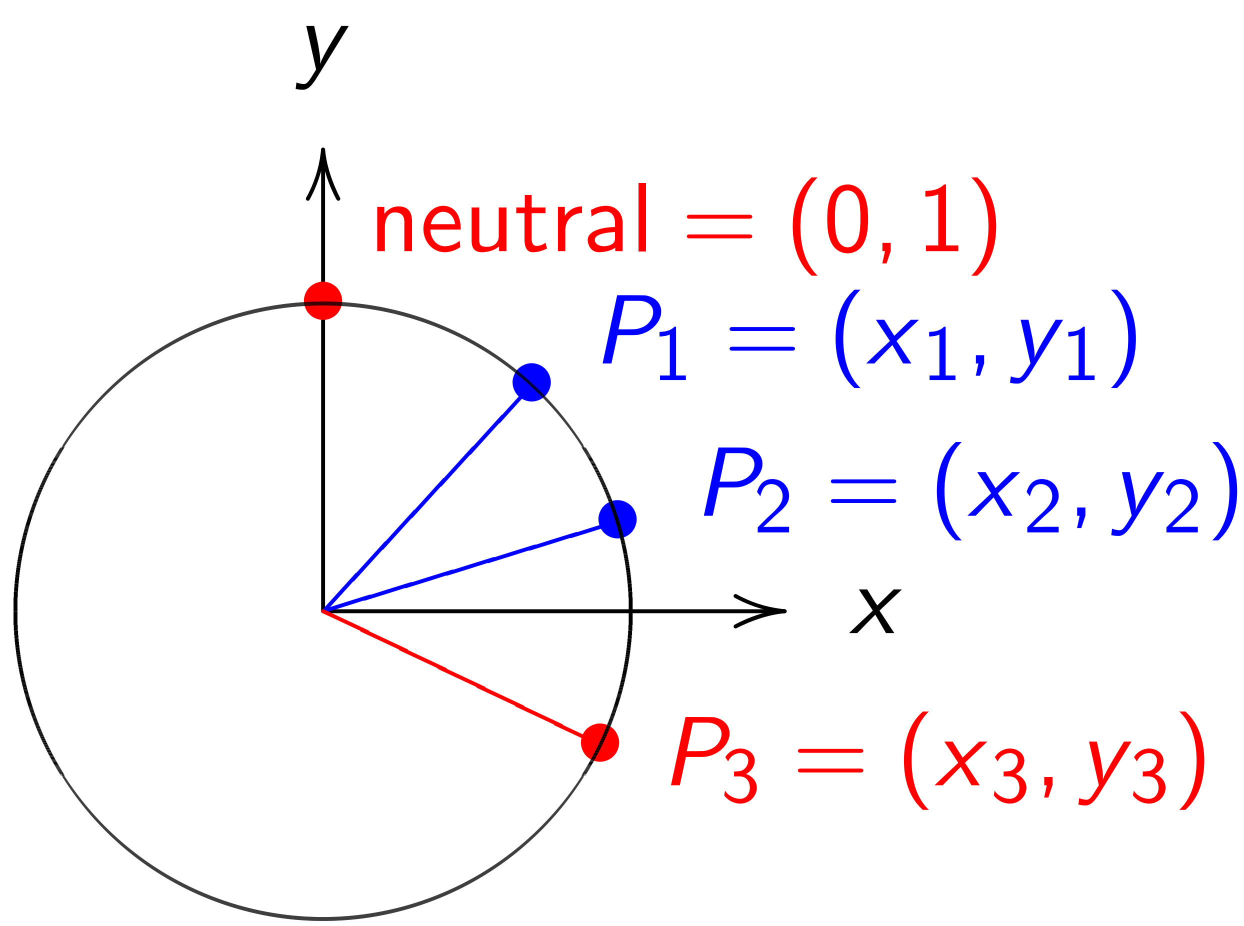
(x_3, y_3)

) is

$_1y_2),$

$_1y_2)).$

The clock again, for comparison:



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

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

$(x_1y_2 + y_1x_2,$

$y_1y_2 - x_1x_2).$

“Hey, there were divisions

in the Edwards addition law

What if the denominators ar

Answer: They aren’t!

If $x_i = 0$ or $y_i = 0$ then

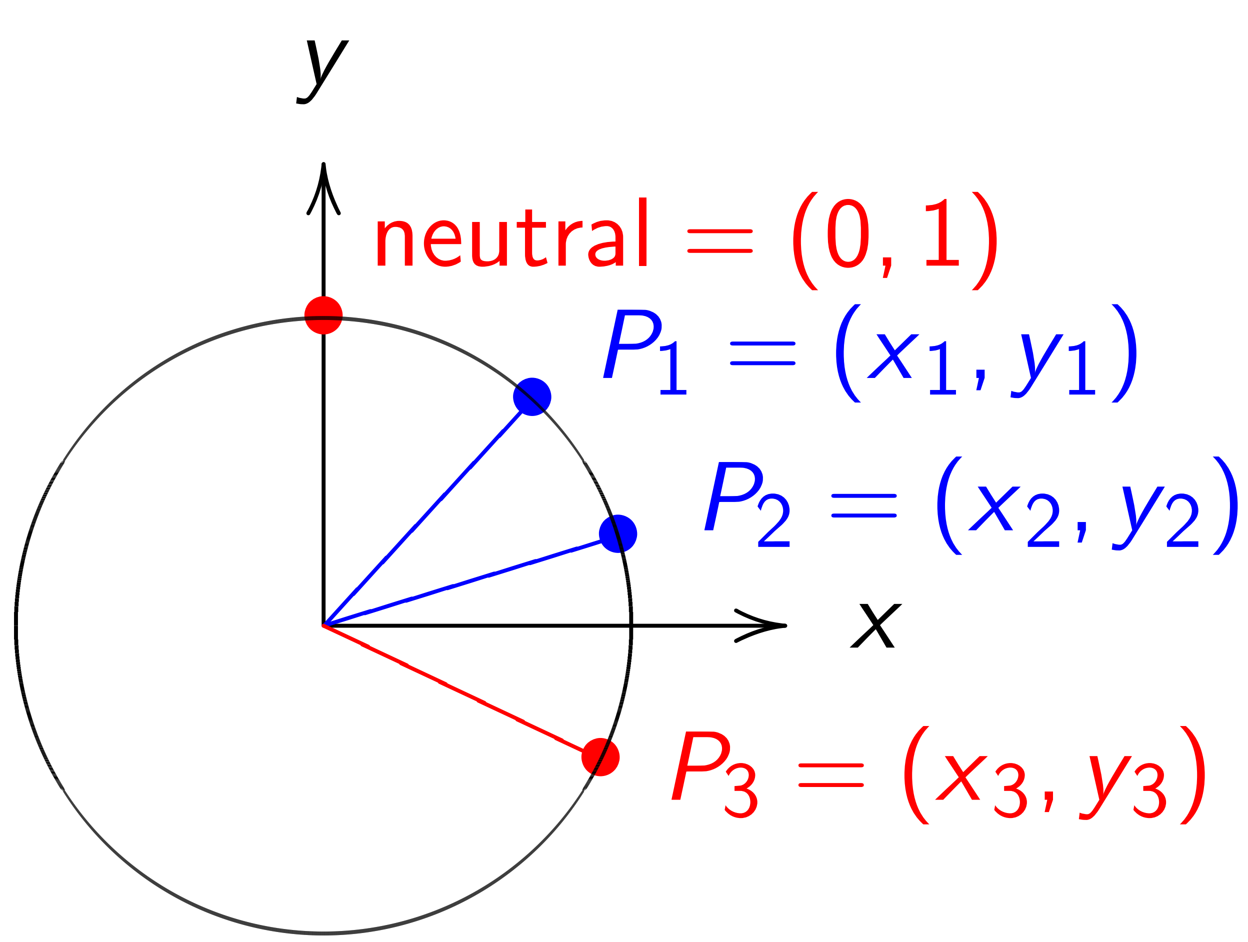
$1 \pm 30x_1x_2y_1y_2 = 1 \neq 0.$

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

then $30x^2y^2 < 1$

so $\sqrt{30}\,|xy| < 1.$

The clock again, for comparison:



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

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

$(x_1 y_2 + y_1 x_2,$

$y_1 y_2 - x_1 x_2).$

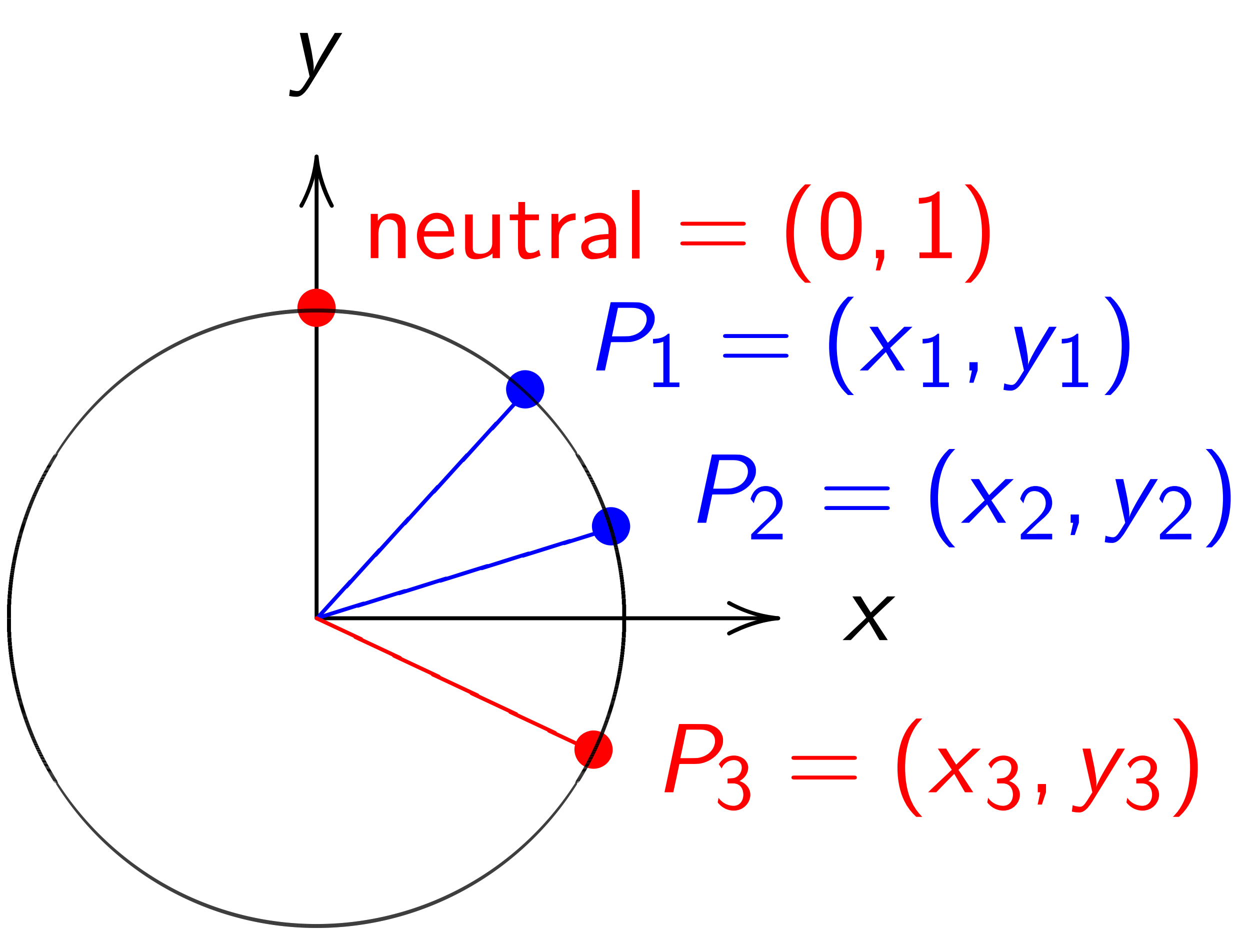
"Hey, there were divisions in the Edwards addition law! What if the denominators are 0?"

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 $\sqrt{30}\,|xy| < 1.$

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 $\sqrt{30}\,|xy| < 1$.

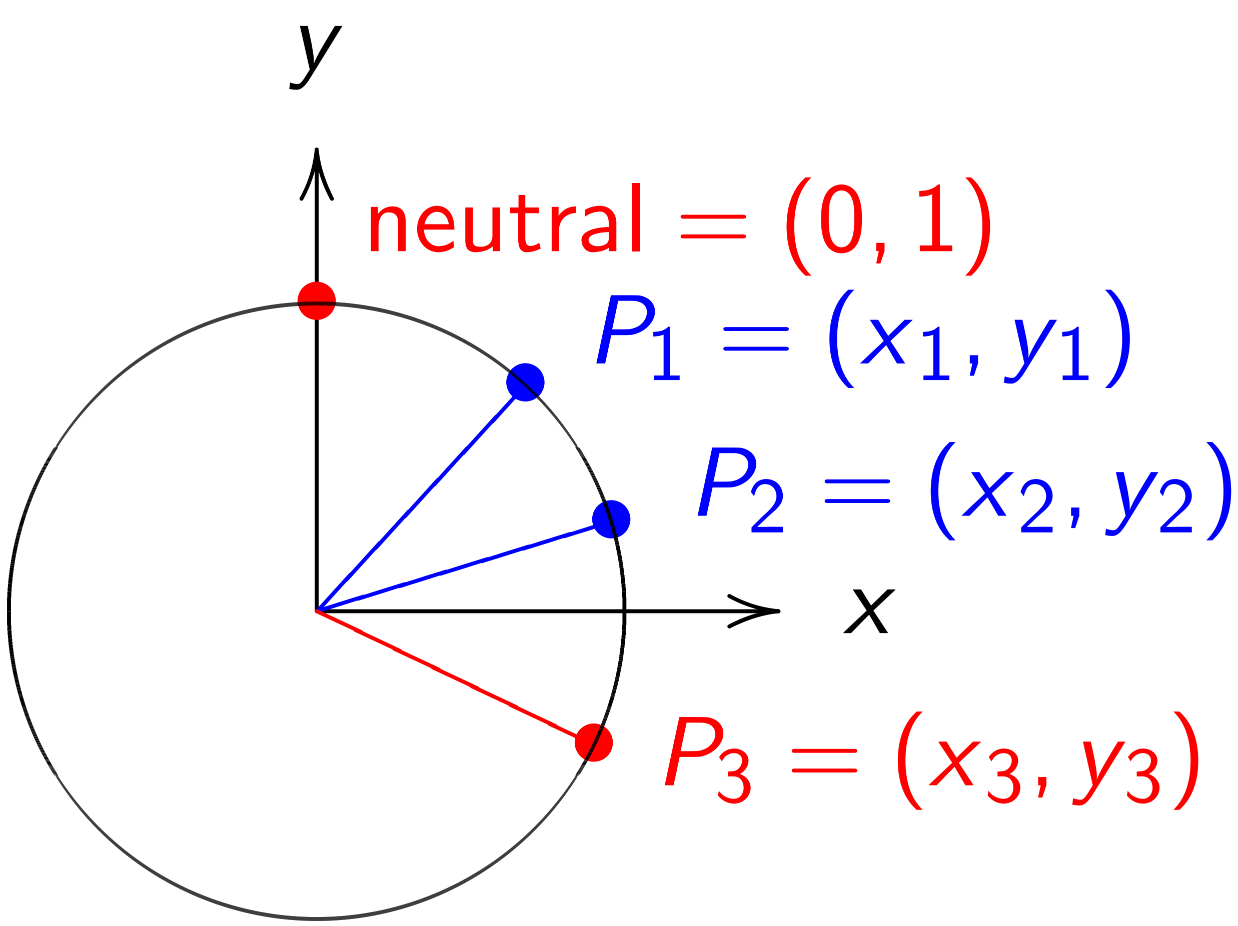
If $x_1^2 + y_1^2 = 1 - 30x_1^2y_1^2$

and $x_2^2 + y_2^2 = 1 - 30x_2^2y_2^2$

then $\sqrt{30}\,|x_1y_1| < 1$

and $\sqrt{30}\,|x_2y_2| < 1$

The clock again, for comparison:



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

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

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

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

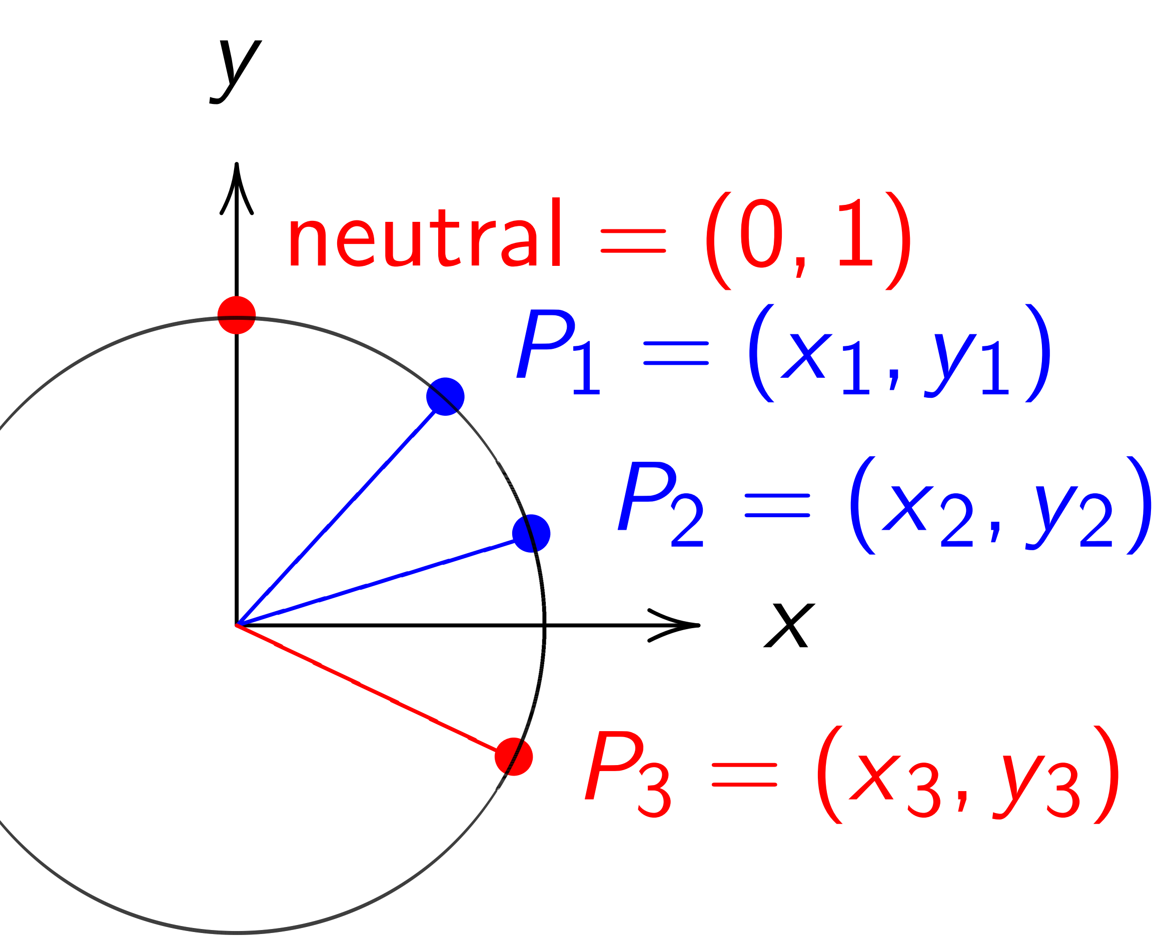
Answer: They aren't!

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

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 $1 \pm 30x_1x_2y_1y_2 > 0.$

ck again, for comparison:



$= 1.$

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

$y_1 x_2,$

$x_1 x_2).$

"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 $\sqrt{30}\ |xy| < 1.$

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 $\sqrt{30}\ |x_1 y_1| < 1$ and $\sqrt{30}\ |x_2 y_2| < 1$ so $30\,|x_1 y_1 x_2 y_2| < 1$ so $1 \pm 30 x_1 x_2 y_1 y_2 > 0.$

The Edw

(x_1, y_1)

$((x_1 y_2 +$

$(y_1 y_2 -$

is a grou

$x^2 + y^2$

Some ca

addition

addition

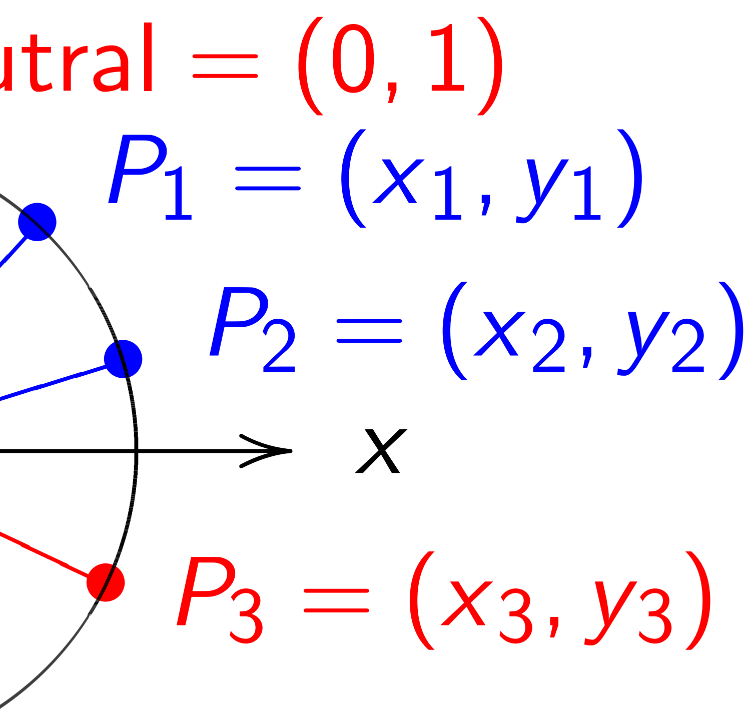
Other pa

addition

$(0, 1)$ is

(x_1, y_1)

or comparison:



nd (x_2, y_2) is

"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 addi

$(x_1, y_1) + (x_2, y_2)$

$((x_1y_2{+}y_1x_2)/(1{+}$

$(y_1y_2{-}x_1x_2)/(1{-}$

is a group law for

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

Some calculation r

addition result is o

addition law is ass

Other parts of pro

addition law is cor

$(0, 1)$ is neutral ele

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

rison:

1)

$x_1, y_1)$

(x_2, y_2)

(x_3, y_3)

) is

"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
$(x_1, y_1) + (x_2, y_2) =$
$((x_1y_2 + y_1x_2)/(1 - 30x_1x_2y$
$(y_1y_2 - x_1x_2)/(1 + 30x_1x_2y$
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 eas
addition law is commutative
$(0, 1)$ is neutral element;
$(x_1, y_1) + (-x_1, y_1) = (0, 1$

“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

$$(x_1, y_1) + (x_2, y_2) = ((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2),\ (y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2))$$

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

here were divisions
dwards addition law!
the denominators are 0?”
They aren’t!
0 or $y_i = 0$ then
${}_1x_2y_1y_2 = 1 \neq 0$.
$y^2 = 1 - 30x^2y^2$
$x^2y^2 < 1$
$|xy| < 1$.

$y_1^2 = 1 - 30x_1^2y_1^2$
$+ y_2^2 = 1 - 30x_2^2y_2^2$
$0\ |x_1y_1| < 1$
$0\ |x_2y_2| < 1$
${}_1y_1x_2y_2| < 1$
$0x_1x_2y_1y_2 > 0$.

The Edwards addition law
$(x_1, y_1) + (x_2, y_2) =$
$((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2),$
$(y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2))$
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;
$(x_1, y_1) + (-x_1, y_1) = (0, 1)$.

<u>Edwards</u>

Choose
Choose
$\{(x, y)$
x^2
is a “co
Roughly

```
def edwa
  x1,y1 =
  x2,y2 =
  x3 = (
  (1+d*x
  y3 = (
  (1-d*x
  return
```

divisions

dition law!

ninators are 0?"

n't!

0 then

$= 1 \neq 0$.

$30x^2y^2$

$30x_1^2y_1^2$

$- 30x_2^2y_2^2$

< 1

1

< 1

$_2 > 0$.

The Edwards addition law

$(x_1, y_1) + (x_2, y_2) =$
$((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2),$
$(y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2))$
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;
$(x_1, y_1) + (-x_1, y_1) = (0, 1)$.

Edwards curves mo

Choose an odd pri
Choose a *non-squa*
$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p$
$x^2 + y^2 = 1$
is a "complete Edw
Roughly $p + 1$ pai

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

!

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

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

Edwards curves mod p

Choose an odd prime p.

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

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

is a “complete Edwards curv

Roughly $p + 1$ pairs (x, y).

```
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
```

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

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

Edwards curves mod p

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

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

is a "complete Edwards curve".
Roughly $p + 1$ pairs (x, y).

```
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
```

wards addition law

$+ (x_2, y_2) =$

$y_1x_2)/(1-30x_1x_2y_1y_2),$

$x_1x_2)/(1+30x_1x_2y_1y_2))$

up law for the curve

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

lculation required:

result is on curve;

law is associative.

arts of proof are easy:

law is commutative;

neutral element;

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

Edwards curves mod p

Choose an odd prime p.

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

$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p :$

$\quad x^2 + y^2 = 1 + dx^2y^2\}$

is a "complete Edwards curve".

Roughly $p + 1$ pairs (x, y).

```
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
```

Denomin

But need

"$x^2 + y$

tion law

$=$

$-30x_1x_2y_1y_2),$

$+30x_1x_2y_1y_2))$

the curve

x^2y^2.

required:

on curve;

ociative.

of are easy:

mmutative;

ement;

$_1) = (0, 1)$.

Edwards curves mod p

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".

Roughly $p + 1$ pairs (x, y).

```
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
```

Denominators are

But need different

"$x^2 + y^2 > 0$" doe

$_1y_2)$,

$_1y_2))$

y:

;

).

Edwards curves mod p

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".

Roughly $p + 1$ pairs (x, y).

```
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
```

Denominators are never 0.

But need different proof;

"$x^2 + y^2 > 0$" doesn't work

Edwards curves mod p

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".

Roughly $p + 1$ pairs (x, y).

```
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
```

Denominators are never 0.

But need different proof;

"$x^2 + y^2 > 0$" doesn't work.

Edwards curves mod p

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".

Roughly $p + 1$ pairs (x, y).

```
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
```

Denominators are never 0.

But need different proof; "$x^2 + y^2 > 0$" doesn't work.

Answer: Can prove that the denominators are never 0.

Addition law is **complete**.

Edwards curves mod p

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".

Roughly $p + 1$ pairs (x, y).

```
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
```

Denominators are never 0.

But need different proof;
"$x^2 + y^2 > 0$" doesn't work.

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

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

Edwards curves mod p

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".
Roughly $p + 1$ pairs (x, y).

```
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
```

Denominators are never 0.
But need different proof;
"$x^2 + y^2 > 0$" doesn't work.

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.

curves mod p

an odd prime p.

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

$\in \mathbf{F}_p \times \mathbf{F}_p$:

$+ y^2 = 1 + dx^2y^2\}$

mplete Edwards curve".

$p + 1$ pairs (x, y).

```
rdsadd(P1,P2):
= P1
= P2
x1*y2+y1*x2)/ \
1*x2*y1*y2)
y1*y2-x1*x2)/ \
1*x2*y1*y2)
 x3,y3
```

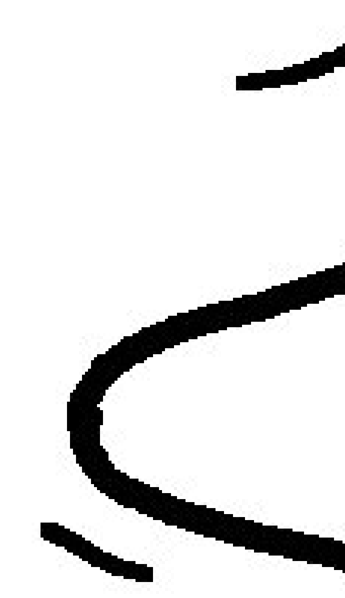
Denominators are never 0.
But need different proof;
"$x^2 + y^2 > 0$" doesn't work.

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.

Edwards



od *p*

me *p*.

are $d \in \mathbf{F}_p$.

:

$+ dx^2y^2\}$

wards curve".

rs (x, y).

```
P2):



2)/ \
)
2)/ \
)
```

Denominators are never 0.
But need different proof;
"$x^2 + y^2 > 0$" doesn't work.

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.

Edwards curves are

ve”.

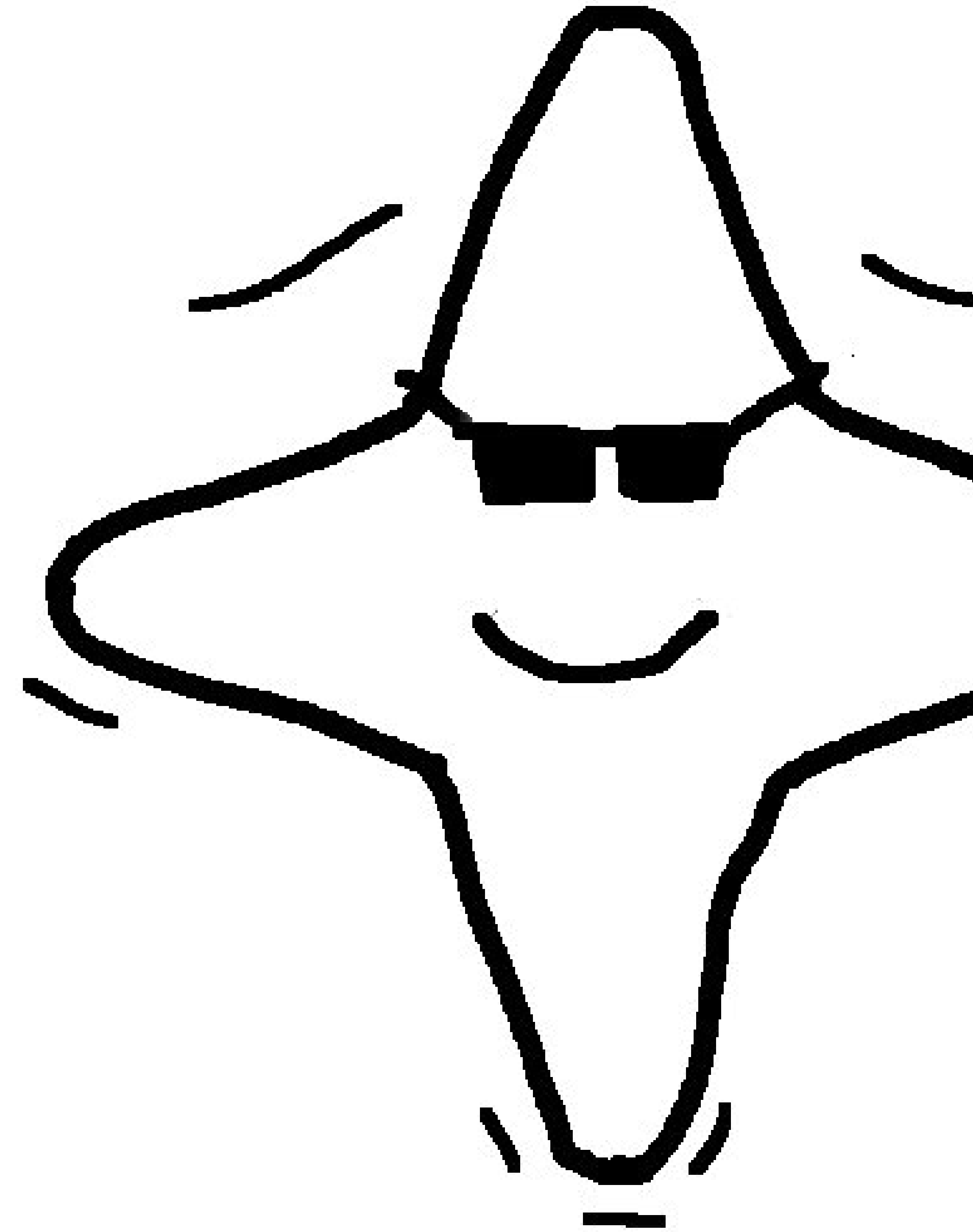
Denominators are never 0.
But need different proof;
“$x^2 + y^2 > 0$” doesn’t work.

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.

Edwards curves are cool



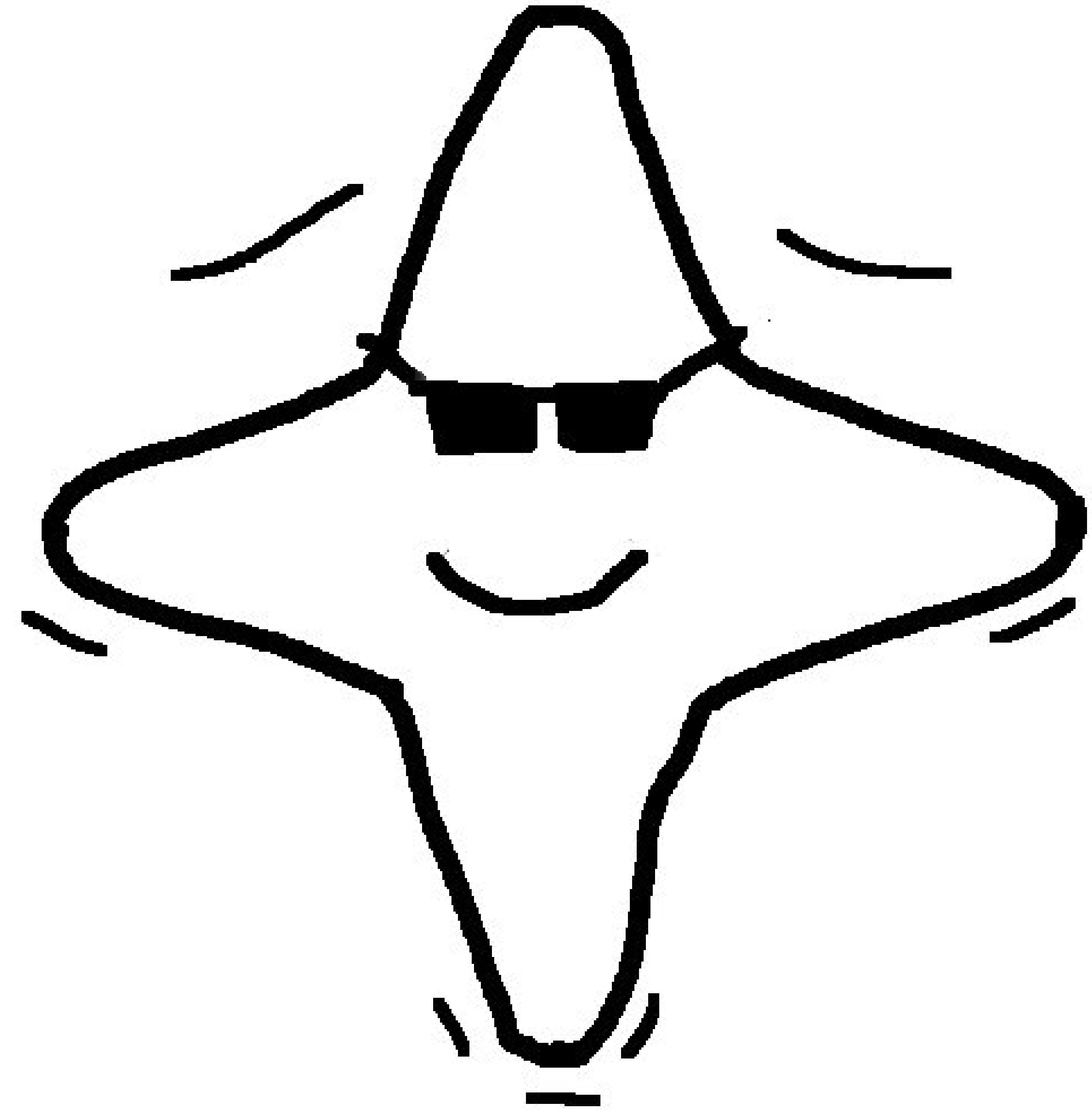
Denominators are never 0.
But need different proof;
"$x^2 + y^2 > 0$" doesn't work.

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.

Edwards curves are cool



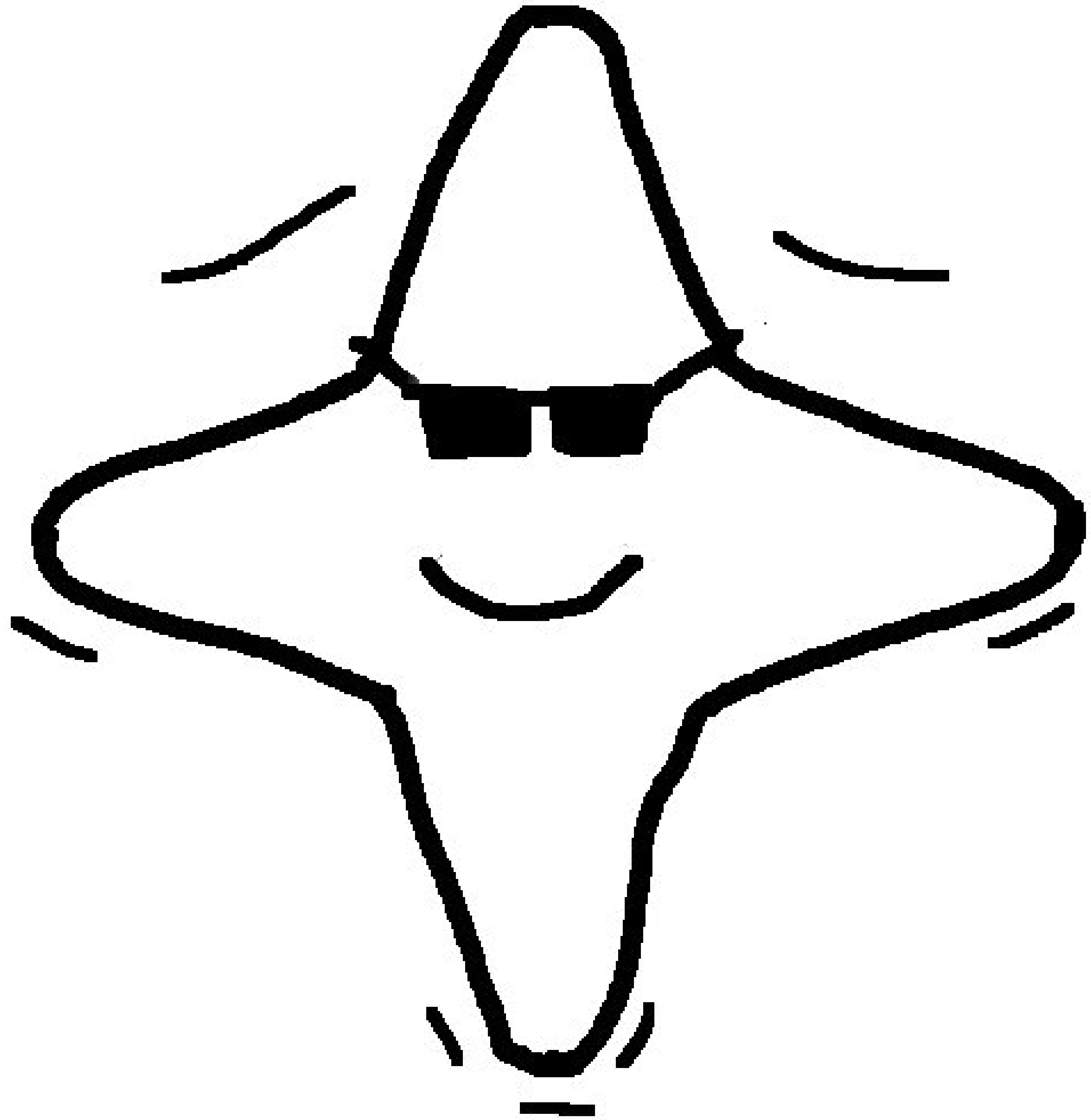
nators are never 0.
d different proof;
$^2 > 0$" doesn't work.

Can prove that
ominators are never 0.
n law is **complete**.

of relies on
g *non-square d*.

stead choose square *d*:
still elliptic, and
seems to work,
e are failure cases,
ploitable by attackers.
de is more complicated.

Edwards curves are cool



ECDSA

Users ca
using Ed

Take a p
curve mo

ECDSA
the *orde*

There ar
points; a
Adding
reach (0
integer >
This ℓ is

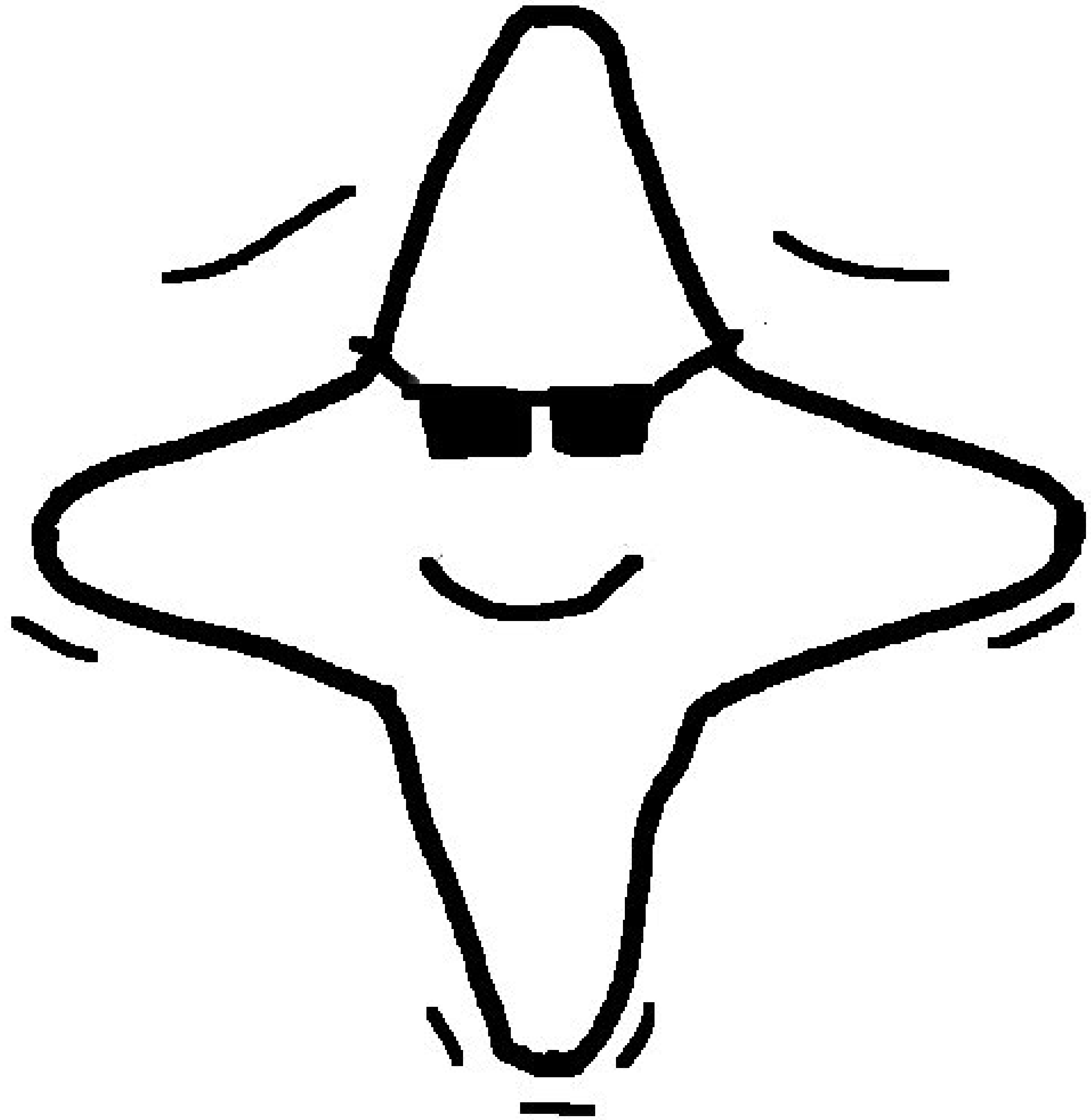
never 0.
proof;
esn't work.

e that
are never 0.
mplete.

n
are d.

se square d:
c, and
work,
re cases,
y attackers.
complicated.

Edwards curves are cool



ECDSA

Users can sign me
using Edwards cur

Take a point P on
curve modulo a pr

ECDSA signer nee
the *order of P*.

There are only fini
points; about p in
Adding P to itself
reach $(0, 1)$; let ℓ
integer > 0 with ℓ
This ℓ is the order

.

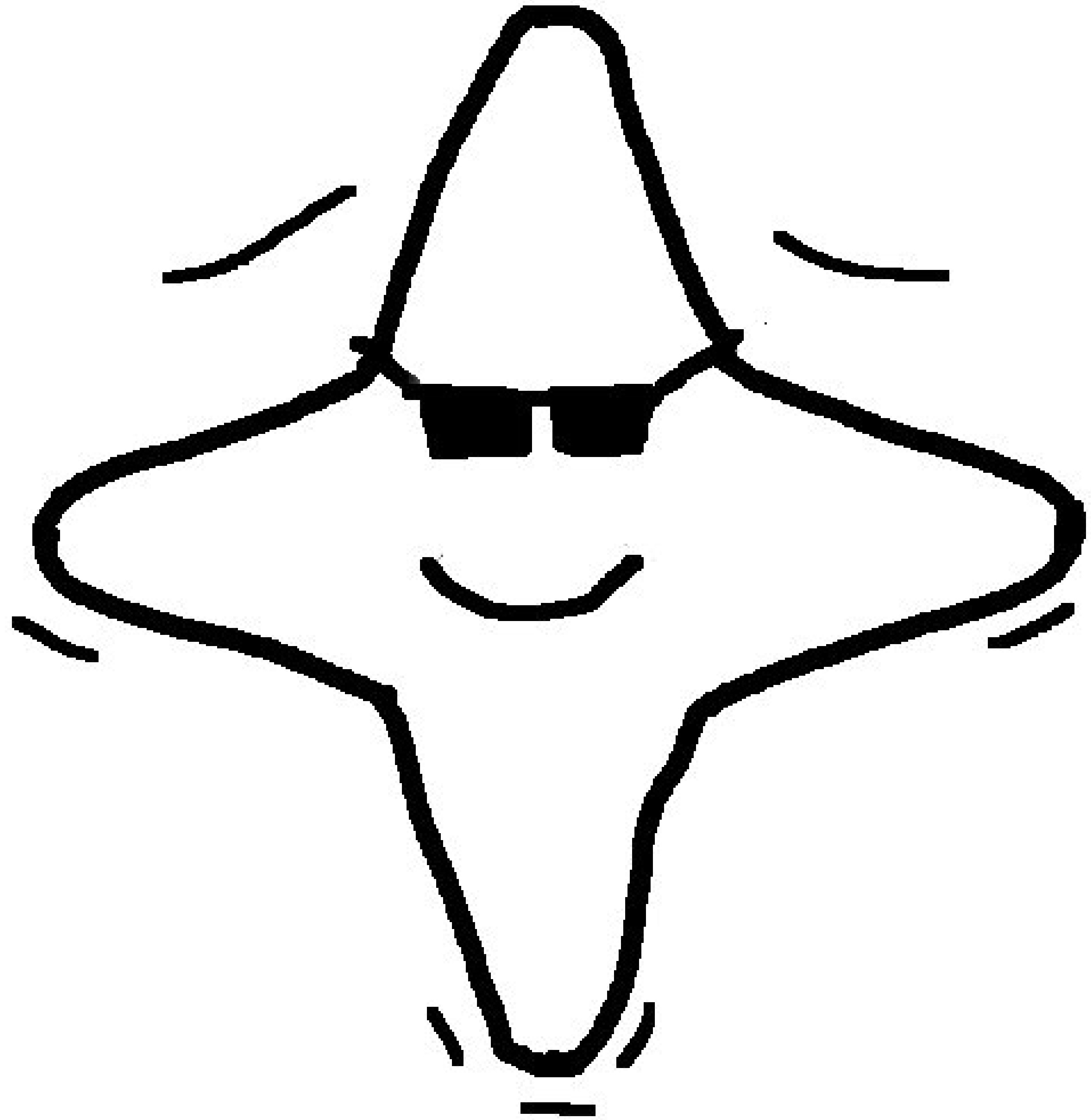
0.

d:

rs.

ed.

Edwards curves are cool



ECDSA

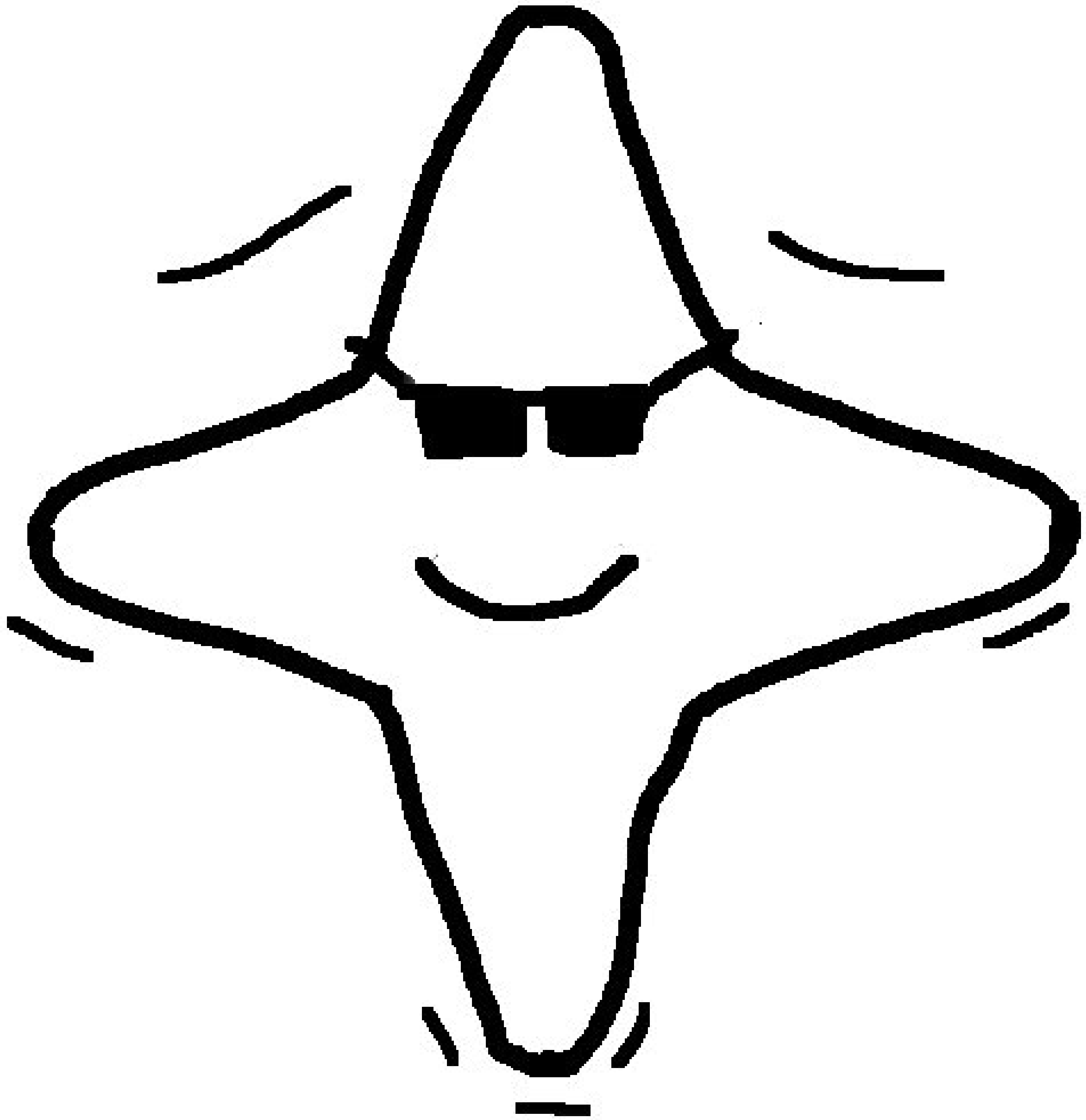
Users can sign messages using Edwards curves.

Take a point P on an Edwar curve modulo a prime $p > 2$

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

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

Edwards curves are cool



ECDSA

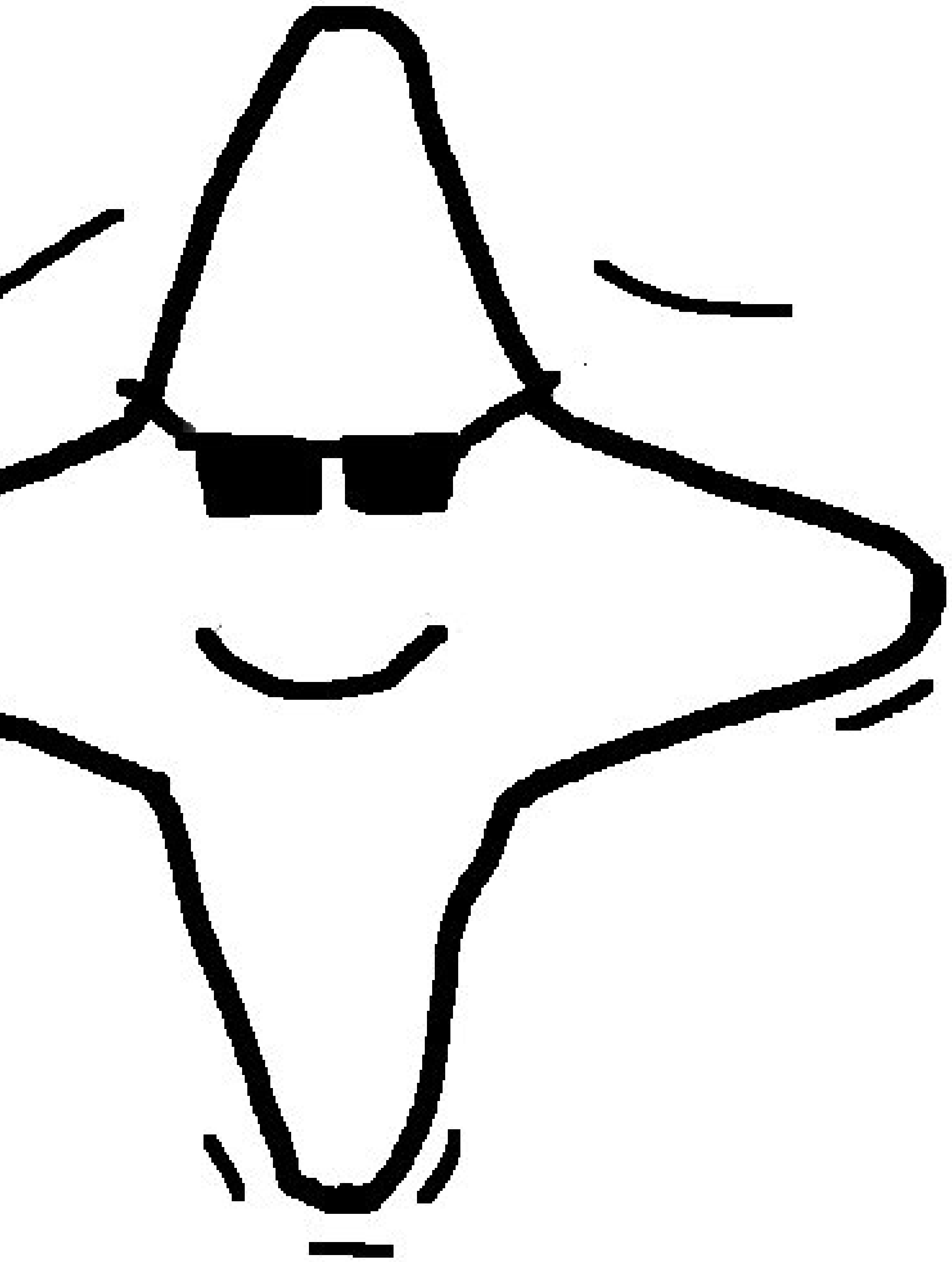
Users can sign messages using Edwards curves.

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

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

There are only finitely many other points; about p 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.

curves are cool



ECDSA

Users can sign messages using Edwards curves.

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

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

There are only finitely many other points; about p 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 sign
system p
base poi
h with c
$\lfloor \log_2 \ell \rfloor$
Alice's s
and her

To sign
Alice co
picks ra
compute
puts $r \equiv$
$s \equiv k^{-1}$

The sign

e cool

ECDSA

Users can sign messages using Edwards curves.

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

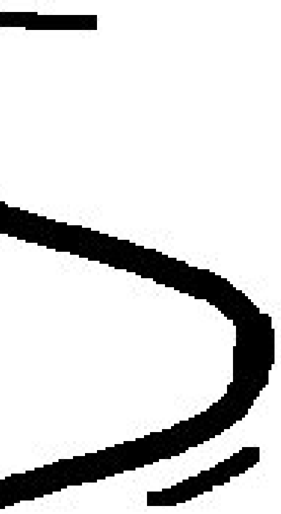
ECDSA signer needs to know the *order of* P.

There are only finitely many other points; about p 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 sche
system parameters
base point P; and
h with output leng
$\lfloor \log_2 \ell \rfloor + 1$.
Alice's secret key i
and her public key

To sign message m
Alice computes $h($
picks random k;
computes $R = kP$
puts $r \equiv y_1$ mod ℓ
$s \equiv k^{-1}(h(m) + r$

The signature on



ECDSA

Users can sign messages using Edwards curves.

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

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

There are only finitely many other points; about p 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
base point P; and a hash fu
h with output length at leas
$\lfloor \log_2 \ell \rfloor + 1$.
Alice's secret key is an integ
and her public key is $P_A = a$

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

The signature on m is (r, s)

ECDSA

Users can sign messages using Edwards curves.

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

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

There are only finitely many other points; about p 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).

n sign messages
dwards curves.

point P on an Edwards
odulo a prime $p > 2$.

signer needs to know
r of P.

re only finitely many other
about p in total.
P to itself will eventually
$, 1)$; let ℓ be the smallest
> 0 with $\ell P = (0, 1)$.
s 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
given m
Comput
and $w_2 \equiv$
Check w
of $w_1 P$
and if so

Alice's s
$w_1 P + w$
$(s^{-}$
$(s^{-}$
and so t
expressio
the y-co

ssages
ves.

n an Edwards
ime $p > 2$.

eds to know

itely many other
total.
will eventually
be the smallest
$P = (0, 1)$.
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 verif
given m and (r, s)
Compute $w_1 \equiv s^{-}$
and $w_2 \equiv s^{-1} \cdot r$
Check whether the
of $w_1 P + w_2 P_A$ eq
and if so, accept s

Alice's signatures
$w_1 P + w_2 P_A =$
$(s^{-1} h(m))P$
$(s^{-1}(h(m) +$
and so the y-coord
expression equals
the y-coordinate o

rds

ℓ.

w

other

tually

allest

).

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 signatur
given m and (r, s):
Compute $w_1 \equiv s^{-1}h(m)$ m
and $w_2 \equiv s^{-1} \cdot r \bmod \ell$.
Check whether the y-coordi
of $w_1P + w_2P_A$ equals r mo
and if so, accept signature.

Alice's signatures are valid:
$w_1P + w_2P_A =$
$(s^{-1}h(m))P + (s^{-1} \cdot r$
$(s^{-1}(h(m) + ra))P =$
and so the y-coordinate of t
expression equals r,
the y-coordinate of kP.

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 $w_1P + w_2P_A$ equals r modulo ℓ and if so, accept signature.

Alice's signatures are valid:

$$
\begin{aligned}
w_1P + w_2P_A &= \\
&(s^{-1}h(m))P + (s^{-1} \cdot r)P_A = \\
&(s^{-1}(h(m) + ra))P = kP
\end{aligned}
$$

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

nature scheme has as
parameters a curve E; a
nt P; and a hash function
output length at least
$+ 1$.
secret key is an integer a
public key is $P_A = aP$.

message m,
mputes $h(m)$;
ndom k;
es $R = kP = (x_1, y_1)$;
$\equiv y_1 \bmod \ell$; computes
$(h(m) + r \cdot a) \bmod \ell$.

nature 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 $w_1P + w_2P_A$ equals r modulo ℓ and if so, accept signature.

Alice's signatures are valid:

$$w_1P + w_2P_A = (s^{-1}h(m))P + (s^{-1} \cdot r)P_A = (s^{-1}(h(m) + ra))P = kP$$

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

Attacker

Anybody
Alice's p
$s \equiv k^{-1}$

Can fake
break th
compute
Most of
methods
Sometim

eme has as
s a curve E; a
a hash function
gth at least

is an integer a
is $P_A = aP$.

m,
(m);

$= (x_1, y_1)$;
ℓ; computes
$r \cdot a)$ mod ℓ.

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 $w_1P + w_2P_A$ equals r modulo ℓ and if so, accept signature.

Alice's signatures are valid:

$$w_1P + w_2P_A = (s^{-1}h(m))P + (s^{-1} \cdot r)P_A = (s^{-1}(h(m) + ra))P = kP$$

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

Attacker's view on

Anybody can prod
Alice's private key
$s \equiv k^{-1}(h(m) +$

Can fake signature
break the DLP, i.e
compute a from P
Most of this cours
methods for break
Sometimes attacks

s
E; a
nction
st

ger a
aP.

);
es
ℓ.
.

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:

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

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

Attacker's view on signature

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

Can fake signatures if one ca
break the DLP, i.e., if one ca
compute a from P_A.
Most of this course deals wi
methods for breaking DLPs.
Sometimes attacks are easie

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 $w_1P + w_2P_A$ equals r modulo ℓ and if so, accept signature.

Alice's signatures are valid:

$$w_1P + w_2P_A = (s^{-1}h(m))P + (s^{-1} \cdot r)P_A = (s^{-1}(h(m) + ra))P = kP$$

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.
Most of this course deals with methods for breaking DLPs.
Sometimes attacks are easier...

y can verify signature
and (r, s):
e $w_1 \equiv s^{-1}h(m) \bmod \ell$
$\equiv s^{-1} \cdot r \bmod \ell$.
whether the y-coordinate
$+ w_2 P_A$ equals r modulo ℓ
o, accept signature.

ignatures are valid:
$w_2 P_A =$
$^1 h(m))P + (s^{-1} \cdot r)P_A =$
$^1(h(m) + ra))P = kP$
he y-coordinate of this
on equals r,
oordinate 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.
Most of this course deals with
methods for breaking DLPs.
Sometimes attacks are easier...

If k is kn
then $a \equiv$

If two si
$m_2, (r, s$
for r: as
$s_1 - s_2$
$(h(m_2)$
$(s_1 - s_2$
Continue

If bits of
(biased
$s \equiv k^-$
as hidde
using lat

y signature
):
$^{-1}h(m) \bmod \ell$
mod ℓ.
e y-coordinate
quals r modulo ℓ
ignature.

are valid:

$+ (s^{-1} \cdot r)P_A =$
$ra))P = kP$
dinate of this
r,
f 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.
Most of this course deals with
methods for breaking DLPs.
Sometimes attacks are easier. . .

If k is known for s
then $a \equiv (sk - h($

If two signatures r
$m_2, (r, s_2)$ have th
for r: assume k_1
$s_1 - s_2 = k_1^{-1}(h$
$(h(m_2) + ra))$; co
$(s_1 - s_2)/(h(m_1)$
Continue as above

If bits of many k's
(biased PRNG) ca
$s \equiv k^{-1}(h(m) +$
as hidden number
using lattice basis

e

od ℓ

nate

odulo ℓ

$)P_A =$

kP

this

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.

Most of this course deals with methods for breaking DLPs.

Sometimes attacks are easier...

If k is known for some $m, ($

then $a \equiv (sk - h(m))/r$ mo

If two signatures $m_1, (r, s_1)$

$m_2, (r, s_2)$ have the same va

for r: assume $k_1 = k_2$; obs

$s_1 - s_2 = k_1^{-1}(h(m_1) + r$

$(h(m_2) + ra))$; compute k

$(s_1 - s_2)/(h(m_1) - h(m_2)$

Continue as above.

If bits of many k's are know

(biased PRNG) can attack

$s \equiv k^{-1}(h(m) + r \cdot a)$ mo

as hidden number problem

using lattice basis reduction.

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.

Most of this course deals with methods for breaking DLPs.

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.

r's view on signatures

y can produce an $R = kP$.
private key is only used in
$(h(m) + r \cdot a)$ mod ℓ.

e signatures if one can
e DLP, i.e., if one can
e a from P_A.
this course deals with
s for breaking DLPs.
nes 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.

Maliciou

Alice car
that two
share the
i.e., she
signed m
$R = (x_1$
have the
Thus, $(r$
$s \equiv k^{-1}$
$-R = ($
$s \equiv -k$
$a \equiv -(h$

signatures

duce an $R = kP$.

is only used in

$r \cdot a)$ mod ℓ.

es if one can

., if one can

P_A.

e deals with

ing DLPs.

s 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 h

that two messages

share the same sig

i.e., she can claim

signed m_1 or m_2

$R = (x_1, y_1)$ and

have the same y-c

Thus, (r, s) fits R

$s \equiv k^{-1}(h(m_1) +$

$-R = (-k)P$,

$s \equiv -k^{-1}(h(m_2)$

$a \equiv -(h(m_1)+h($

es

$= kP$.

ed in

ℓ.

an

an

th

r. . .

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 k

that two messages of her ch

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 $-R = (-x$

have the same y-coordinate.

Thus, (r, s) fits $R = kP$,

$s \equiv k^{-1}(h(m_1) + ra) \bmod$

$-R = (-k)P$,

$s \equiv -k^{-1}(h(m_2) + ra) \bmod$

$a \equiv -(h(m_1)+h(m_2))/2r$

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 $a \equiv -(h(m_1)+h(m_2))/2r \bmod \ell$.

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 $a \equiv -(h(m_1)+h(m_2))/2r \bmod \ell$.

(Easy tweak: include bit of x_1.)

nown for some $m, (r, s)$
$\equiv (sk - h(m))/r \bmod \ell$.

gnatures $m_1, (r, s_1)$ and
$s_2)$ have the same value
ssume $k_1 = k_2$; observe
$= k_1^{-1}(h(m_1) + ra -$
$+ ra))$; compute $k =$
$_2)/(h(m_1) - h(m_2))$.
e as above.

f many k's are known
PRNG) can attack
$^1(h(m) + r \cdot a) \bmod \ell$
n number problem
ttice 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
$a \equiv -(h(m_1)+h(m_2))/2r \bmod \ell$.

(Easy tweak: include bit of x_1.)

More ell

Edwards
Easiest v
elliptic c

Geometr
are Edwa

Algebrai
more elli
(not alw

Every oc
expresse
$v^2 = u^3$

Warning
different

ome $m, (r, s)$
$(m))/r \bmod \ell$.

$n_1, (r, s_1)$ and
he same value
$= k_2$; observe
$(m_1) + ra -$
ompute $k =$
$) - h(m_2))$.
e.

s are known
n attack
$r \cdot a) \bmod \ell$
problem
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
$a \equiv -(h(m_1)+h(m_2))/2r \bmod \ell$.

(Easy tweak: include bit of x_1.)

More elliptic curve

Edwards curves ar
Easiest way to und
elliptic curves is E

Geometrically, all
are Edwards curve

Algebraically,
more elliptic curve
(not always point

Every odd-char cu
expressed as Weier
$v^2 = u^3 + a_2u^2 +$

Warning: "Weiers
different meaning

$r, s)$
od ℓ.

and
alue
erve
$a -$
$=$
$))$.

n

od ℓ

.

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.)

More elliptic curves

Edwards curves are elliptic.
Easiest way to understand elliptic curves is Edwards.

Geometrically, all elliptic cur
are Edwards curves.

Algebraically,
more elliptic curves exist
(not always point of order 4

Every odd-char curve can be
expressed as Weierstrass cur
$v^2 = u^3 + a_2u^2 + a_4u + a_6$

Warning: "Weierstrass" has
different meaning in char 2.

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.)

More elliptic curves

Edwards curves are elliptic.
Easiest way to understand elliptic curves is Edwards.

Geometrically, all elliptic curves are Edwards curves.

Algebraically, more elliptic curves exist (not always point of order 4).

Every odd-char curve can be expressed as Weierstrass curve $v^2 = u^3 + a_2u^2 + a_4u + a_6$.

Warning: "Weierstrass" has different meaning in char 2.

us signer

n set up her public key so
messages of her choice
e same signature,
can claim to have
n_1 or m_2 at will:
$, y_1)$ and $-R = (-x_1, y_1)$
e same y-coordinate.
$r, s)$ fits $R = kP$,
$(h(m_1) + ra) \bmod \ell$ and
$-k)P$,
$^{-1}(h(m_2) + ra) \bmod \ell$ if
$h(m_1)+h(m_2))/2r \bmod \ell$.

veak: include bit of x_1.)

More elliptic curves

Edwards curves are elliptic.
Easiest way to understand
elliptic curves is Edwards.

Geometrically, all elliptic curves
are Edwards curves.

Algebraically,
more elliptic curves exist
(not always point of order 4).

Every odd-char curve can be
expressed as Weierstrass curve
$v^2 = u^3 + a_2u^2 + a_4u + a_6$.

Warning: "Weierstrass" has
different meaning in char 2.

Addition

$v^2 = u^3$

Slope λ
Note tha

er public key so
s of her choice
gnature,
to have
at will:
$-R = (-x_1, y_1)$
coordinate.
$= kP$,
$ra)$ mod ℓ and

$+ ra)$ mod ℓ if
$m_2))/2r$ mod ℓ.

ude bit of x_1.)

More elliptic curves

Edwards curves are elliptic.
Easiest way to understand
elliptic curves is Edwards.

Geometrically, all elliptic curves
are Edwards curves.

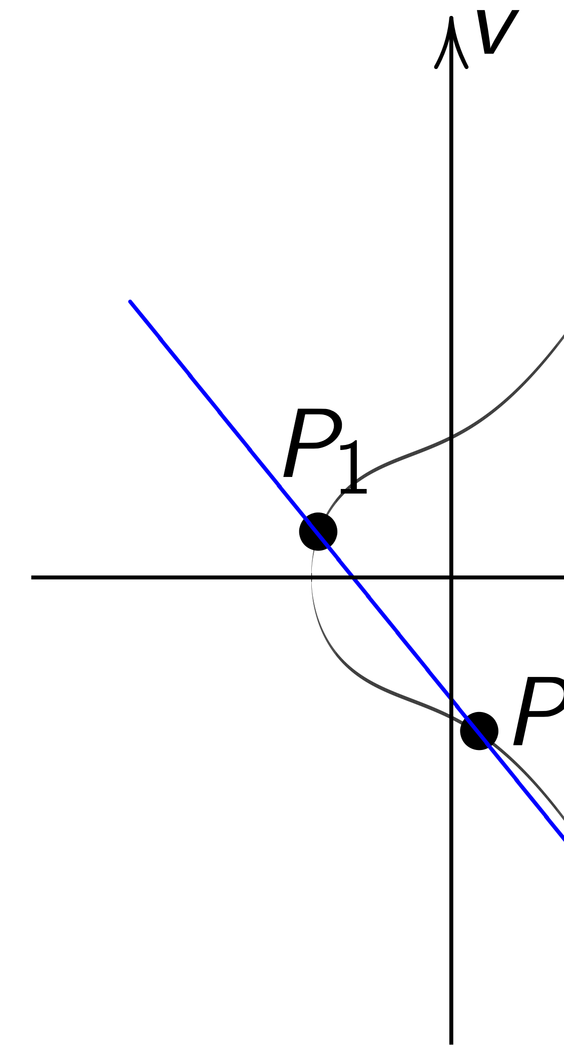
Algebraically,
more elliptic curves exist
(not always point of order 4).

Every odd-char curve can be
expressed as Weierstrass curve
$v^2 = u^3 + a_2u^2 + a_4u + a_6$.

Warning: "Weierstrass" has
different meaning in char 2.

Addition on Weier

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



Slope $\lambda = (v_2 - v$
Note that $u_1 \neq u_2$

key so
oice

$x_1, y_1)$

ℓ and

od ℓ if
mod ℓ.

x_1.)

More elliptic curves

Edwards curves are elliptic.
Easiest way to understand elliptic curves is Edwards.

Geometrically, all elliptic curves are Edwards curves.

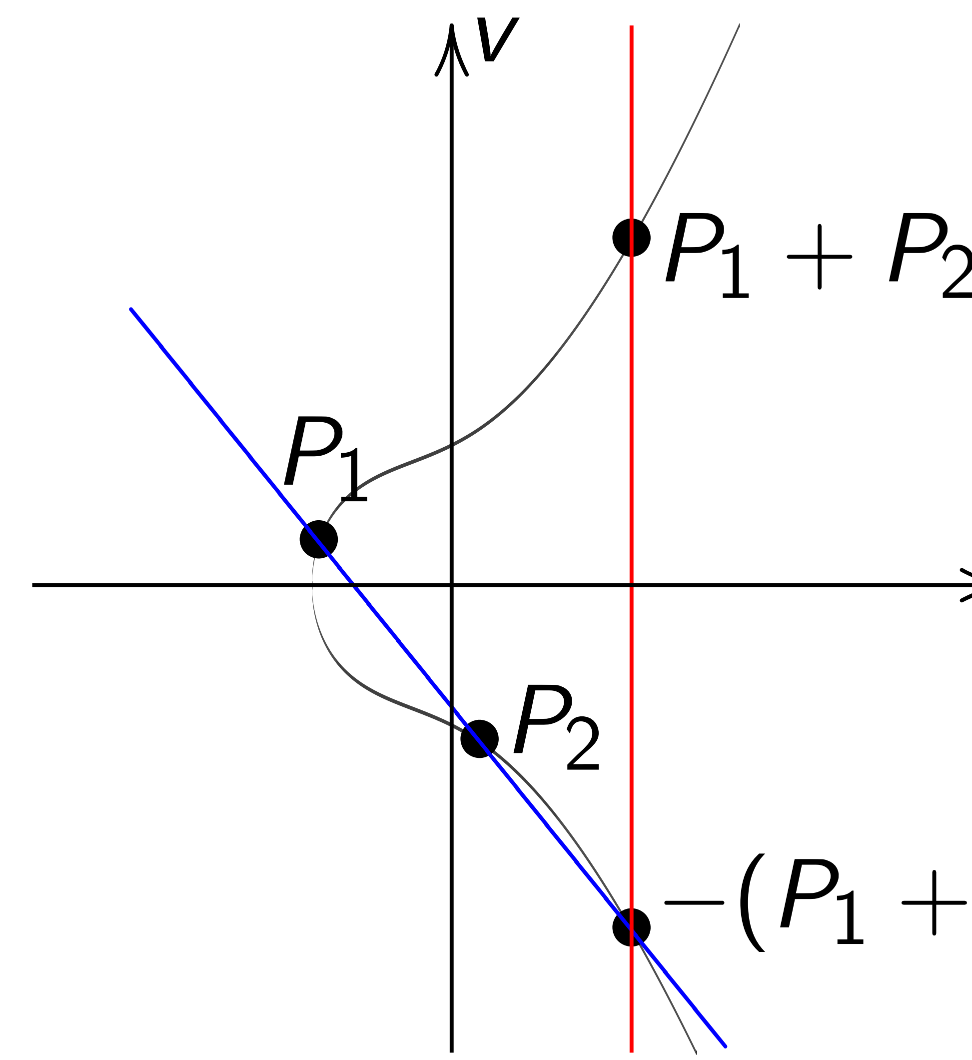
Algebraically, more elliptic curves exist (not always point of order 4).

Every odd-char curve can be expressed as Weierstrass curve $v^2 = u^3 + a_2 u^2 + a_4 u + a_6$.

Warning: "Weierstrass" has different meaning in char 2.

Addition on Weierstrass curv

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



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

More elliptic curves

Edwards curves are elliptic.
Easiest way to understand elliptic curves is Edwards.

Geometrically, all elliptic curves are Edwards curves.

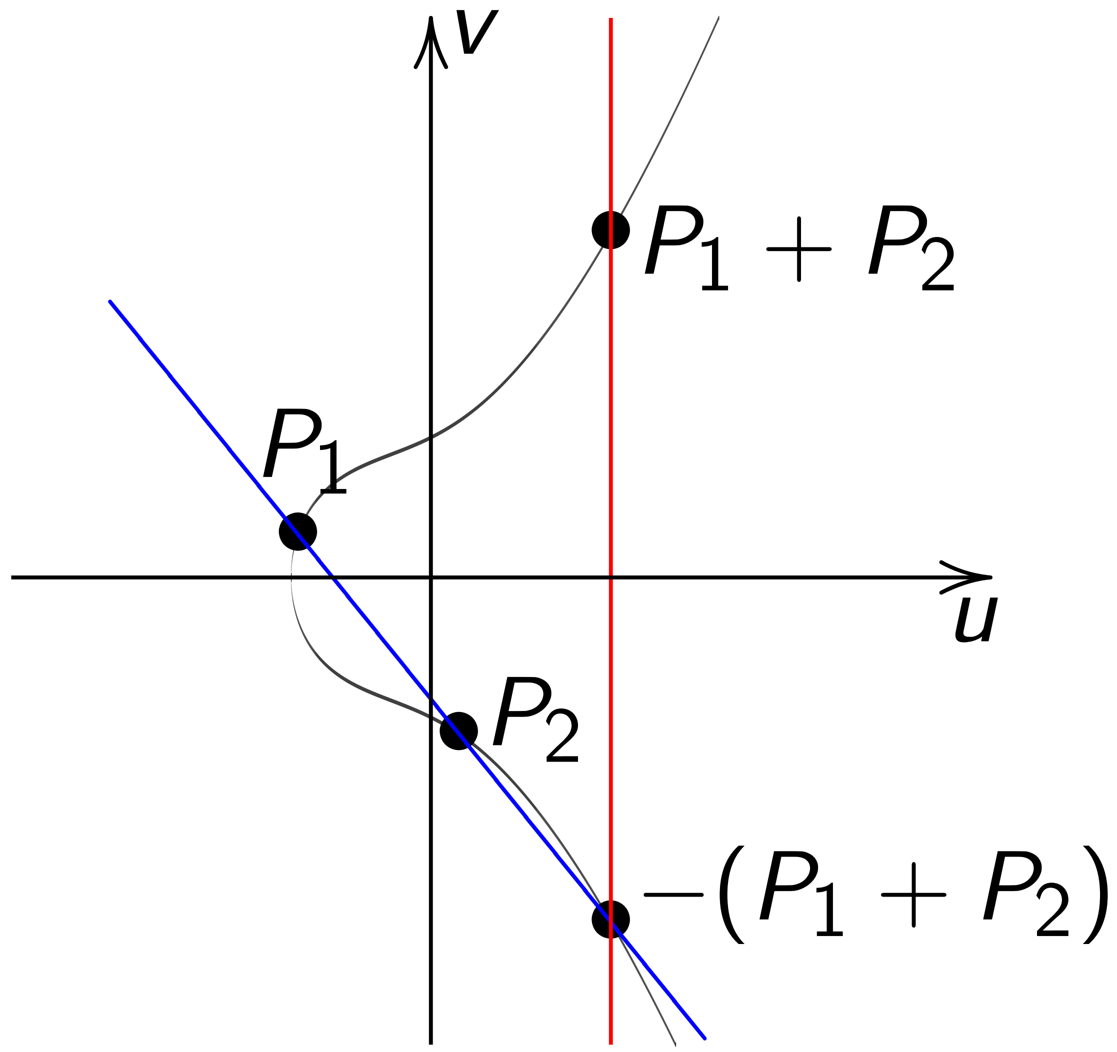
Algebraically,
more elliptic curves exist
(not always point of order 4).

Every odd-char curve can be expressed as Weierstrass curve $v^2 = u^3 + a_2 u^2 + a_4 u + a_6$.

Warning: "Weierstrass" has different meaning in char 2.

Addition on Weierstrass curve

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



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

iptic curves

curves are elliptic.
way to understand
curves is Edwards.

rically, all elliptic curves
ards curves.

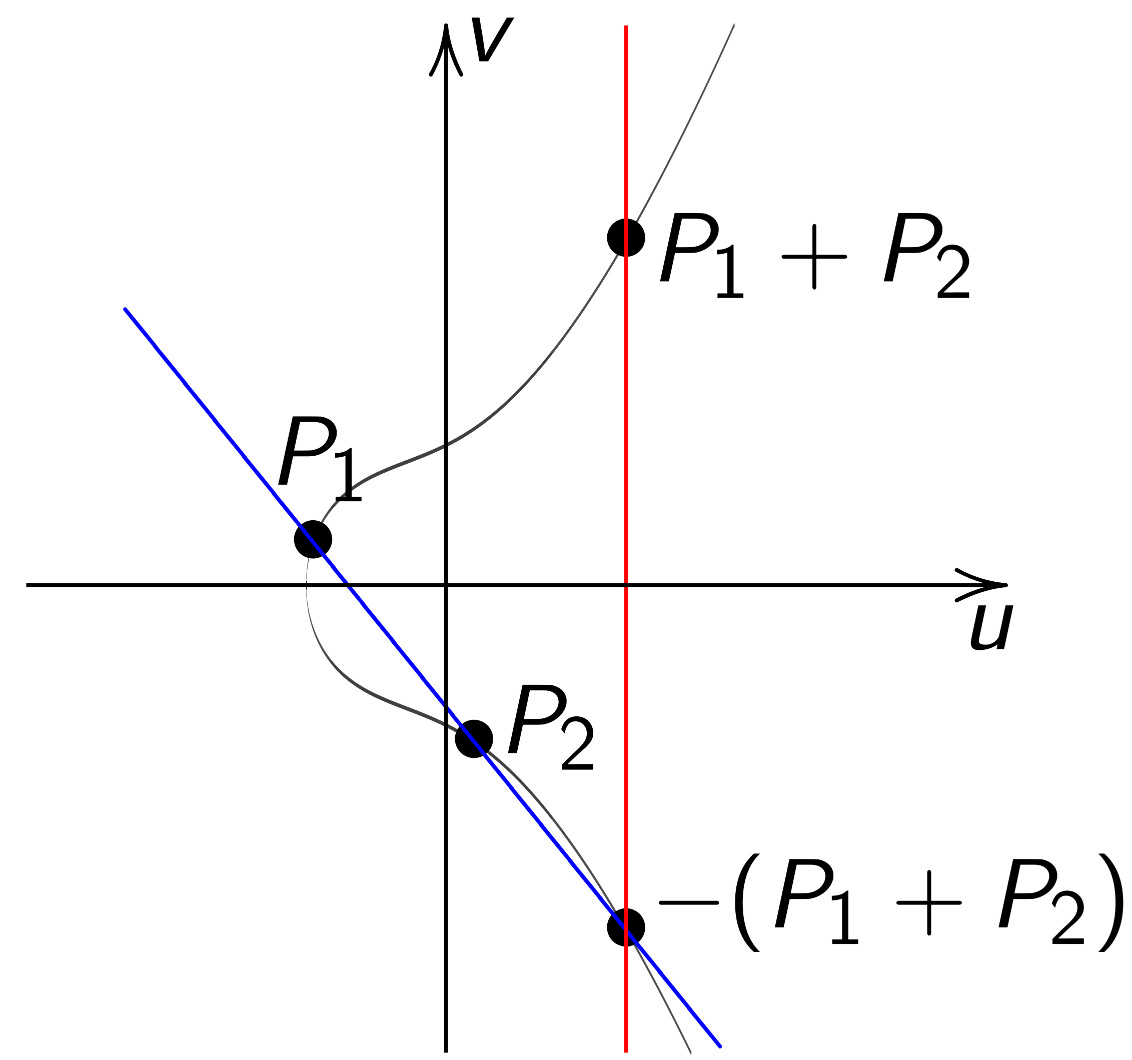
cally,
iptic curves exist
ays point of order 4).

dd-char curve can be
d as Weierstrass curve
$+ a_2 u^2 + a_4 u + a_6$.

: "Weierstrass" has
meaning in char 2.

Addition on Weierstrass curve

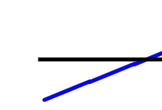
$v^2 = u^3 + u^2 + u + 1$



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

Doubling

$v^2 = u^3$



Slope λ

es

e elliptic.
derstand
dwards.

elliptic curves
s.

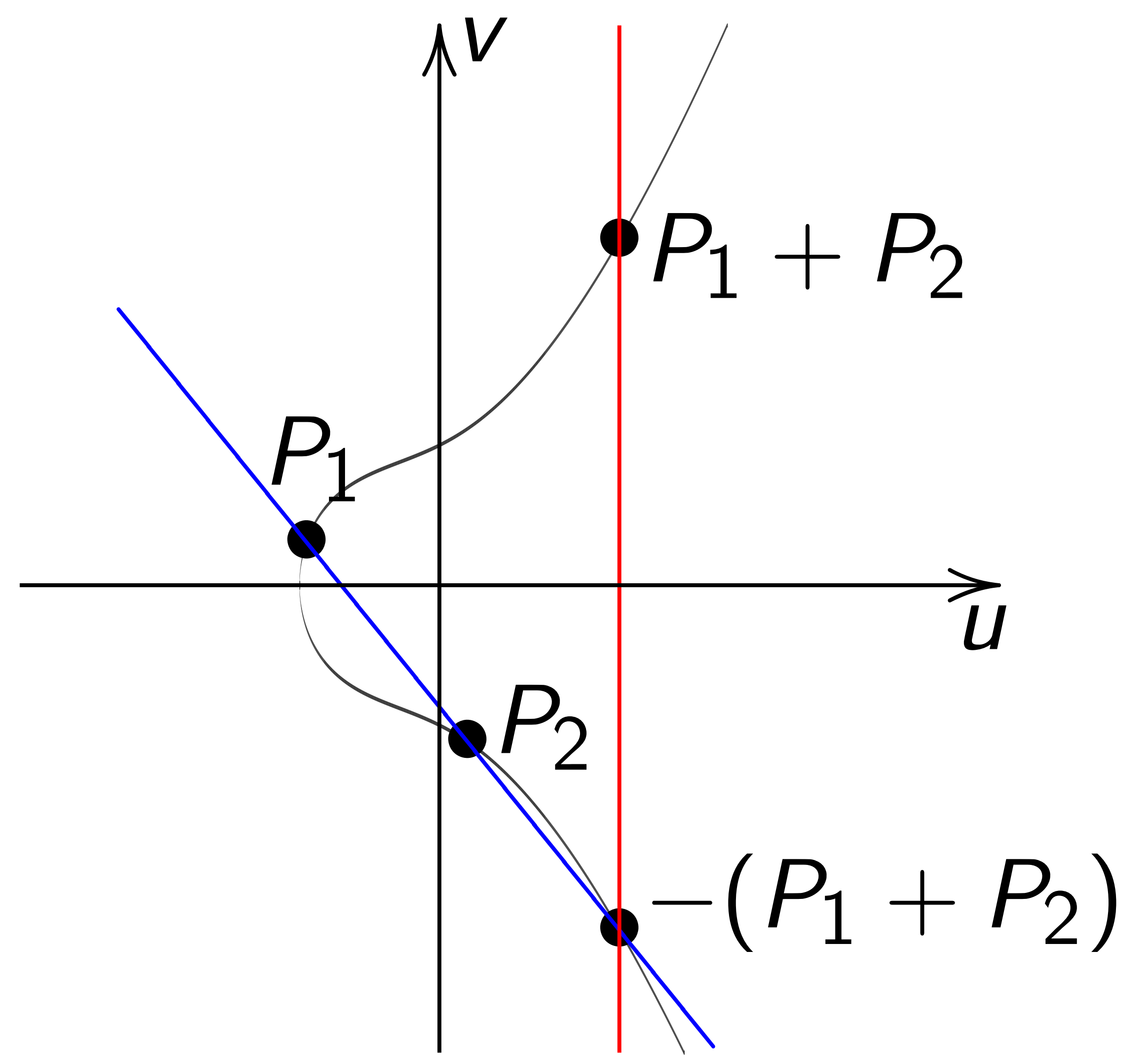
es exist
of order 4).

rve can be
rstrass curve
$- a_4 u + a_6$.

trass" has
in char 2.

Addition on Weierstrass curve

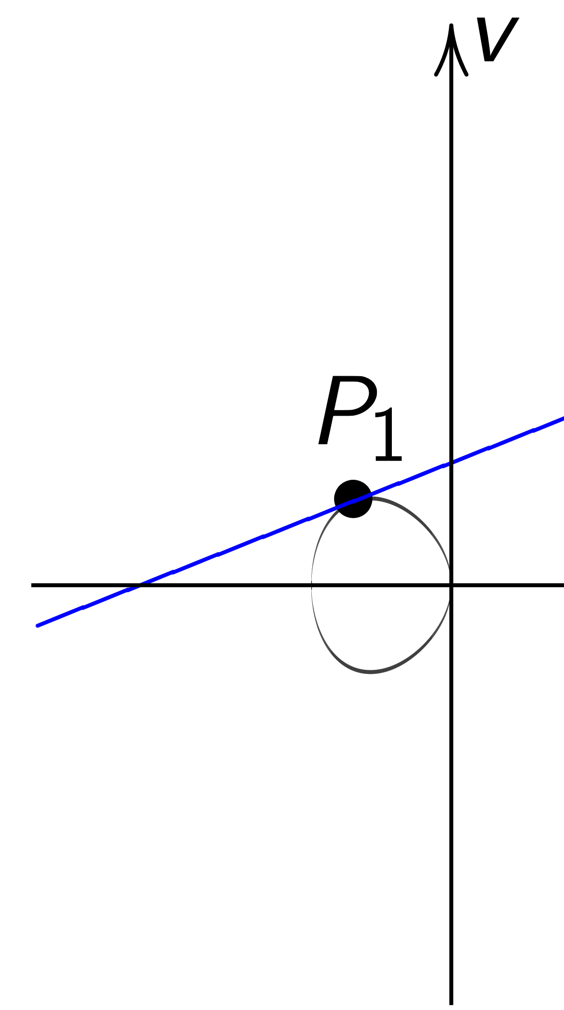
$v^2 = u^3 + u^2 + u + 1$



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

Doubling on Weier

$v^2 = u^3 - u$



Slope $\lambda = (3u_1^2 -$

rves

).

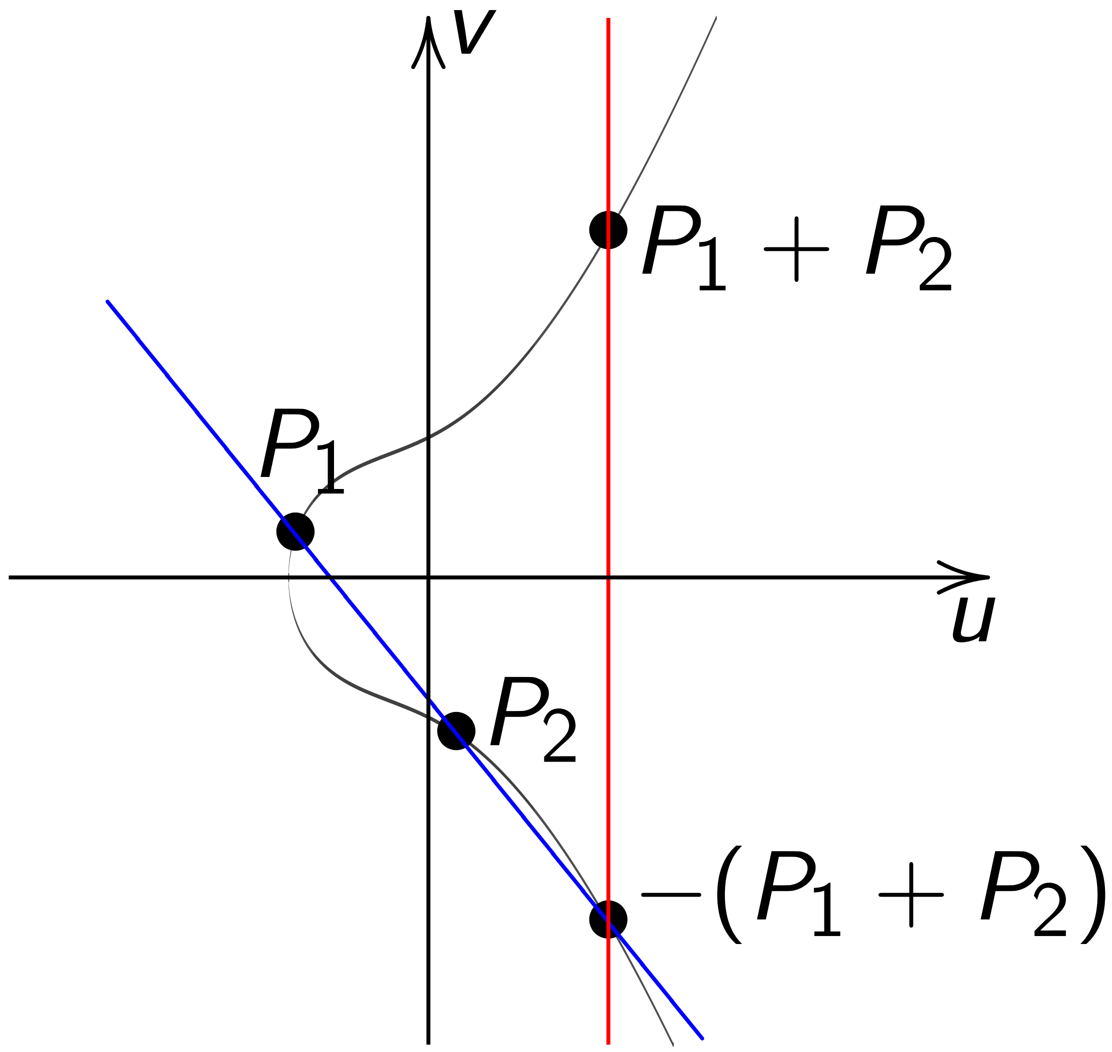
e

rve

.

Addition on Weierstrass curve

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

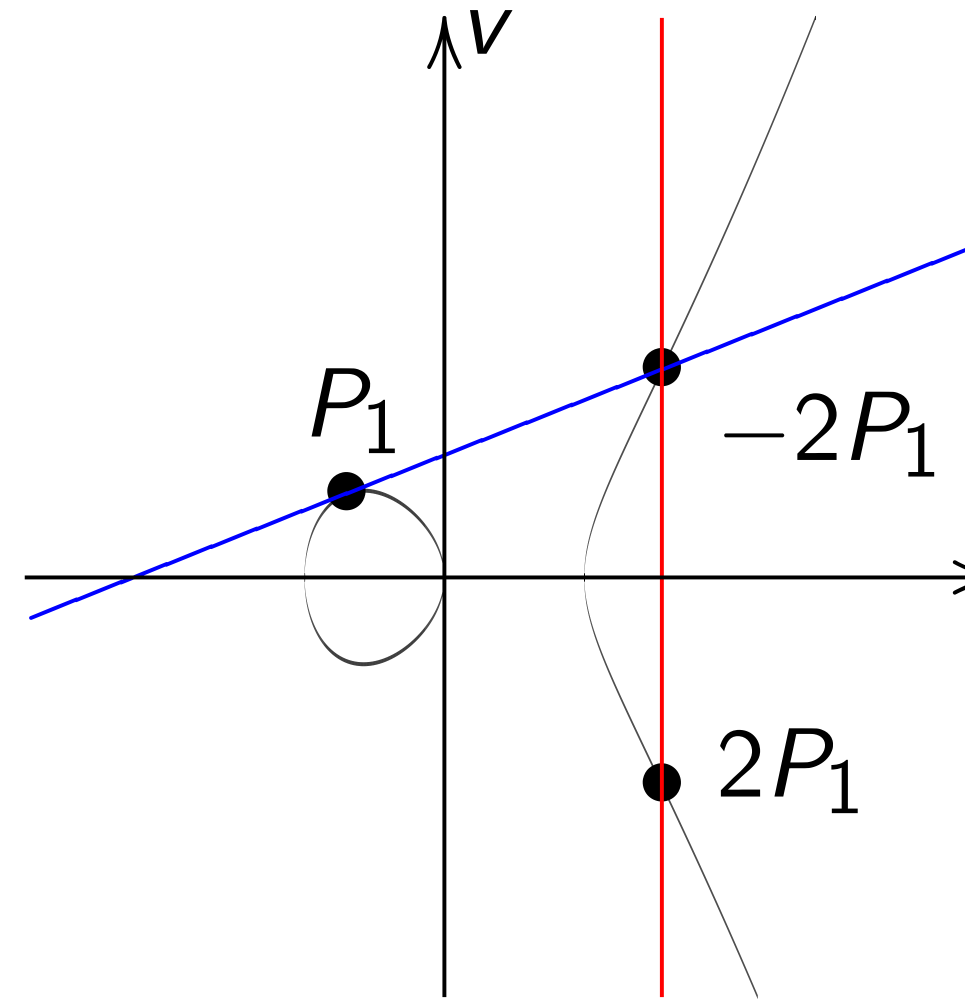


Slope $\lambda = (v_2 - v_1)/(u_2 - u_1)$.

Note that $u_1 \neq u_2$.

Doubling on Weierstrass cur

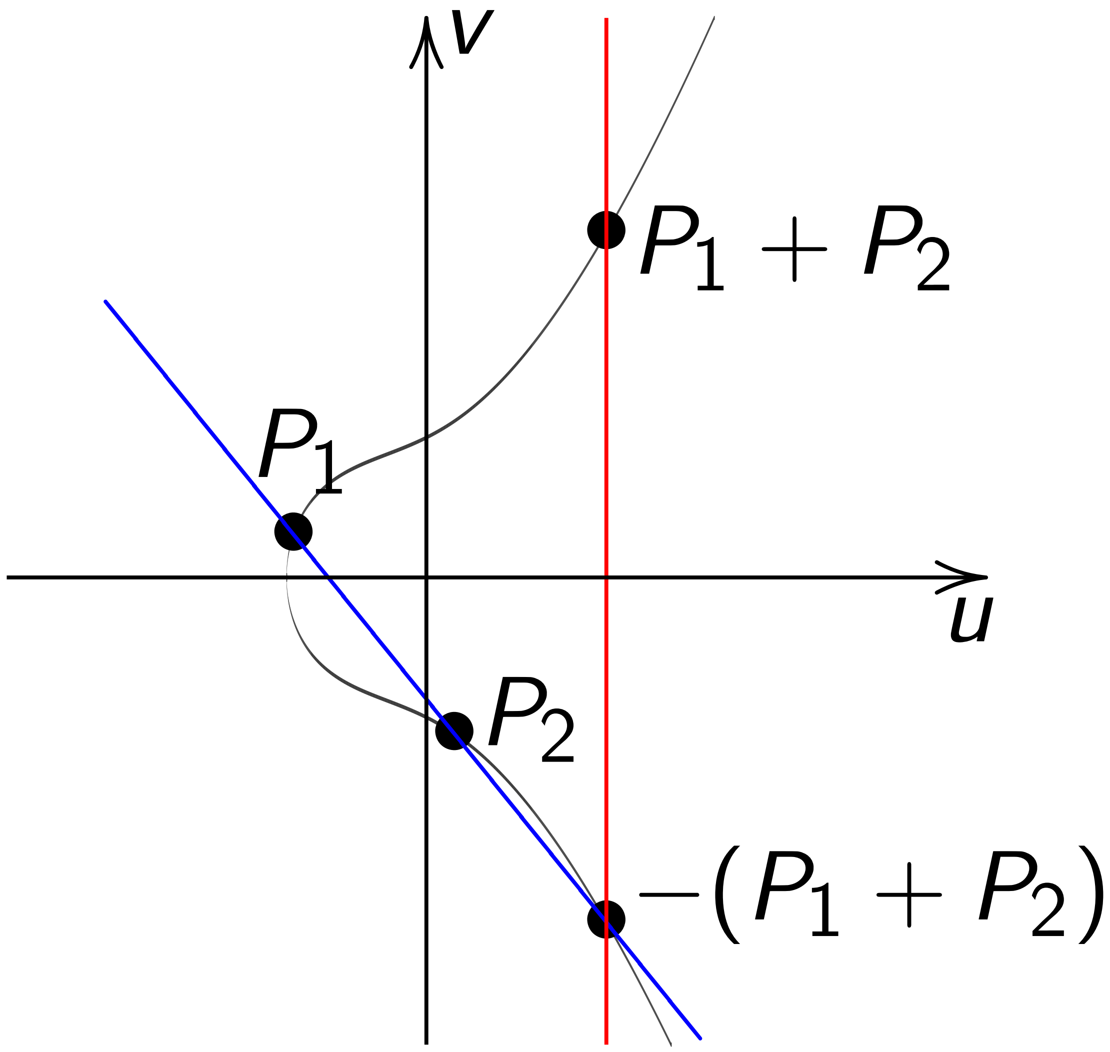
$v^2 = u^3 - u$



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

Addition on Weierstrass curve

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

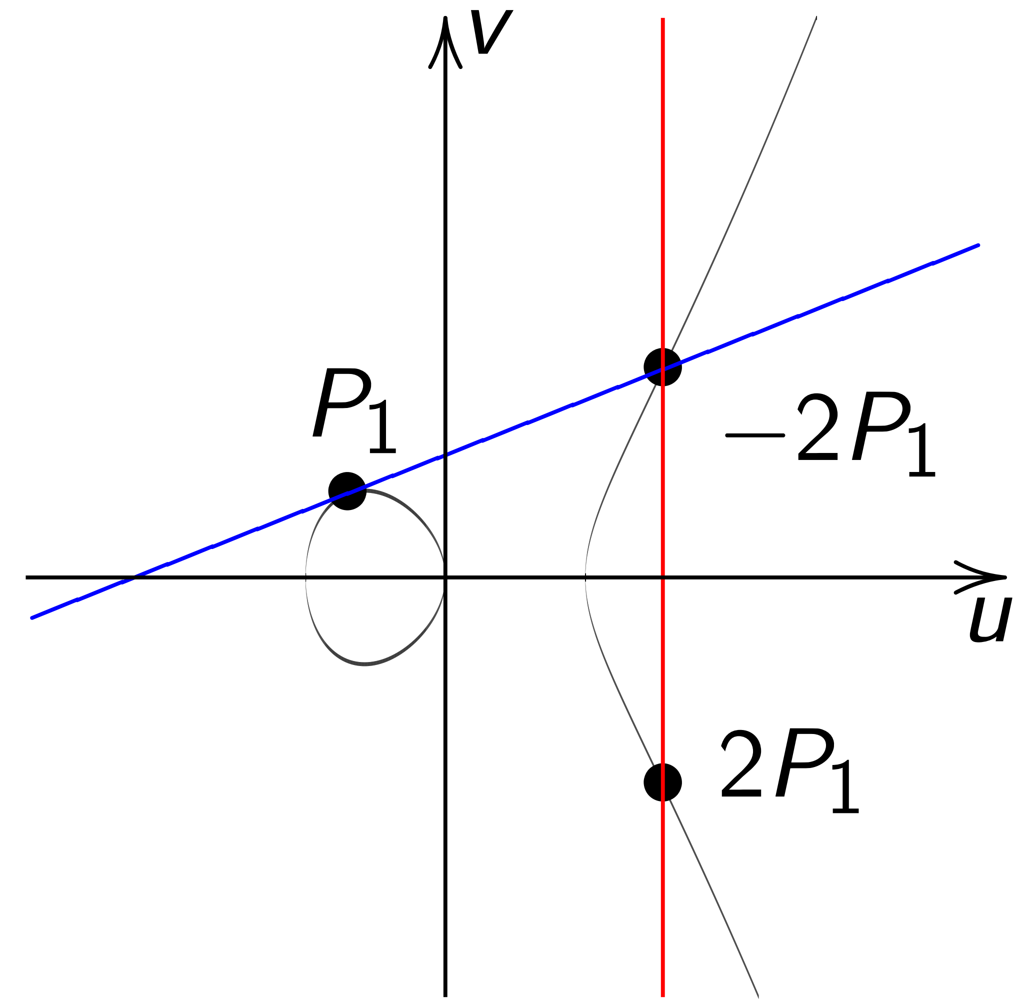


Slope $\lambda = (v_2 - v_1)/(u_2 - u_1)$.

Note that $u_1 \neq u_2$.

Doubling on Weierstrass curve

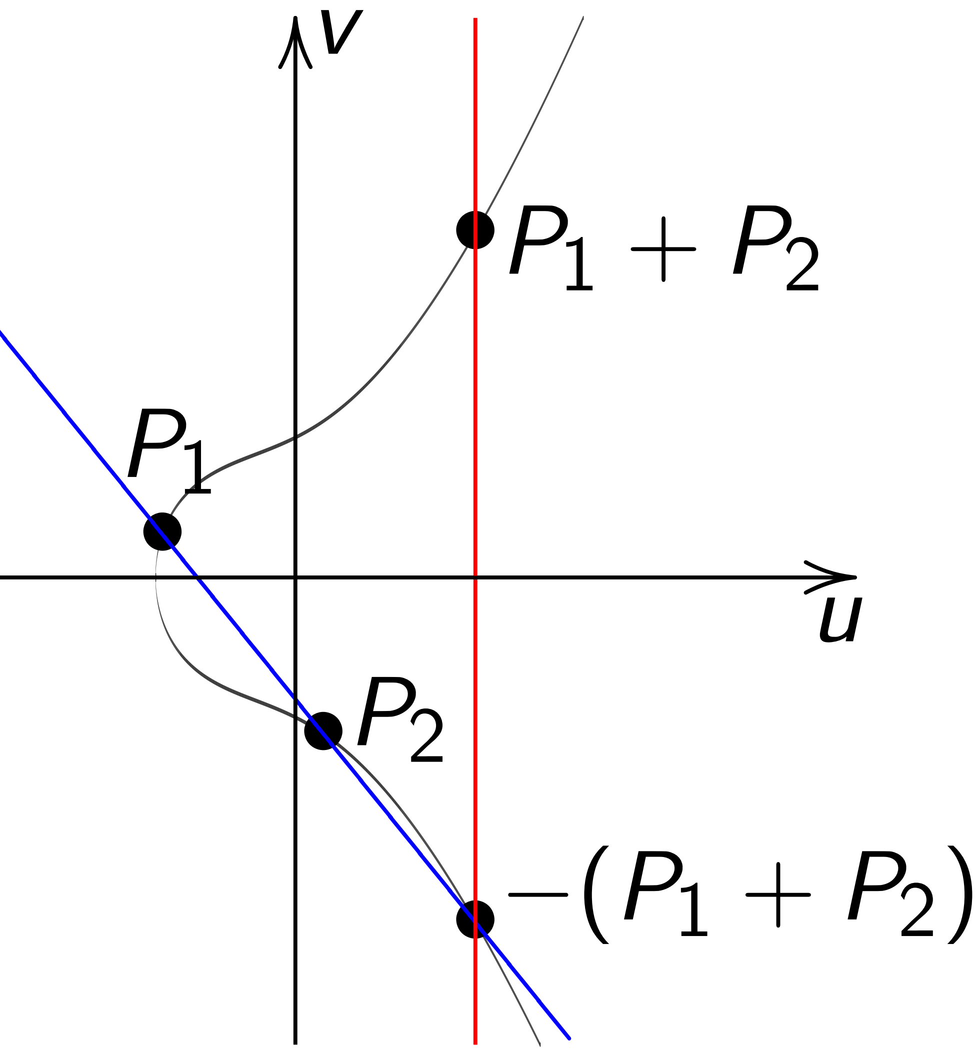
$v^2 = u^3 - u$



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

n on Weierstrass curve

$+ u^2 + u + 1$

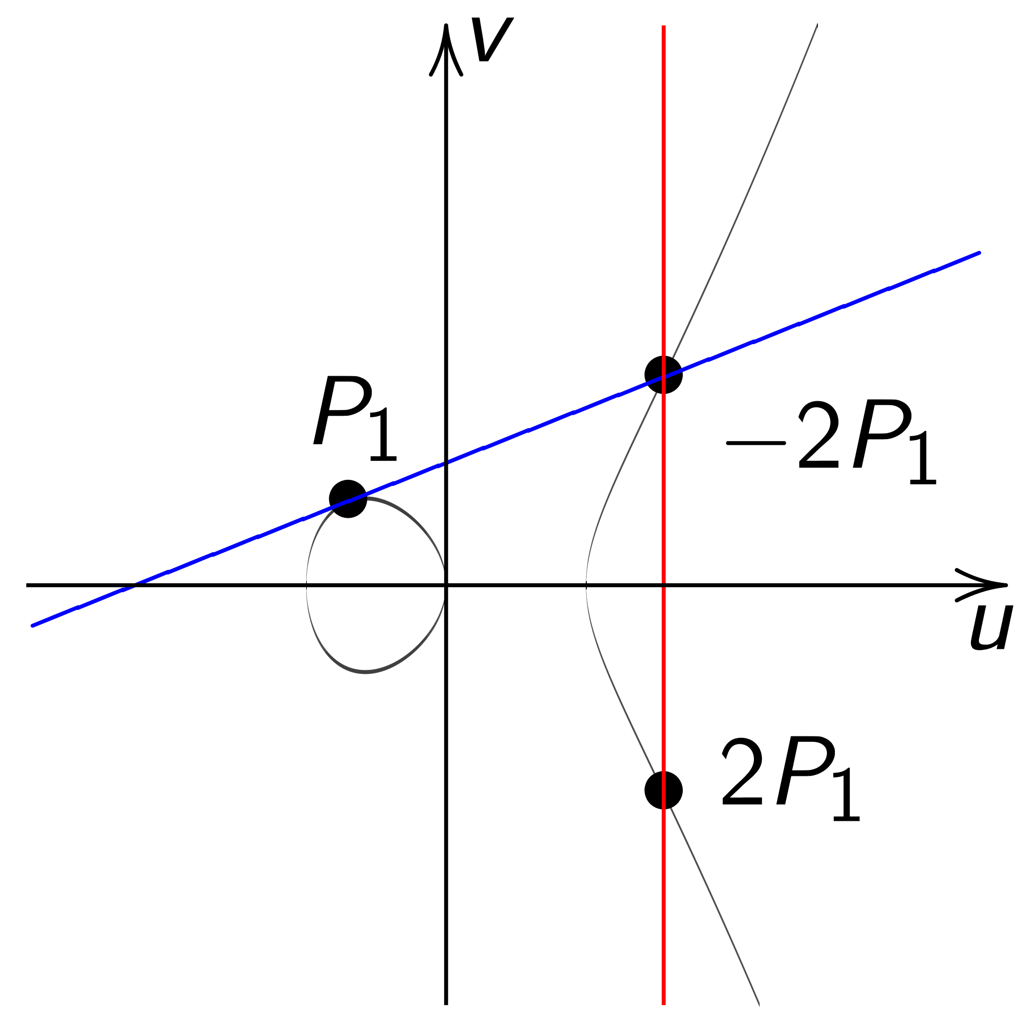


$= (v_2 - v_1)/(u_2 - u_1)$.

at $u_1 \neq u_2$.

Doubling on Weierstrass curve

$v^2 = u^3 - u$



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

In most

(u_1, v_1)

(u_3, v_3)

$(\lambda^2 - u_1$

$u_1 \neq u_2$

$\lambda = (v_2$

Total co

(u_1, v_1)

“doublin

$\lambda = (3u$

Total co

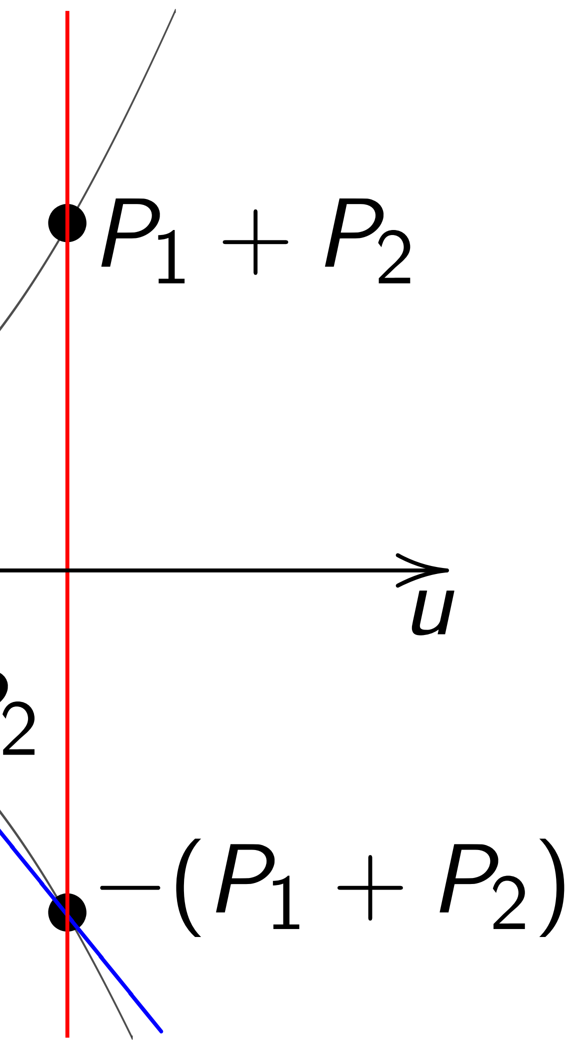
Also har

(u_1, v_1)

Messy to

rstrass curve

$+ 1$

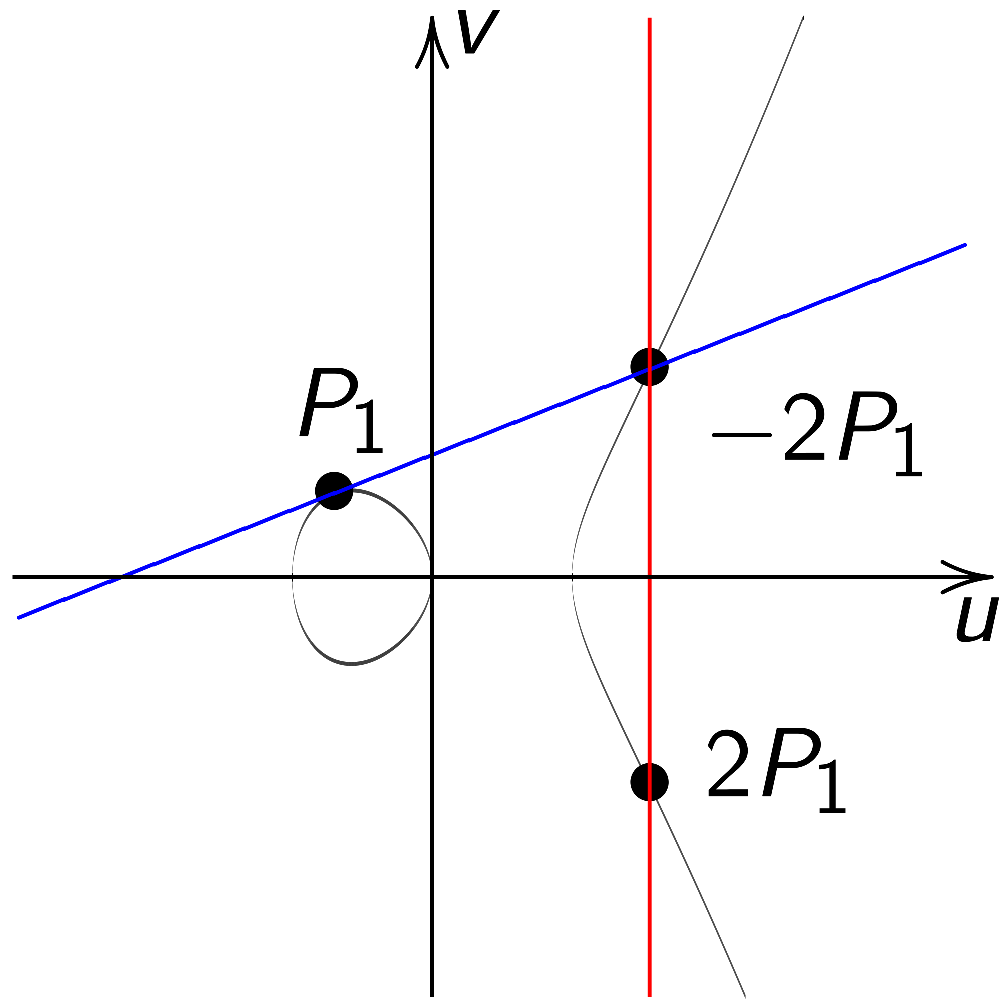


$_1)/(u_2 - u_1)$.

$_2$.

Doubling on Weierstrass curve

$v^2 = u^3 - u$



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

In most cases

$(u_1, v_1) + (u_2, v_2$

(u_3, v_3) where $(u_3$

$(\lambda^2 - u_1 - u_2, \lambda(u_1$

$u_1 \neq u_2$, "additio

$\lambda = (v_2 - v_1)/(u_2$

Total cost $1\mathbf{I} + 2\mathbf{M}$

$(u_1, v_1) = (u_2, v_2$

"doubling" (alert!

$\lambda = (3u_1^2 + 2a_2u_1$

Total cost $1\mathbf{I} + 2\mathbf{M}$

Also handle some

$(u_1, v_1) = (u_2, -v$

Messy to impleme

ve

2

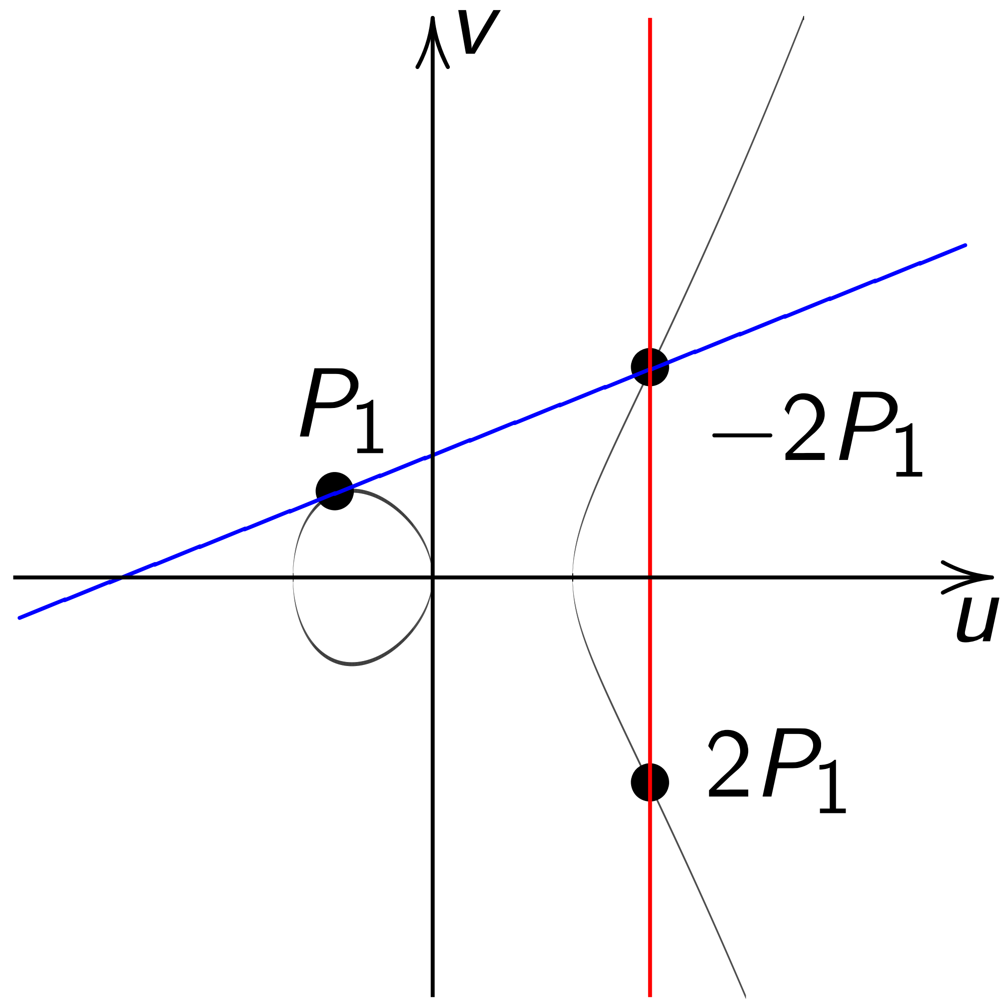
u

$P_2)$

$u_1).$

Doubling on Weierstrass curve

$v^2 = u^3 - u$



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

In most cases
$(u_1, v_1) + (u_2, v_2) =$
(u_3, v_3) where $(u_3, v_3) =$
$(\lambda^2 - u_1 - u_2, \lambda(u_1 - u_3) - v_1$

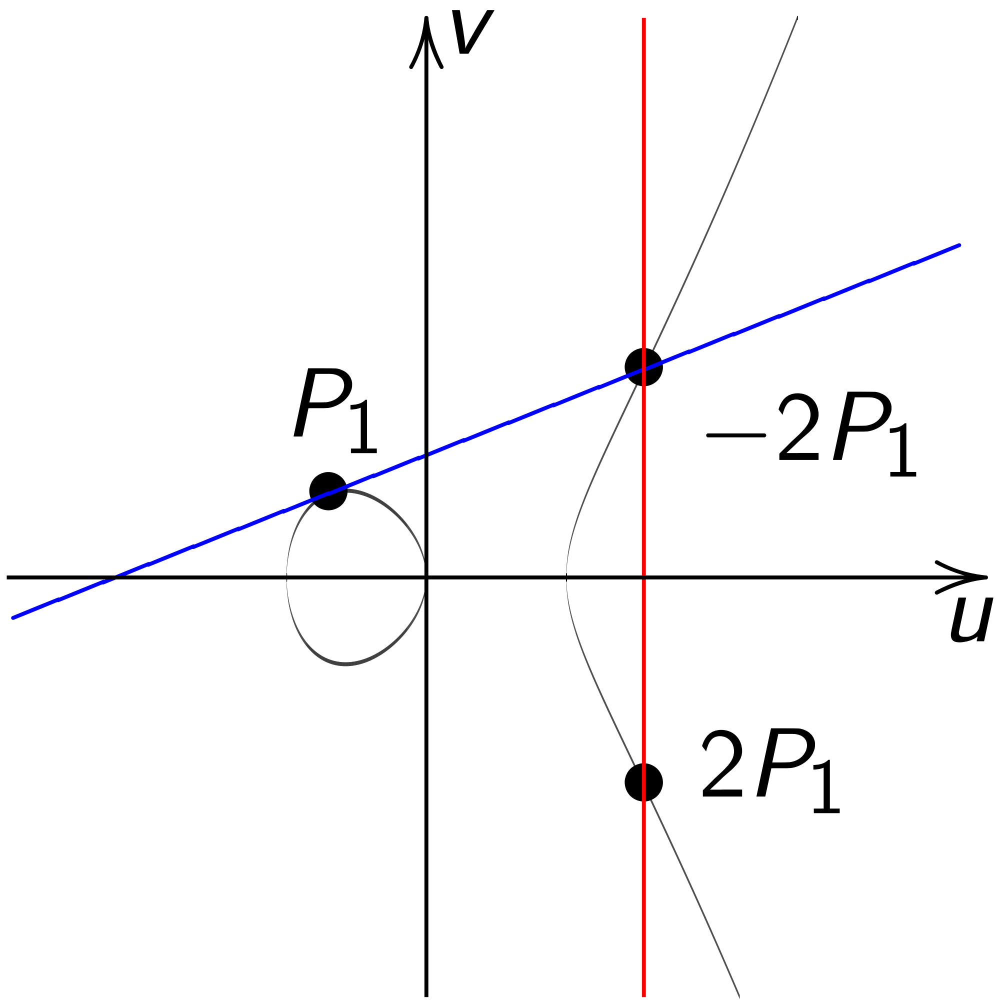
$u_1 \neq u_2$, "addition" (alert!)
$\lambda = (v_2 - v_1)/(u_2 - u_1).$
Total cost $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$.

$(u_1, v_1) = (u_2, v_2)$ and $v_1 \neq$
"doubling" (alert!):
$\lambda = (3u_1^2 + 2a_2u_1 + a_4)/(2$
Total cost $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

Also handle some exception
$(u_1, v_1) = (u_2, -v_2)$; ∞ as
Messy to implement and tes

Doubling on Weierstrass curve

$v^2 = u^3 - u$



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

In most cases
$(u_1, v_1) + (u_2, v_2) =$
(u_3, v_3) where $(u_3, v_3) =$
$(\lambda^2 - u_1 - u_2, \lambda(u_1 - u_3) - v_1)$.

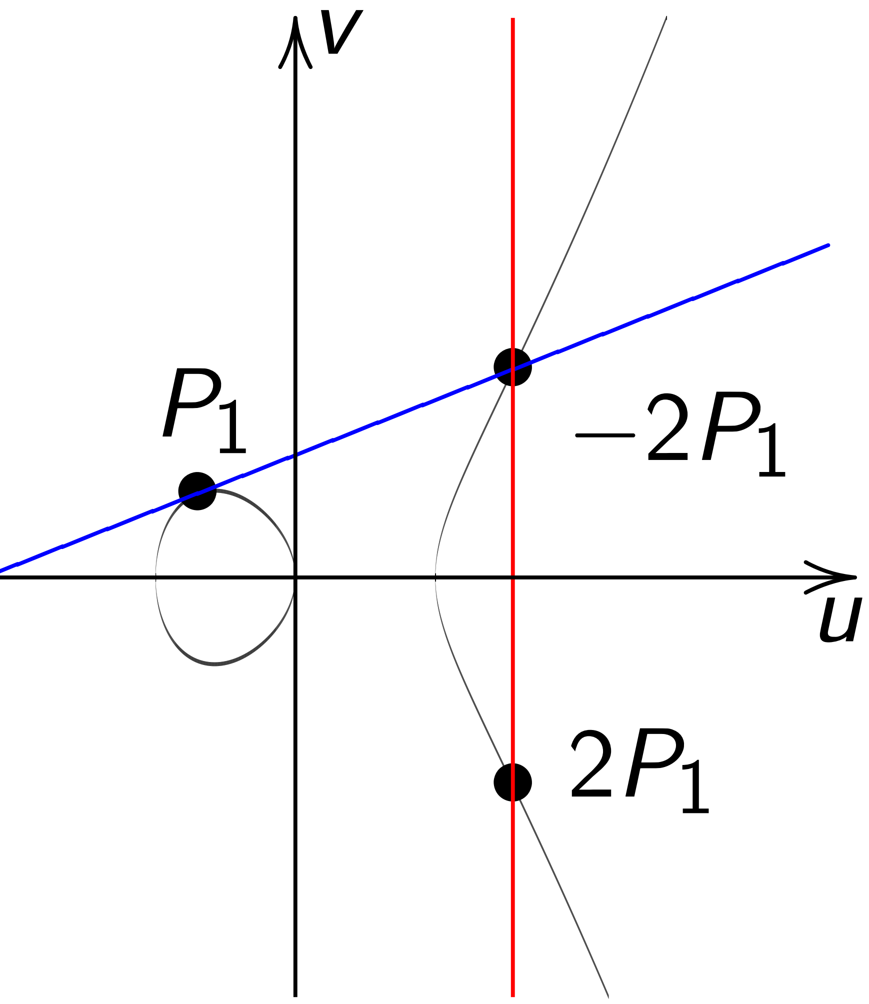
$u_1 \neq u_2$, "addition" (alert!):
$\lambda = (v_2 - v_1)/(u_2 - u_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$.

$(u_1, v_1) = (u_2, v_2)$ and $v_1 \neq 0$,
"doubling" (alert!):
$\lambda = (3u_1^2 + 2a_2u_1 + a_4)/(2v_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

Also handle some exceptions:
$(u_1, v_1) = (u_2, -v_2)$; ∞ as input.
Messy to implement and test.

g on Weierstrass curve

$- u$



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

In most cases

$(u_1, v_1) + (u_2, v_2) =$
(u_3, v_3) where $(u_3, v_3) =$
$(\lambda^2 - u_1 - u_2, \lambda(u_1 - u_3) - v_1)$.

$u_1 \neq u_2$, "addition" (alert!):
$\lambda = (v_2 - v_1)/(u_2 - u_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$.

$(u_1, v_1) = (u_2, v_2)$ and $v_1 \neq 0$,
"doubling" (alert!):
$\lambda = (3u_1^2 + 2a_2u_1 + a_4)/(2v_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

Also handle some exceptions:
$(u_1, v_1) = (u_2, -v_2)$; ∞ as input.
Messy to implement and test.

Biration

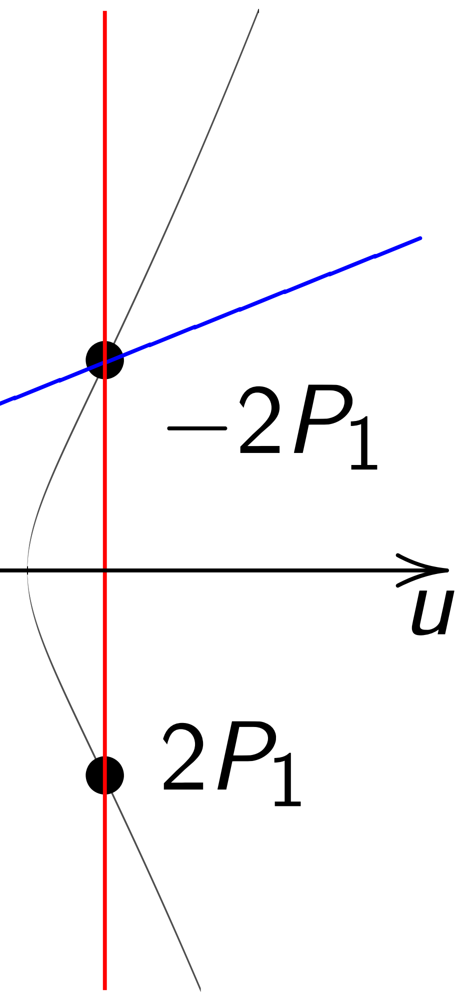
Starting
on $x^2 +$

Define A
$B = 4/($
$u = (1 -$
$v = u/x$
(Skip a

Then $(u$
a Weiers
$v^2 = u^3$

Easily in
$x = u/v$

rstrass curve



$1)/(2v_1)$.

In most cases

$(u_1, v_1) + (u_2, v_2) =$
(u_3, v_3) where $(u_3, v_3) =$
$(\lambda^2 - u_1 - u_2, \lambda(u_1 - u_3) - v_1)$.

$u_1 \neq u_2$, "addition" (alert!):
$\lambda = (v_2 - v_1)/(u_2 - u_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$.

$(u_1, v_1) = (u_2, v_2)$ and $v_1 \neq 0$,
"doubling" (alert!):
$\lambda = (3u_1^2 + 2a_2u_1 + a_4)/(2v_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

Also handle some exceptions:
$(u_1, v_1) = (u_2, -v_2)$; ∞ as input.
Messy to implement and test.

Birational equivale

Starting from poin
on $x^2 + y^2 = 1 +$

Define $A = 2(1 +$
$B = 4/(1 - d)$;
$u = (1 + y)/(B(1$
$v = u/x = (1 + y$
(Skip a few except

Then (u, v) is a p
a Weierstrass curv
$v^2 = u^3 + (A/B)u$

Easily invert this r
$x = u/v$, $y = (Bu$

rve

In most cases

$(u_1, v_1) + (u_2, v_2) =$
(u_3, v_3) where $(u_3, v_3) =$
$(\lambda^2 - u_1 - u_2, \lambda(u_1 - u_3) - v_1)$.

$u_1 \neq u_2$, "addition" (alert!):
$\lambda = (v_2 - v_1)/(u_2 - u_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$.

$(u_1, v_1) = (u_2, v_2)$ and $v_1 \neq 0$,
"doubling" (alert!):
$\lambda = (3u_1^2 + 2a_2u_1 + a_4)/(2v_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

Also handle some exceptions:
$(u_1, v_1) = (u_2, -v_2)$; ∞ as input.
Messy to implement and test.

Birational equivalence

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

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

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

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

In most cases

$(u_1, v_1) + (u_2, v_2) =$
(u_3, v_3) where $(u_3, v_3) =$
$(\lambda^2 - u_1 - u_2, \lambda(u_1 - u_3) - v_1)$.

$u_1 \neq u_2$, "addition" (alert!):
$\lambda = (v_2 - v_1)/(u_2 - u_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$.

$(u_1, v_1) = (u_2, v_2)$ and $v_1 \neq 0$,
"doubling" (alert!):
$\lambda = (3u_1^2 + 2a_2u_1 + a_4)/(2v_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

Also handle some exceptions:
$(u_1, v_1) = (u_2, -v_2)$; ∞ as input.
Messy to implement and test.

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

cases

$+ (u_2, v_2) =$

where $(u_3, v_3) =$

$-u_2, \lambda(u_1 - u_3) - v_1)$.

, "addition" (alert!):

$- v_1)/(u_2 - u_1)$.

st $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$.

$= (u_2, v_2)$ and $v_1 \neq 0$,

ng" (alert!):

$_1^2 + 2a_2 u_1 + a_4)/(2v_1)$.

st $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

ndle some exceptions:

$= (u_2, -v_2)$; ∞ as input.

o implement and test.

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

curve to

vice vers

$\Rightarrow$ Same

Can cho

so that i

is faster,

System c

represen

runs fast

about se

$) =$

$_3, v_3) =$

$-u_3)-v_1)$.

n" (alert!):

$_2 - u_1)$.

$\mathbf{M} + 1\mathbf{S}$.

$)$ and $v_1 \neq 0$,

$)$:

$+ a_4)/(2v_1)$.

$\mathbf{M} + 2\mathbf{S}$.

exceptions:

$_2)$; ∞ as input.

nt and test.

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 trans
curve to Weierstra
vice versa; $n(x, y)$
$\Rightarrow$ Same discrete-l
Can choose curve
so that implement
is faster/easier.

System designer ca
representation so t
runs fastest; no ne
about security deg

).

:

≠ 0,

v_1).

s:

input.

st.

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 Edw
curve to Weierstrass curve a
vice versa; $n(x, y) \mapsto n(u, v$
$\Rightarrow$ Same discrete-log securit
Can choose curve representa
so that implementation of a
is faster/easier.

System designer can choose
representation so that proto
runs fastest; no need to wor
about security degradation.

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.

Birational equivalence

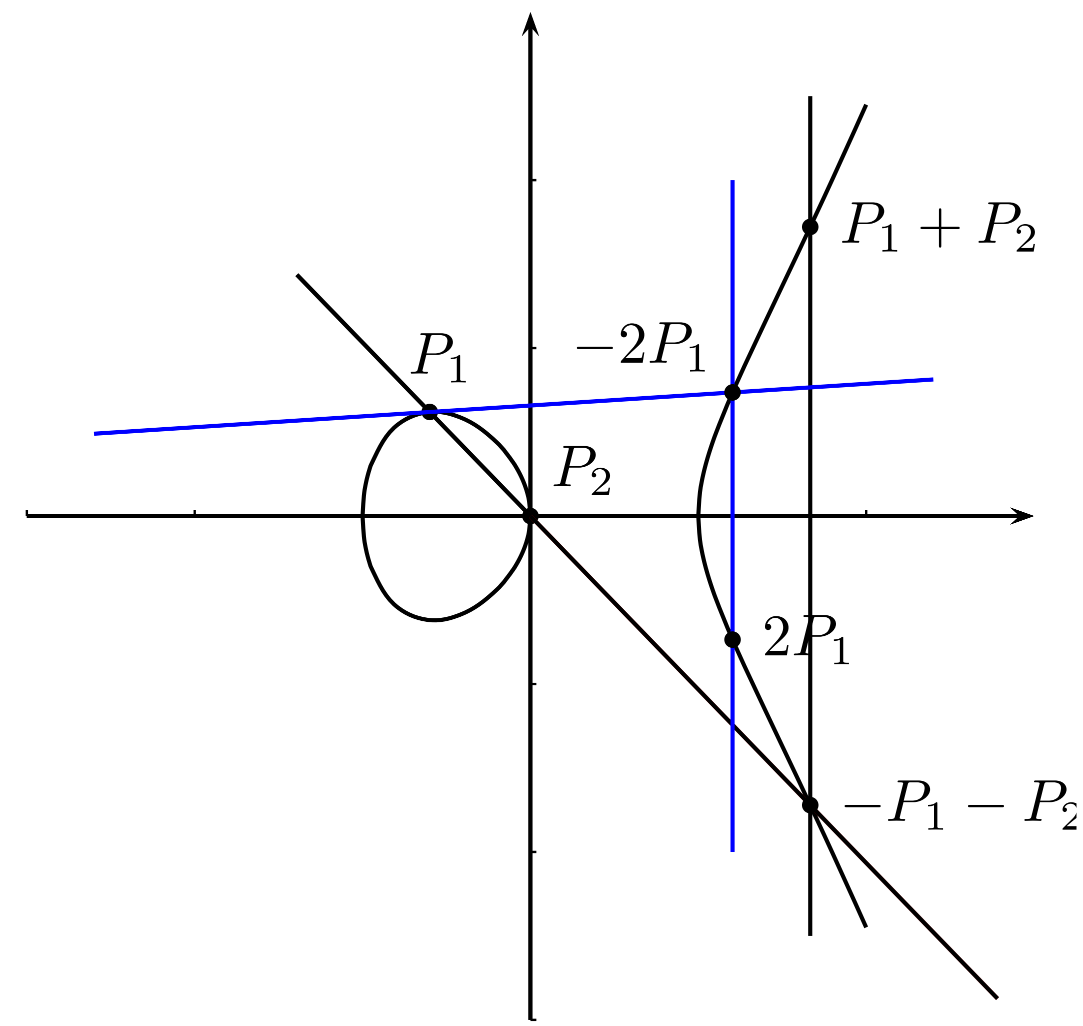
Starting from point (x, y) on $x^2 + y^2 = 1 + dx^2y^2$:

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

Elliptic-curve groups



Birational equivalence

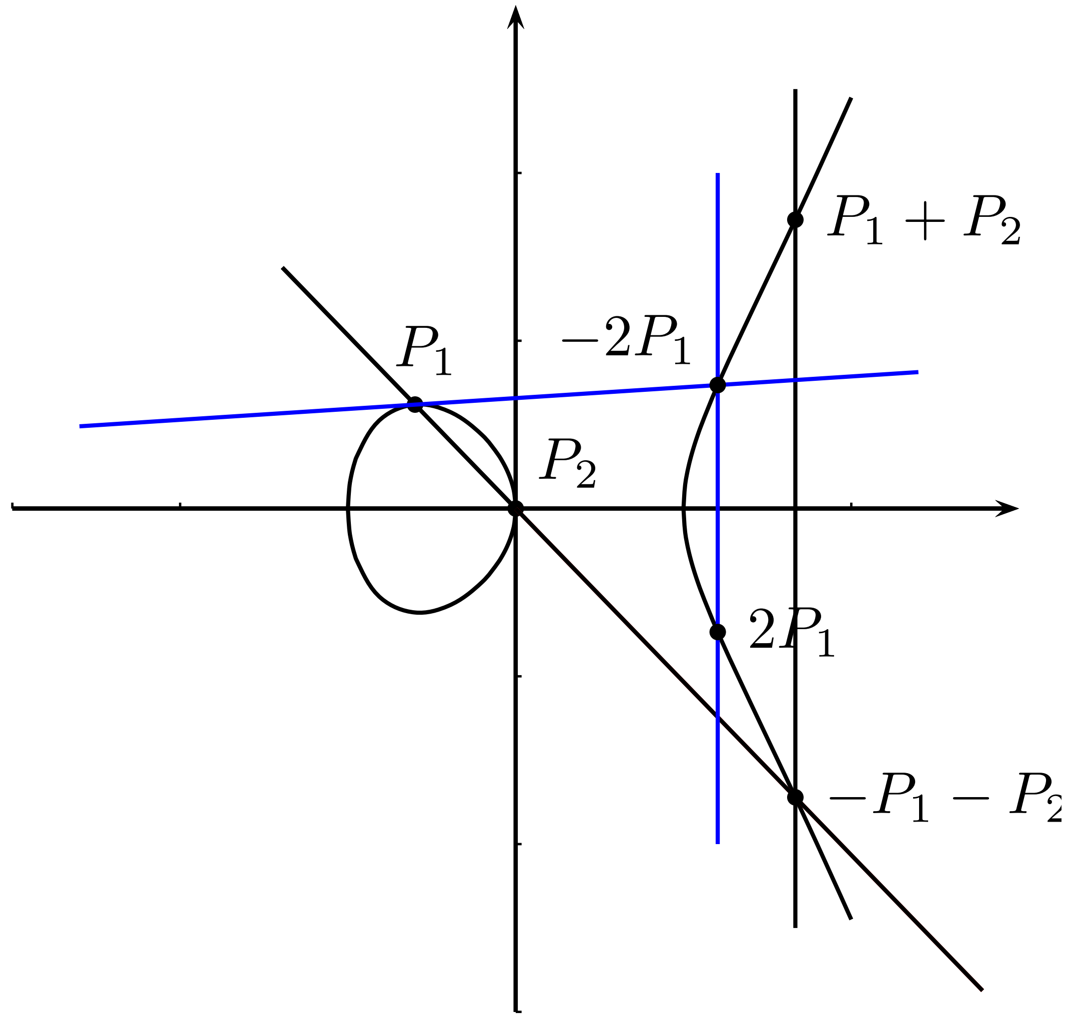
Starting from point (x, y) on $x^2 + y^2 = 1 + dx^2y^2$:

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

Elliptic-curve groups



Following algorithms will need a unique representative per point. For that Weierstrass curves are the speed leader.

al equivalence

from point (x, y)

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

$A = 2(1+d)/(1-d)$,

$(1-d)$;

$+ y)/(B(1-y))$,

$x = (1+y)/(Bx(1-y))$.

few exceptional points.)

$, v)$ is a point on

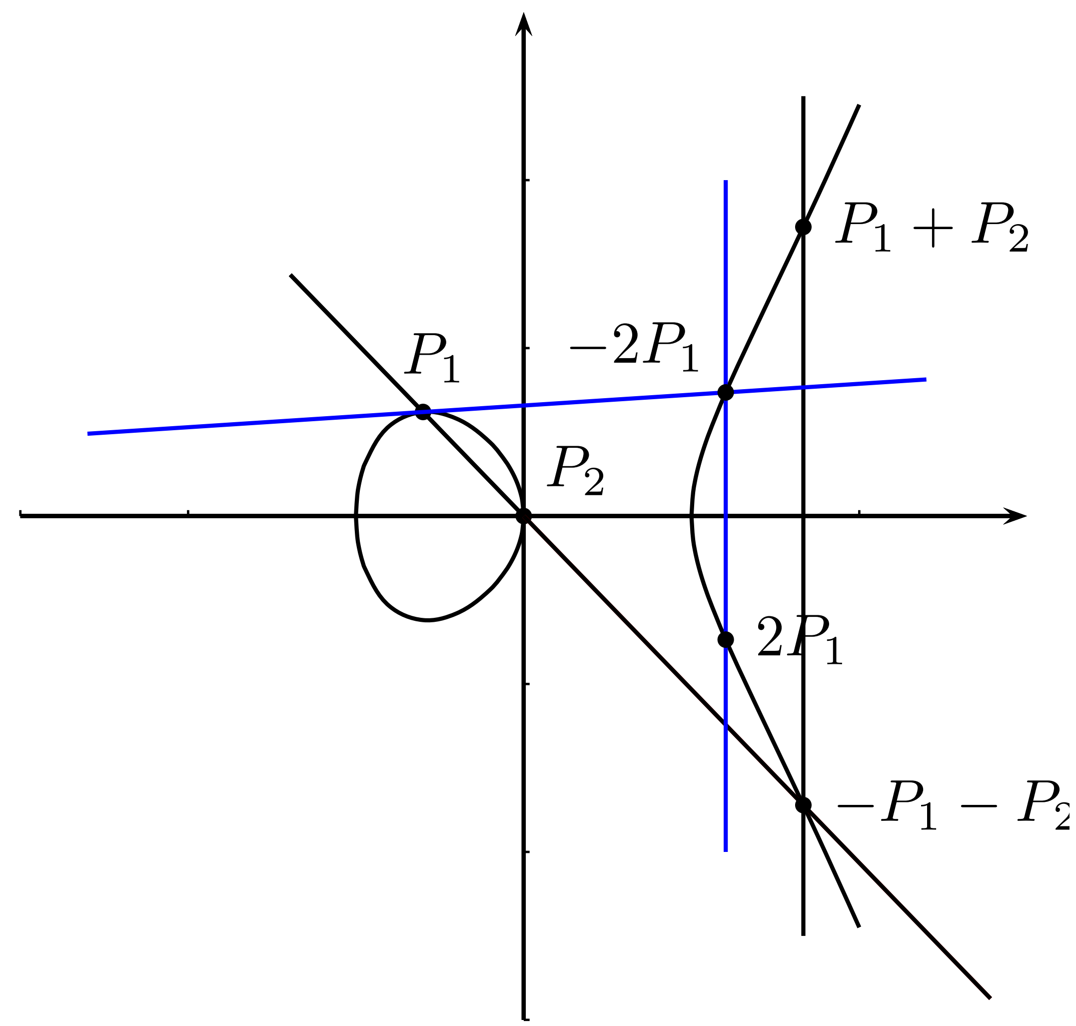
strass curve:

$+ (A/B)u^2 + (1/B^2)u$.

vert this map:

$, y = (Bu-1)/(Bu+1)$.

Elliptic-curve groups



Following algorithms will need a unique representative per point. For that Weierstrass curves are the speed leader.

The disc

Define p

consider

$y^2 = x^3$

This cur

ence

nt (x, y)

dx^2y^2:

$d)/(1 - d)$,

$- y))$,

$)/(Bx(1 - y))$.

tional points.)

oint on

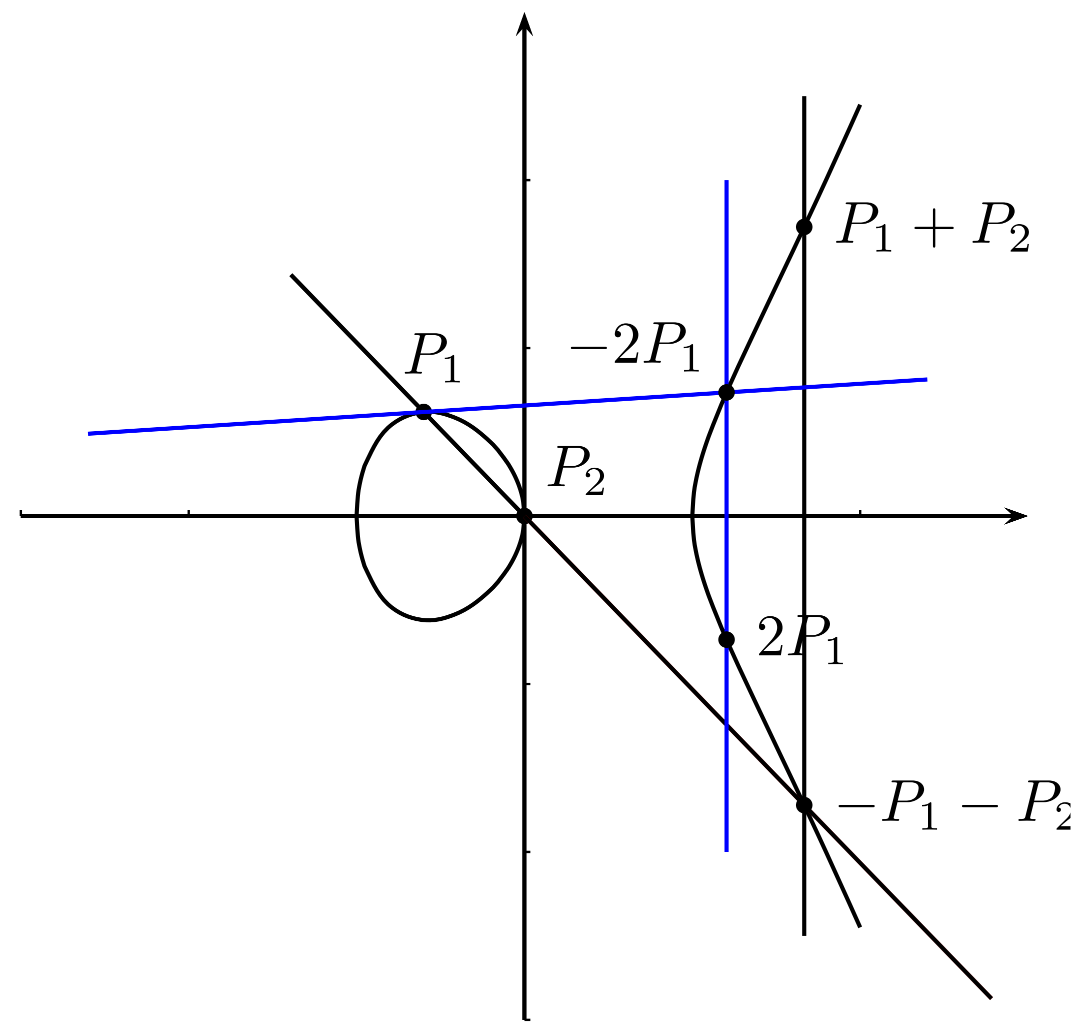
e:

$u^2 + (1/B^2)u$.

map:

$u - 1)/(Bu + 1)$.

Elliptic-curve groups



Following algorithms will need a unique representative per point. For that Weierstrass curves are the speed leader.

The discrete-logar

Define $p = 100000$

consider the Weier

$y^2 = x^3 - x$ over

This curve has

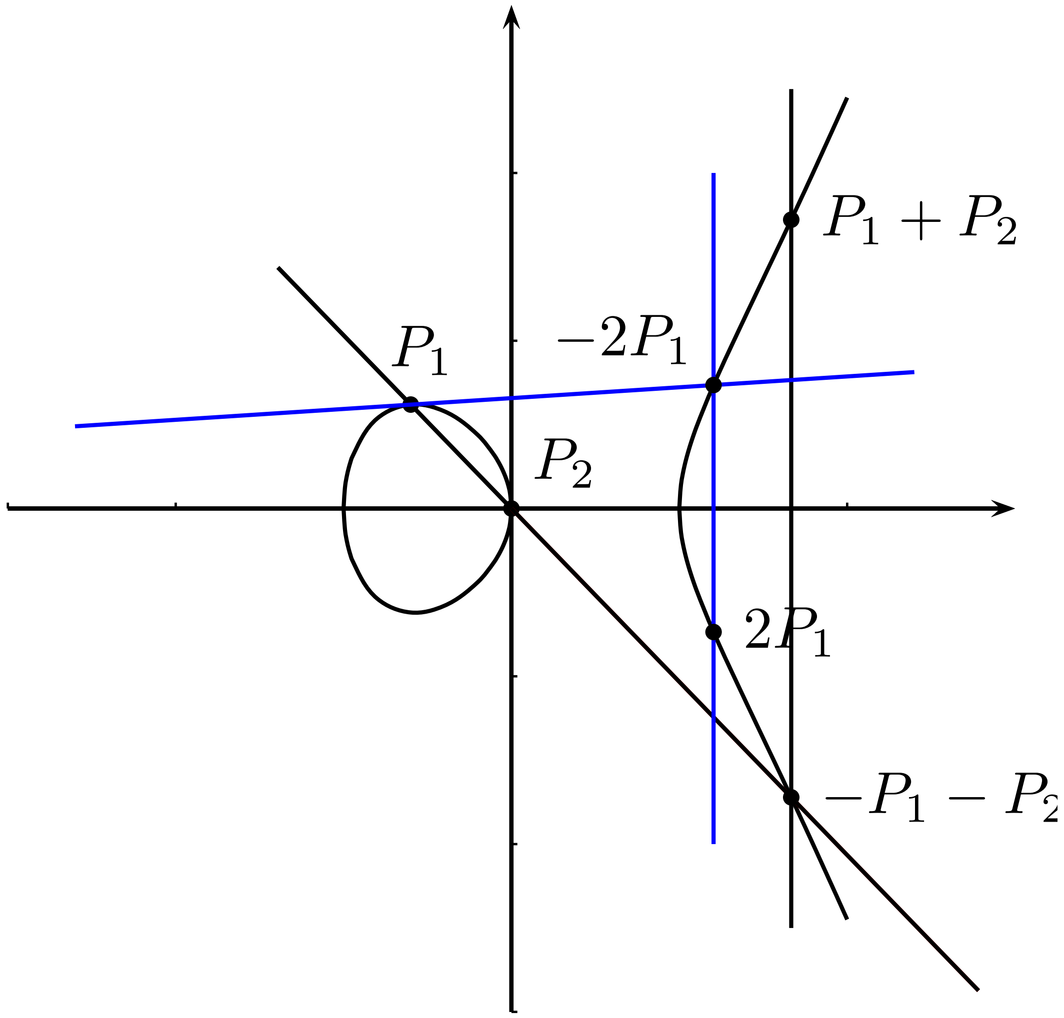
),

$-\ y))$.
ts.)

$^2)u$.

$u + 1)$.

Elliptic-curve groups

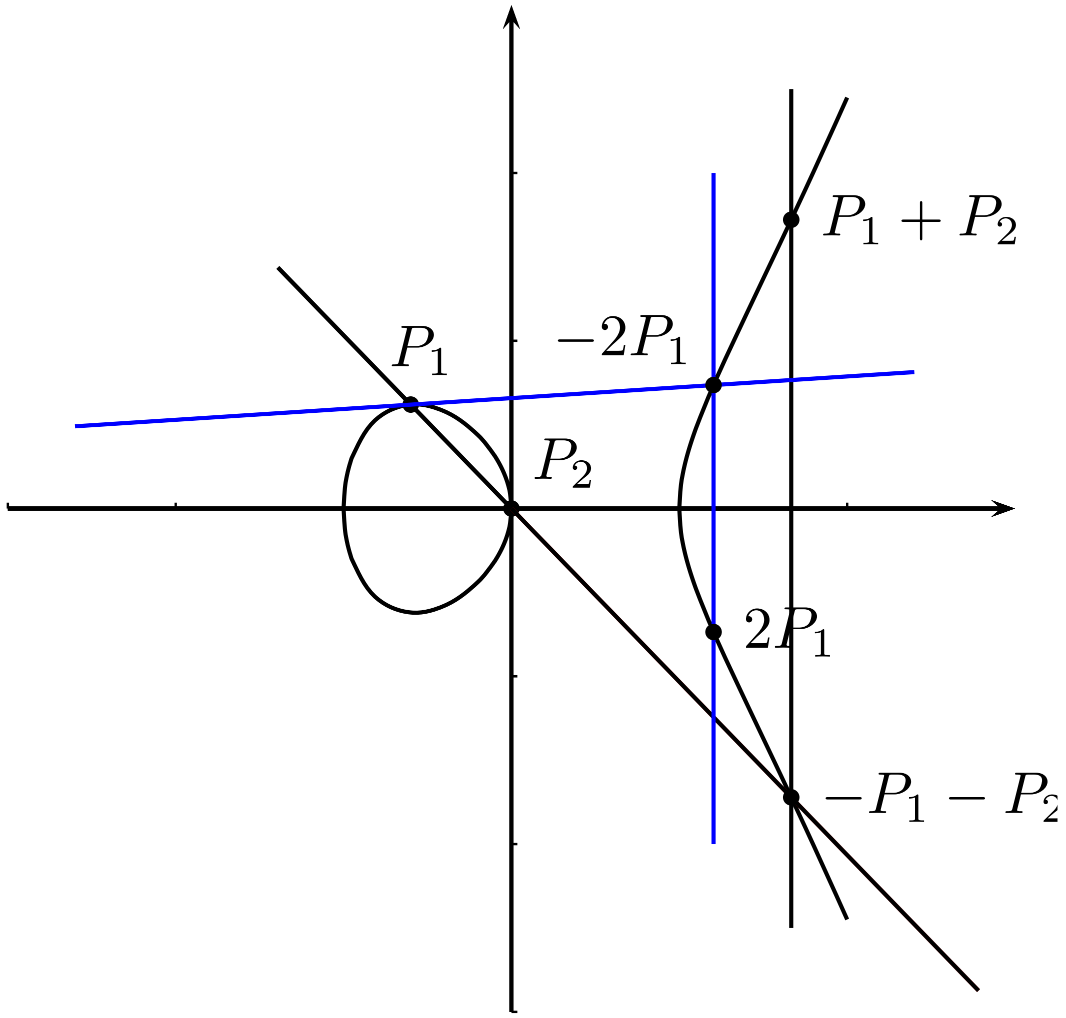


Following algorithms will need a unique representative per point. For that Weierstrass curves are the speed leader.

The discrete-logarithm probl

Define $p = 1000003$ and consider the Weierstrass cur $y^2 = x^3 - x$ over $\mathbf{F}_p$. This curve has

Elliptic-curve groups

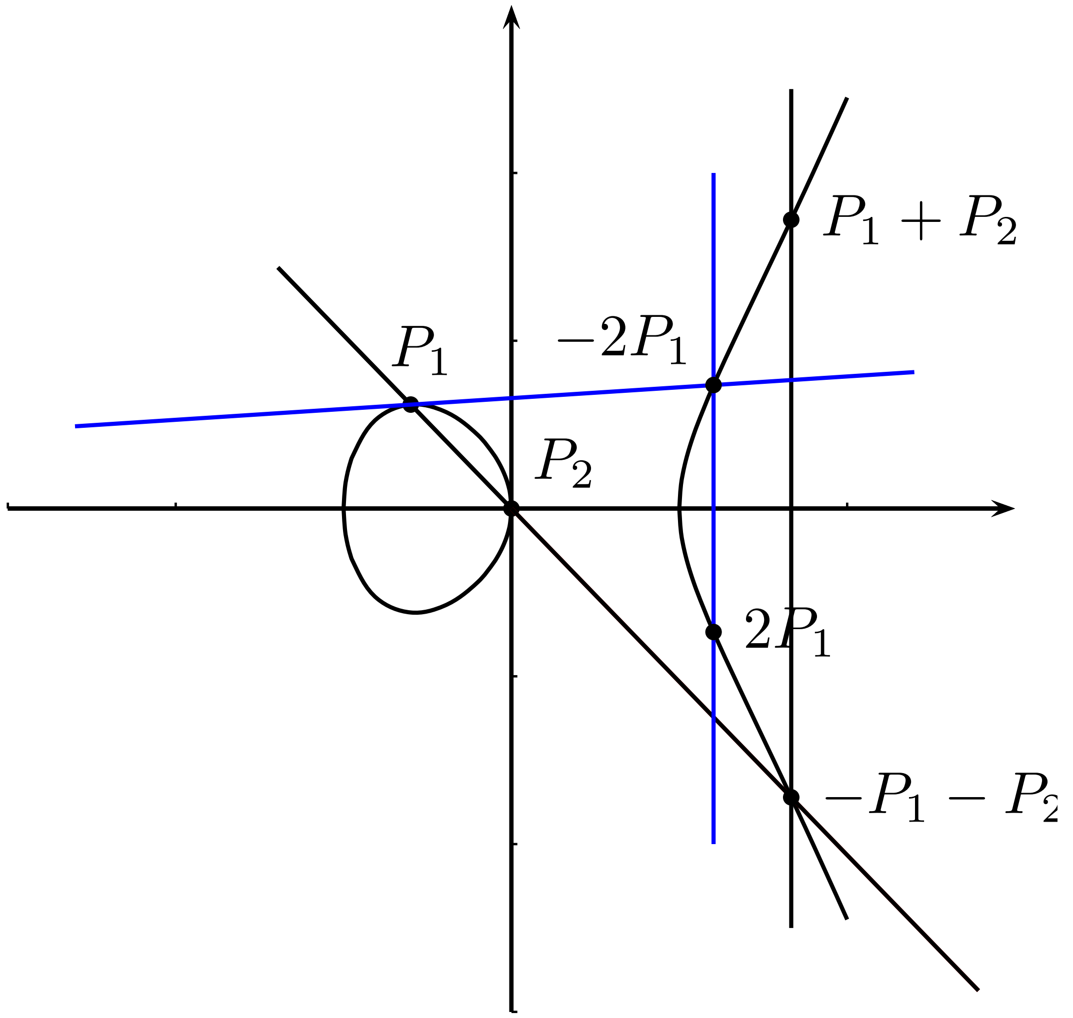


Following algorithms will need a unique representative per point. For that Weierstrass curves are the speed leader.

The discrete-logarithm problem

Define $p = 1000003$ and consider the Weierstrass curve $y^2 = x^3 - x$ over $\mathbf{F}_p$. This curve has

Elliptic-curve groups

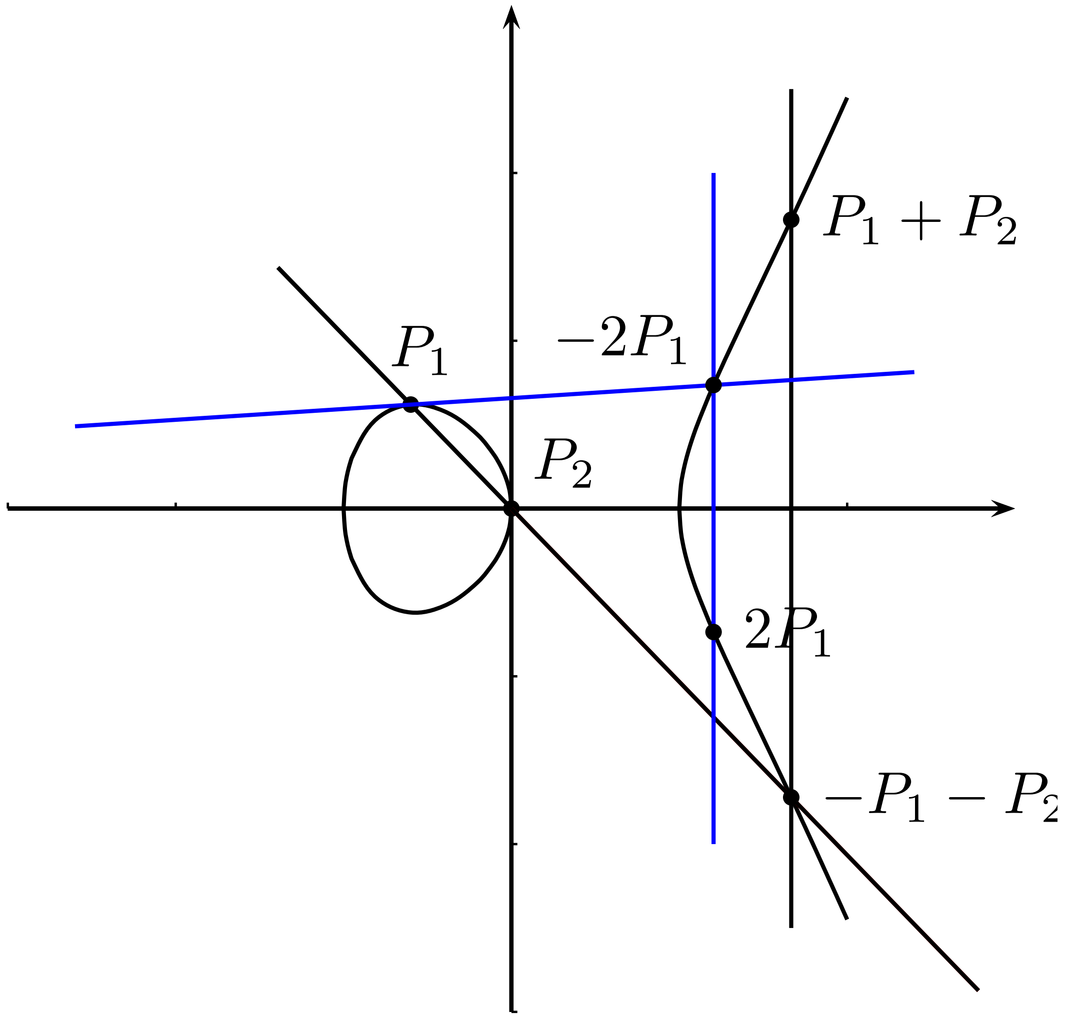


Following algorithms will need a unique representative per point. For that Weierstrass curves are the speed leader.

The discrete-logarithm problem

Define $p = 1000003$ and consider the Weierstrass curve $y^2 = x^3 - x$ over $\mathbf{F}_p$.
This curve has
$1000004 = 2^2 \cdot 53^2 \cdot 89$
points and $P = (101384, 614510)$ is a point of order $2 \cdot 53^2 \cdot 89$.

Elliptic-curve groups



Following algorithms will need a unique representative per point. For that Weierstrass curves are the speed leader.

The discrete-logarithm problem

Define $p = 1000003$ and consider the Weierstrass curve $y^2 = x^3 - x$ over $\mathbf{F}_p$.

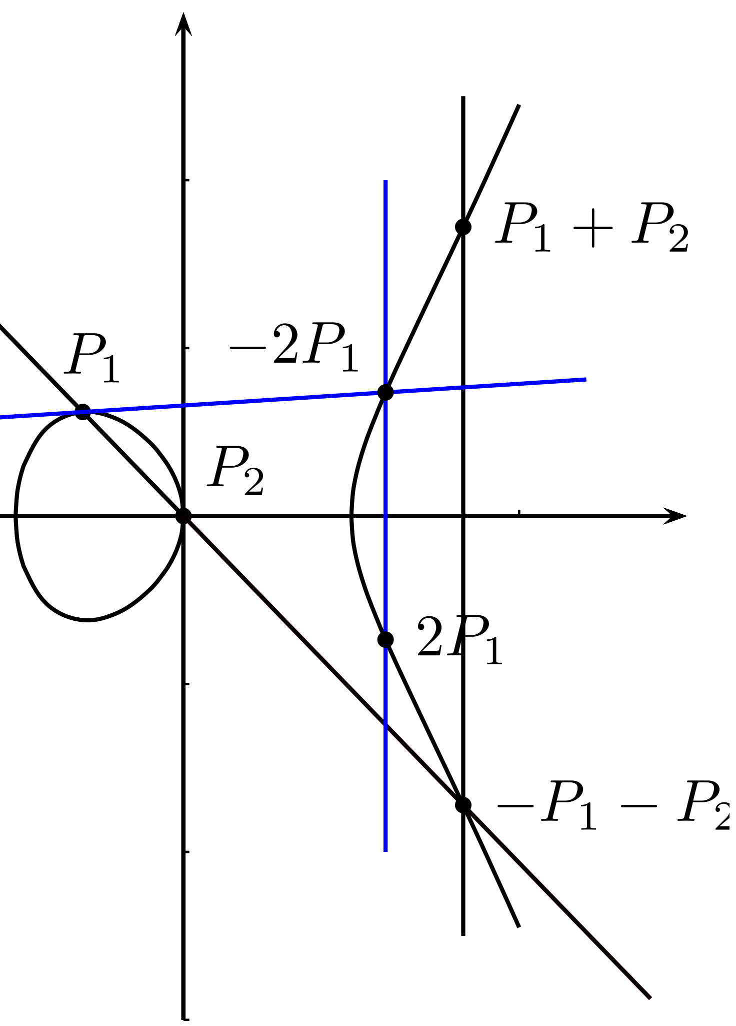
This curve has $1000004 = 2^2 \cdot 53^2 \cdot 89$ points and $P = (101384, 614510)$ is a point of order $2 \cdot 53^2 \cdot 89$.

In general, point counting over $\mathbf{F}_p$ runs in time polynomial in $\log p$.

Number of points in $[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]$.

The group is isomorphic to $\mathbf{Z}/n \times \mathbf{Z}/m$, where $n \mid m$ and $n \mid (p - 1)$.

curve groups



g algorithms will need a
epresentative per point.
Weierstrass curves are
d leader.

The discrete-logarithm problem

Define $p = 1000003$ and consider the Weierstrass curve $y^2 = x^3 - x$ over $\mathbf{F}_p$.

This curve has $1000004 = 2^2 \cdot 53^2 \cdot 89$ points and $P = (101384, 614510)$ is a point of order $2 \cdot 53^2 \cdot 89$.

In general, point counting over $\mathbf{F}_p$ runs in time polynomial in $\log p$.

Number of points in $[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]$.

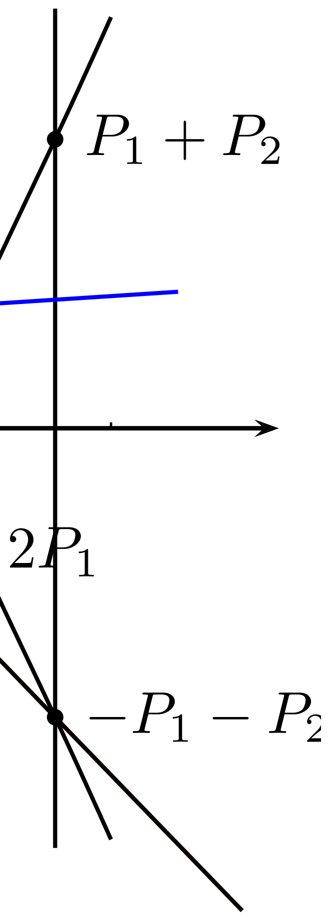
The group is isomorphic to $\mathbf{Z}/n \times \mathbf{Z}/m$, where $n \mid m$ and $n \mid (p - 1)$.

Can we
$n \in \{1,$
such tha
(670366

This poi
a multip
outside

Could fi
Is there

ps



ms will need a
tive per point.
ss curves are

The discrete-logarithm problem

Define $p = 1000003$ and consider the Weierstrass curve $y^2 = x^3 - x$ over $\mathbf{F}_p$.

This curve has $1000004 = 2^2 \cdot 53^2 \cdot 89$ points and $P = (101384, 614510)$ is a point of order $2 \cdot 53^2 \cdot 89$.

In general, point counting over $\mathbf{F}_p$ runs in time polynomial in $\log p$.

Number of points in $[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]$.

The group is isomorphic to $\mathbf{Z}/n \times \mathbf{Z}/m$, where $n \mid m$ and $n \mid (p - 1)$.

Can we find an int
$n \in \{1, 2, 3, \ldots, 5$
such that $nP \quad =$
$(670366, 740819)$?

This point was ge
a multiple of P; c
outside cyclic grou

Could find n by br
Is there a faster w

ed a
oint.
are

The discrete-logarithm problem

Define $p = 1000003$ and consider the Weierstrass curve $y^2 = x^3 - x$ over $\mathbf{F}_p$.

This curve has $1000004 = 2^2 \cdot 53^2 \cdot 89$ points and $P = (101384, 614510)$ is a point of order $2 \cdot 53^2 \cdot 89$.

In general, point counting over $\mathbf{F}_p$ runs in time polynomial in $\log p$.

Number of points in $[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]$.

The group is isomorphic to $\mathbf{Z}/n \times \mathbf{Z}/m$, where $n \mid m$ and $n \mid (p - 1)$.

Can we find an integer $n \in \{1, 2, 3, \ldots, 500001\}$ such that $nP = (670366, 740819)$?

This point was generated as a multiple of P; could also b outside cyclic group.

Could find n by brute force. Is there a faster way?

The discrete-logarithm problem

Define $p = 1000003$ and consider the Weierstrass curve $y^2 = x^3 - x$ over $\mathbf{F}_p$.

This curve has $1000004 = 2^2 \cdot 53^2 \cdot 89$ points and $P = (101384, 614510)$ is a point of order $2 \cdot 53^2 \cdot 89$.

In general, point counting over $\mathbf{F}_p$ runs in time polynomial in $\log p$.

Number of points in $[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]$.

The group is isomorphic to $\mathbf{Z}/n \times \mathbf{Z}/m$, where $n \mid m$ and $n \mid (p - 1)$.

Can we find an integer $n \in \{1, 2, 3, \ldots, 500001\}$ such that $nP = (670366, 740819)$?

This point was generated as a multiple of P; could also be outside cyclic group.

Could find n by brute force.
Is there a faster way?

crete-logarithm problem

$p = 1000003$ and
the Weierstrass curve
$- x$ over $\mathbf{F}_p$.
ve has
$= 2^2 \cdot 53^2 \cdot 89$
nd $P = (101384, 614510)$
nt of order $2 \cdot 53^2 \cdot 89$.
al, point counting over $\mathbf{F}_p$
time polynomial in $\log p$.
of points in
$2\sqrt{p}, p + 1 + 2\sqrt{p}]$.
up is isomorphic to
$\mathbf{Z}/m$, where $n \mid m$ and
1).

Can we find an integer
$n \in \{1, 2, 3, \ldots, 500001\}$
such that $nP =$
$(670366, 740819)$?

This point was generated as
a multiple of P; could also be
outside cyclic group.

Could find n by brute force.
Is there a faster way?

Understa

Can com
$1P = (1$
$2P = (1$
$3P = (7$
$4P = (6$
$500001P$
$500002P$

At some
with nP

Maximu
≤ 50000
≤ 50000
that doe

ithm problem

03 and

rstrass curve

$\mathbf{F}_p$.

$^2 \cdot 89$

$101384, 614510)$

$2 \cdot 53^2 \cdot 89$.

counting over $\mathbf{F}_p$

omial in $\log p$.

in

$1 + 2\sqrt{p}]$.

orphic to

re $n \mid m$ and

Can we find an integer $n \in \{1, 2, 3, \ldots, 500001\}$ such that $nP = (670366, 740819)$?

This point was generated as a multiple of P; could also be outside cyclic group.

Could find n by brute force.
Is there a faster way?

Understanding bru

Can compute succ

$1P = (101384, 61$

$2P = (102361, 62$

$3P = (77571, 876$

$4P = (650289, 31$

$500001P = -P$.

$500002P = \infty$.

At some point we'

with $nP = (6703$

Maximum cost of

≤ 500001 addition

≤ 500001 nanosec

that does 1 ADD/

lem

ve

4510)

9.

ver $\mathbf{F}_p$

og p.

.

and

Can we find an integer $n \in \{1, 2, 3, \ldots, 500001\}$ such that $nP = (670366, 740819)$?

This point was generated as a multiple of P; could also be outside cyclic group.

Could find n by brute force.
Is there a faster way?

Understanding brute force

Can compute successively
$1P = (101384, 614510)$,
$2P = (102361, 628914)$,
$3P = (77571, 87643)$,
$4P = (650289, 31313)$,
$500001P = -P$.
$500002P = \infty$.

At some point we'll find n with $nP = (670366, 740819$

Maximum cost of computati
≤ 500001 additions of P;
≤ 500001 nanoseconds on a
that does 1 ADD/nanosecor

Can we find an integer $n \in \{1, 2, 3, \ldots, 500001\}$ such that $nP = (670366, 740819)$?

This point was generated as a multiple of P; could also be outside cyclic group.

Could find n by brute force.
Is there a faster way?

Understanding brute force

Can compute successively
$1P = (101384, 614510)$,
$2P = (102361, 628914)$,
$3P = (77571, 87643)$,
$4P = (650289, 31313)$,
$500001P = -P$.
$500002P = \infty$.

At some point we'll find n with $nP = (670366, 740819)$.

Maximum cost of computation:
≤ 500001 additions of P;
≤ 500001 nanoseconds on a CPU that does 1 ADD/nanosecond.

find an integer
2, 3, ..., 500001}
nt nP =
, 740819)?

nt was generated as
le of P; could also be
cyclic group.

nd n by brute force.
a faster way?

Understanding brute force

Can compute successively
$1P = (101384, 614510)$,
$2P = (102361, 628914)$,
$3P = (77571, 87643)$,
$4P = (650289, 31313)$,
$500001P = -P$.
$500002P = \infty$.

At some point we'll find n
with $nP = (670366, 740819)$.

Maximum cost of computation:
≤ 500001 additions of P;
≤ 500001 nanoseconds on a CPU
that does 1 ADD/nanosecond.

This is n
for $p \approx 2$

But user
standard
making

Attack c
$\approx 2^{50}$ A
$\approx 2^{100}$ A

(Not exa
cost of A
But this

teger

00001}

nerated as

ould also be

up.

rute force.

ay?

Understanding brute force

Can compute successively

$1P = (101384, 614510)$,

$2P = (102361, 628914)$,

$3P = (77571, 87643)$,

$4P = (650289, 31313)$,

$500001P = -P$.

$500002P = \infty$.

At some point we'll find n with $nP = (670366, 740819)$.

Maximum cost of computation:

≤ 500001 additions of P;

≤ 500001 nanoseconds on a CPU that does 1 ADD/nanosecond.

This is negligible v

for $p \approx 2^{20}$.

But users can

standardize a large

making the attack

Attack cost scales

$\approx 2^{50}$ ADDs for p

$\approx 2^{100}$ ADDs for

(Not exactly linear

cost of ADDs grov

But this is a mino

be

Understanding brute force

Can compute successively

$1P = (101384, 614510)$,

$2P = (102361, 628914)$,

$3P = (77571, 87643)$,

$4P = (650289, 31313)$,

$500001P = -P$.

$500002P = \infty$.

At some point we'll find n with $nP = (670366, 740819)$.

Maximum cost of computation: ≤ 500001 additions of P; ≤ 500001 nanoseconds on a CPU that does 1 ADD/nanosecond.

This is negligible work for $p \approx 2^{20}$.

But users can standardize a larger p, making the attack slower.

Attack cost scales linearly:

$\approx 2^{50}$ ADDs for $p \approx 2^{50}$,

$\approx 2^{100}$ ADDs for $p \approx 2^{100}$,

(Not exactly linearly: cost of ADDs grows with p. But this is a minor effect.)

Understanding brute force

Can compute successively
$1P = (101384, 614510)$,
$2P = (102361, 628914)$,
$3P = (77571, 87643)$,
$4P = (650289, 31313)$,
$500001P = -P$.
$500002P = \infty$.

At some point we'll find n with $nP = (670366, 740819)$.

Maximum cost of computation:
≤ 500001 additions of P;
≤ 500001 nanoseconds on a CPU that does 1 ADD/nanosecond.

This is negligible work for $p \approx 2^{20}$.

But users can standardize a larger p, making the attack slower.

Attack cost scales linearly:
$\approx 2^{50}$ ADDs for $p \approx 2^{50}$,
$\approx 2^{100}$ ADDs for $p \approx 2^{100}$, etc.

(Not exactly linearly: cost of ADDs grows with p. But this is a minor effect.)

anding brute force

npute successively

01384, 614510),

02361, 628914),

7571, 87643),

50289, 31313),

$P = -P$.

$P = \infty$.

point we'll find n

$P = (670366, 740819)$.

m cost of computation:

01 additions of P;

01 nanoseconds on a CPU

es 1 ADD/nanosecond.

This is negligible work
for $p \approx 2^{20}$.

But users can
standardize a larger p,
making the attack slower.

Attack cost scales linearly:
$\approx 2^{50}$ ADDs for $p \approx 2^{50}$,
$\approx 2^{100}$ ADDs for $p \approx 2^{100}$, etc.

(Not exactly linearly:
cost of ADDs grows with p.
But this is a minor effect.)

Computa

of finishi

Chance

1/2 char

1/10 cha

"So user

ute force

cessively
4510),
8914),
43),
313),

ll find n
66, 740819).

computation:
ns of P;
conds on a CPU
/nanosecond.

This is negligible work for $p \approx 2^{20}$.

But users can standardize a larger p, making the attack slower.

Attack cost scales linearly:
$\approx 2^{50}$ ADDs for $p \approx 2^{50}$,
$\approx 2^{100}$ ADDs for $p \approx 2^{100}$, etc.

(Not exactly linearly: cost of ADDs grows with p. But this is a minor effect.)

Computation has a
of finishing earlier.
Chance scales line
1/2 chance of 1/2
1/10 chance of 1/

"So users should c

).

ion:

CPU

nd.

This is negligible work for $p \approx 2^{20}$.

But users can standardize a larger p, making the attack slower.

Attack cost scales linearly: $\approx 2^{50}$ ADDs for $p \approx 2^{50}$, $\approx 2^{100}$ ADDs for $p \approx 2^{100}$, etc.

(Not exactly linearly: cost of ADDs grows with p. But this is a minor effect.)

Computation has a good ch of finishing earlier.
Chance scales linearly:
$1/2$ chance of $1/2$ cost;
$1/10$ chance of $1/10$ cost; e

"So users should choose larg

This is negligible work
for $p \approx 2^{20}$.

But users can
standardize a larger p,
making the attack slower.

Attack cost scales linearly:
$\approx 2^{50}$ ADDs for $p \approx 2^{50}$,
$\approx 2^{100}$ ADDs for $p \approx 2^{100}$, etc.

(Not exactly linearly:
cost of ADDs grows with p.
But this is a minor effect.)

Computation has a good chance
of finishing earlier.
Chance scales linearly:
$1/2$ chance of $1/2$ cost;
$1/10$ chance of $1/10$ cost; etc.

"So users should choose large n."

This is negligible work for $p \approx 2^{20}$.

But users can standardize a larger p, making the attack slower.

Attack cost scales linearly: $\approx 2^{50}$ ADDs for $p \approx 2^{50}$, $\approx 2^{100}$ ADDs for $p \approx 2^{100}$, etc.

(Not exactly linearly: cost of ADDs grows with p. But this is a minor effect.)

Computation has a good chance of finishing earlier.
Chance scales linearly:
$1/2$ chance of $1/2$ cost;
$1/10$ chance of $1/10$ cost; etc.

"So users should choose large n."

That's pointless. We can apply "random self-reduction":
choose random r, say 69961;
compute $rP = (593450, 987590)$;
compute $(r + n)P$ as
$(593450, 987590) + (670366, 740819)$;
compute discrete log;
subtract r mod 500002; obtain n.

negligible work
2^{20}.

rs can
lize a larger p,
the attack slower.

cost scales linearly:
DDs for $p \approx 2^{50}$,
ADDs for $p \approx 2^{100}$, etc.

actly linearly:
ADDs grows with p.
is a minor effect.)

Computation has a good chance
of finishing earlier.
Chance scales linearly:
$1/2$ chance of $1/2$ cost;
$1/10$ chance of $1/10$ cost; etc.

"So users should choose large n."

That's pointless. We can apply
"random self-reduction":
choose random r, say 69961;
compute $rP = (593450, 987590)$;
compute $(r + n)P$ as
$(593450, 987590)+(670366, 740819)$;
compute discrete log;
subtract r mod 500002; obtain n.

Comput

One low
many pa
Example
2^{10} core
each 2^{30}
Maybe;
for detai

Attacker
many pa
Example
so 2^{34} c
so 2^{64} A
so 2^{89} A

work

er p,
slower.

linearly:
$\approx 2^{50}$,
$p \approx 2^{100}$, etc.

rly:
ws with p.
r effect.)

Computation has a good chance of finishing earlier.
Chance scales linearly:
1/2 chance of 1/2 cost;
1/10 chance of 1/10 cost; etc.

"So users should choose large n."

That's pointless. We can apply "random self-reduction":
choose random r, say 69961;
compute $rP = (593450, 987590)$;
compute $(r + n)P$ as
$(593450, 987590)+(670366, 740819)$;
compute discrete log;
subtract r mod 500002; obtain n.

Computation can

One low-cost chip
many parallel searc
Example, 2^6 €: or
2^{10} cores on the c
each 2^{30} ADDs/se
Maybe; see SHAR
for detailed cost a

Attacker can run
many parallel chip
Example, 2^{30} €: 2
so 2^{34} cores,
so 2^{64} ADDs/seco
so 2^{89} ADDs/year

etc.

Computation has a good chance of finishing earlier.
Chance scales linearly:
1/2 chance of 1/2 cost;
1/10 chance of 1/10 cost; etc.

"So users should choose large n."

That's pointless. We can apply "random self-reduction":
choose random r, say 69961;
compute $rP = (593450, 987590)$;
compute $(r + n)P$ as
$(593450, 987590)+(670366, 740819)$;
compute discrete log;
subtract r mod 500002; obtain n.

Computation can be parallel

One low-cost chip can run many parallel searches.
Example, 2^6 €: one chip,
2^{10} cores on the chip,
each 2^{30} ADDs/second?
Maybe; see SHARCS worksh
for detailed cost analyses.

Attacker can run many parallel chips.
Example, 2^{30} €: 2^{24} chips,
so 2^{34} cores,
so 2^{64} ADDs/second,
so 2^{89} ADDs/year.

Computation has a good chance of finishing earlier.
Chance scales linearly:
1/2 chance of 1/2 cost;
1/10 chance of 1/10 cost; etc.

"So users should choose large n."

That's pointless. We can apply "random self-reduction":
choose random r, say 69961;
compute $rP = (593450, 987590)$;
compute $(r + n)P$ as $(593450, 987590) + (670366, 740819)$;
compute discrete log;
subtract r mod 500002; obtain n.

Computation can be parallelized.

One low-cost chip can run many parallel searches.
Example, 2^6 €: one chip, 2^{10} cores on the chip, each 2^{30} ADDs/second?
Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.
Example, 2^{30} €: 2^{24} chips,
so 2^{34} cores,
so 2^{64} ADDs/second,
so 2^{89} ADDs/year.

ation has a good chance
ing earlier.
scales linearly:
nce of $1/2$ cost;
ance of $1/10$ cost; etc.

rs should choose large n."

pointless. We can apply
n self-reduction":
random r, say 69961;
e $rP = (593450, 987590)$;
e $(r + n)P$ as
$, 987590)+(670366, 740819)$;
e discrete log;
r mod 500002; obtain n.

Computation can be parallelized.

One low-cost chip can run
many parallel searches.
Example, 2^6 €: one chip,
2^{10} cores on the chip,
each 2^{30} ADDs/second?
Maybe; see SHARCS workshops
for detailed cost analyses.

Attacker can run
many parallel chips.
Example, 2^{30} €: 2^{24} chips,
so 2^{34} cores,
so 2^{64} ADDs/second,
so 2^{89} ADDs/year.

Multiple

Comput
to many

Given 10
n_2P, ..
Can find
with $\leq$ 5

Simplest
a sorted
n_1P, ..
Then ch
$1P$, $2P$,

a good chance

arly:
2 cost;
/10 cost; etc.

choose large n."

We can apply
ction":
say 69961;
93450, 987590);
P as
+(670366, 740819);
og;
00002; obtain n.

Computation can be parallelized.

One low-cost chip can run many parallel searches.
Example, 2^6 €: one chip, 2^{10} cores on the chip, each 2^{30} ADDs/second?
Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.
Example, 2^{30} €: 2^{24} chips, so 2^{34} cores, so 2^{64} ADDs/second, so 2^{89} ADDs/year.

Multiple targets ar

Computation can
to many targets at

Given 100 DL targ
$n_2P, \ldots, n_{100}P$:
Can find *all* of n_1
with $\leq$ 500002 AD

Simplest approach
a sorted table cont
$n_1P, \ldots, n_{100}P$.
Then check table
$1P$, $2P$, etc.

ance

tc.

ge n."

pply

.;

7590);

740819);

ain n.

Computation can be parallelized.

One low-cost chip can run many parallel searches.
Example, 2^6 €: one chip, 2^{10} cores on the chip, each 2^{30} ADDs/second?
Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.
Example, 2^{30} €: 2^{24} chips, so 2^{34} cores, so 2^{64} ADDs/second, so 2^{89} ADDs/year.

Multiple targets and giant st

Computation can be applied to many targets at once.

Given 100 DL targets n_1P, n_2P, . . . , $n_{100}P$:
Can find *all* of $n_1, n_2, \ldots, n$
with ≤ 500002 ADDs.

Simplest approach: First bu a sorted table containing n_1P, . . . , $n_{100}P$.
Then check table for $1P$, $2P$, etc.

Computation can be parallelized.

One low-cost chip can run many parallel searches.
Example, 2^6 €: one chip, 2^{10} cores on the chip, each 2^{30} ADDs/second?
Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips.
Example, 2^{30} €: 2^{24} chips, so 2^{34} cores, so 2^{64} ADDs/second, so 2^{89} ADDs/year.

Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets n_1P, n_2P, ..., $n_{100}P$:
Can find *all* of $n_1, n_2, \ldots, n_{100}$ with ≤ 500002 ADDs.

Simplest approach: First build a sorted table containing n_1P, ..., $n_{100}P$.
Then check table for $1P$, $2P$, etc.

ation can be parallelized.

-cost chip can run
arallel searches.
e, 2^6 €: one chip,
s on the chip,
ADDs/second?
see SHARCS workshops
led cost analyses.

r can run
arallel chips.
e, 2^{30} €: 2^{24} chips,
ores,
ADDs/second,
ADDs/year.

Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets n_1P, n_2P, …, $n_{100}P$:
Can find *all* of $n_1, n_2, \ldots, n_{100}$ with ≤ 500002 ADDs.

Simplest approach: First build a sorted table containing n_1P, …, $n_{100}P$.
Then check table for $1P$, $2P$, etc.

Interesti
Solving
isn't mu
solving o

Interesti
Solving
out of 1
is much
solving o

When di
find its

be parallelized.

can run
ches.
ne chip,
hip,
econd?
CS workshops
nalyses.

s.
2^{24} chips,

ond,
.

Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets n_1P, n_2P, ..., $n_{100}P$:
Can find *all* of $n_1, n_2, \ldots, n_{100}$ with ≤ 500002 ADDs.

Simplest approach: First build a sorted table containing n_1P, ..., $n_{100}P$.
Then check table for $1P$, $2P$, etc.

Interesting conseq
Solving all 100 DL
isn't much harder
solving one DL pro

Interesting conseq
Solving *at least on*
out of 100 DL pro
is much easier tha
solving one DL pro

When did this con
find its *first* n_i?

ized.

hops

Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets n_1P, n_2P, . . . , $n_{100}P$:
Can find *all* of $n_1, n_2, \ldots, n_{100}$ with ≤ 500002 ADDs.

Simplest approach: First build a sorted table containing n_1P, . . . , $n_{100}P$.
Then check table for $1P$, $2P$, etc.

Interesting consequence #1:
Solving all 100 DL problems isn't much harder than solving one DL problem.

Interesting consequence #2:
Solving *at least one* out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its *first* n_i?

Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets n_1P, n_2P, ..., $n_{100}P$:
Can find *all* of $n_1, n_2, \ldots, n_{100}$ with ≤ 500002 ADDs.

Simplest approach: First build a sorted table containing n_1P, ..., $n_{100}P$.
Then check table for $1P$, $2P$, etc.

Interesting consequence #1:
Solving all 100 DL problems isn't much harder than solving one DL problem.

Interesting consequence #2:
Solving *at least one* out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its *first* n_i?

Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets n_1P, n_2P, $\ldots$, $n_{100}P$: Can find *all* of $n_1, n_2, \ldots, n_{100}$ with ≤ 500002 ADDs.

Simplest approach: First build a sorted table containing n_1P, $\ldots$, $n_{100}P$. Then check table for $1P$, $2P$, etc.

Interesting consequence #1: Solving all 100 DL problems isn't much harder than solving one DL problem.

Interesting consequence #2: Solving *at least one* out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its *first* n_i? Typically $\approx 500002/100$ mults.

targets and giant steps

ation can be applied
targets at once.

00 DL targets $n_1 P$,
. . , $n_{100} P$:
all of $n_1, n_2, \ldots, n_{100}$
500002 ADDs.

approach: First build
table containing
. . , $n_{100} P$.
eck table for
etc.

Interesting consequence #1:
Solving all 100 DL problems
isn't much harder than
solving one DL problem.

Interesting consequence #2:
Solving *at least one*
out of 100 DL problems
is much easier than
solving one DL problem.

When did this computation
find its *first* n_i?
Typically $\approx 500002/100$ mults.

Can use
to turn
into mul
Let ℓ be

Given n
Choose
Comput
$r_2 P + n$

Solve th
Typically
to find a
$r_i + n$ n
immedia

nd giant steps

be applied
t once.

gets $n_1 P$,

, $n_2, \ldots, n_{100}$
Ds.

: First build
taining

for

Interesting consequence #1:
Solving all 100 DL problems
isn't much harder than
solving one DL problem.

Interesting consequence #2:
Solving *at least one*
out of 100 DL problems
is much easier than
solving one DL problem.

When did this computation
find its *first* n_i?
Typically $\approx 500002/100$ mults.

Can use random s
to turn a single ta
into multiple targe
Let ℓ be the order

Given nP:
Choose random r_1
Compute $r_1 P + n$
$r_2 P + nP$, etc.

Solve these 100 D
Typically $\approx \ell/100$
to find *at least on*
$r_i + n \bmod \ell$,
immediately revea

teps

/100

ild

Interesting consequence #1:
Solving all 100 DL problems
isn't much harder than
solving one DL problem.

Interesting consequence #2:
Solving *at least one*
out of 100 DL problems
is much easier than
solving one DL problem.

When did this computation
find its *first* n_i?
Typically $\approx 500002/100$ mults.

Can use random self-reducti
to turn a single target
into multiple targets.
Let ℓ be the order of P.

Given nP:
Choose random $r_1, r_2, \ldots, r$
Compute $r_1 P + nP$,
$r_2 P + nP$, etc.

Solve these 100 DL problem
Typically $\approx \ell/100$ mults
to find *at least one*
$r_i + n \bmod \ell$,
immediately revealing n.

Interesting consequence #1:
Solving all 100 DL problems
isn't much harder than
solving one DL problem.

Interesting consequence #2:
Solving *at least one*
out of 100 DL problems
is much easier than
solving one DL problem.

When did this computation
find its *first* n_i?
Typically $\approx 500002/100$ mults.

Can use random self-reduction
to turn a single target
into multiple targets.
Let ℓ be the order of P.

Given nP:
Choose random $r_1, r_2, \ldots, r_{100}$.
Compute $r_1 P + nP$,
$r_2 P + nP$, etc.

Solve these 100 DL problems.
Typically $\approx \ell/100$ mults
to find *at least one*
$r_i + n \bmod \ell$,
immediately revealing n.

ng consequence #1:
all 100 DL problems
ch harder than
one DL problem.

ng consequence #2:
at least one
00 DL problems
easier than
one DL problem.

'd this computation
first n_i?
y $\approx 500002/100$ mults.

Can use random self-reduction
to turn a single target
into multiple targets.
Let ℓ be the order of P.

Given nP:
Choose random $r_1, r_2, \ldots, r_{100}$.
Compute $r_1 P + nP$,
$r_2 P + nP$, etc.

Solve these 100 DL problems.
Typically $\approx \ell/100$ mults
to find *at least one*
$r_i + n \bmod \ell$,
immediately revealing n.

Also spe
to comp
$\approx \lg p$ A

Faster:
with r_1
Comput
$r_1 P + n$
$2r_1 P +$
$3r_1 P +$
Just 1 A

$\approx 100 +$
to find r

uence #1:
. problems
than
oblem.

uence #2:
ne
blems
n
oblem.

nputation

$2/100$ mults.

Can use random self-reduction
to turn a single target
into multiple targets.
Let ℓ be the order of P.

Given nP:
Choose random $r_1, r_2, \ldots, r_{100}$.
Compute $r_1 P + nP$,
$r_2 P + nP$, etc.

Solve these 100 DL problems.
Typically $\approx \ell/100$ mults
to find *at least one*
$r_i + n \bmod \ell$,
immediately revealing n.

Also spent some A
to compute each r
$\approx \lg p$ ADDs for e

Faster: Choose r_i
with $r_1 \approx \ell/100$.
Compute $r_1 P$;
$r_1 P + nP$;
$2r_1 P + nP$;
$3r_1 P + nP$; etc.
Just 1 ADD for ea

$\approx 100 + \lg \ell + \ell/1$
to find n given nP

lts.

Can use random self-reduction to turn a single target into multiple targets.
Let ℓ be the order of P.

Given nP:
Choose random $r_1, r_2, \ldots, r_{100}$.
Compute $r_1P + nP$, $r_2P + nP$, etc.

Solve these 100 DL problems.
Typically $\approx \ell/100$ mults to find *at least one* $r_i + n \bmod \ell$, immediately revealing n.

Also spent some ADDs to compute each r_iP: $\approx \lg p$ ADDs for each i.

Faster: Choose $r_i = ir_1$ with $r_1 \approx \ell/100$.
Compute r_1P;
$r_1P + nP$;
$2r_1P + nP$;
$3r_1P + nP$; etc.
Just 1 ADD for each new i.

$\approx 100 + \lg \ell + \ell/100$ ADDs to find n given nP.

Can use random self-reduction
to turn a single target
into multiple targets.
Let ℓ be the order of P.

Given nP:
Choose random $r_1, r_2, \ldots, r_{100}$.
Compute $r_1 P + nP$,
$r_2 P + nP$, etc.

Solve these 100 DL problems.
Typically $\approx \ell/100$ mults
to find *at least one*
$r_i + n \bmod \ell$,
immediately revealing n.

Also spent some ADDs
to compute each $r_i P$:
$\approx \lg p$ ADDs for each i.

Faster: Choose $r_i = i r_1$
with $r_1 \approx \ell/100$.
Compute $r_1 P$;
$r_1 P + nP$;
$2r_1 P + nP$;
$3r_1 P + nP$; etc.
Just 1 ADD for each new i.

$\approx 100 + \lg \ell + \ell/100$ ADDs
to find n given nP.

random self-reduction
a single target
tiple targets.
the order of P.

P:
random $r_1, r_2, \ldots, r_{100}$.
e $r_1 P + nP$,
P, etc.

ese 100 DL problems.
y $\approx \ell/100$ mults
at least one
nod ℓ,
tely revealing n.

Also spent some ADDs
to compute each $r_i P$:
$\approx \lg p$ ADDs for each i.

Faster: Choose $r_i = i r_1$
with $r_1 \approx \ell/100$.
Compute $r_1 P$;
$r_1 P + nP$;
$2r_1 P + nP$;
$3r_1 P + nP$; etc.
Just 1 ADD for each new i.

$\approx 100 + \lg \ell + \ell/100$ ADDs
to find n given nP.

Faster:
Only $\approx$
to solve
"Shanks
discrete-

Example
500002,
$Q = nP$
Compute
Then co
$708P +$
$2 \cdot 708P$
$3 \cdot 708P$
$\ldots$, 706
(534170

elf-reduction
rget
ets.
of P.

$_1, r_2, \ldots, r_{100}$.
P,

L problems.
mults
e

ling n.

Also spent some ADDs
to compute each $r_i P$:
$\approx \lg p$ ADDs for each i.

Faster: Choose $r_i = i r_1$
with $r_1 \approx \ell/100$.
Compute $r_1 P$;
$r_1 P + nP$;
$2r_1 P + nP$;
$3r_1 P + nP$; etc.
Just 1 ADD for each new i.

$\approx 100 + \lg \ell + \ell/100$ ADDs
to find n given nP.

Faster: Increase 10
Only $\approx 2\sqrt{\ell}$ ADD
to solve one DL pr
"Shanks baby-step
discrete-logarithm

Example: p = 10
500002, $P = (101$
$Q = nP = (67036$
Compute $708P$=(
Then compute 707
$708P + Q = (3428$
$2 \cdot 708P + nP = (4$
$3 \cdot 708P + nP = (4$
$\ldots, 706 \cdot 708P +$
(534170, 450849).

on

$/100$.

s.

Also spent some ADDs
to compute each $r_i P$:
$\approx \lg p$ ADDs for each i.

Faster: Choose $r_i = i r_1$
with $r_1 \approx \ell/100$.
Compute $r_1 P$;
$r_1 P + nP$;
$2r_1 P + nP$;
$3r_1 P + nP$; etc.
Just 1 ADD for each new i.

$\approx 100 + \lg \ell + \ell/100$ ADDs
to find n given nP.

Faster: Increase 100 to $\approx \sqrt{}$
Only $\approx 2\sqrt{\ell}$ ADDs
to solve one DL problem!
"Shanks baby-step-giant-ste
discrete-logarithm algorithm

Example: $p = 1000003, \ell$
500002, $P = (101384, 61451$
$Q = nP = (670366, 740819)$
Compute $708P$=(393230, 42
Then compute 707 targets:
$708P + Q = (342867, 15381$
$2 \cdot 708P + nP = (430321, 99$
$3 \cdot 708P + nP = (423151, 63$
$\ldots, 706 \cdot 708P + nP =$
(534170, 450849).

Also spent some ADDs to compute each $r_i P$: $\approx \lg p$ ADDs for each i.

Faster: Choose $r_i = i r_1$ with $r_1 \approx \ell/100$.
Compute $r_1 P$;
$r_1 P + nP$;
$2r_1 P + nP$;
$3r_1 P + nP$; etc.
Just 1 ADD for each new i.

$\approx 100 + \lg \ell + \ell/100$ ADDs to find n given nP.

Faster: Increase 100 to $\approx \sqrt{\ell}$.
Only $\approx 2\sqrt{\ell}$ ADDs to solve one DL problem!
"Shanks baby-step-giant-step discrete-logarithm algorithm."

Example: $p = 1000003, \ell = 500002$, $P = (101384, 614510)$, $Q = nP = (670366, 740819)$.
Compute $708P$=$(393230, 421116)$.
Then compute 707 targets:
$708P + Q = (342867, 153817)$,
$2 \cdot 708P + nP = (430321, 994742)$,
$3 \cdot 708P + nP = (423151, 635197)$,
$\ldots, 706 \cdot 708P + nP = (534170, 450849)$.

ent some ADDs
ute each $r_i P$:
DDs for each i.

Choose $r_i = i r_1$
$\approx \ell/100$.
e $r_1 P$;
P;
nP;
nP; etc.
DD for each new i.

$\lg \ell + \ell/100$ ADDs
n given nP.

Faster: Increase 100 to $\approx \sqrt{\ell}$.
Only $\approx 2\sqrt{\ell}$ ADDs
to solve one DL problem!
"Shanks baby-step-giant-step
discrete-logarithm algorithm."

Example: $p = 1000003, \ell = 500002$, $P = (101384, 614510)$, $Q = nP = (670366, 740819)$.
Compute $708P = (393230, 421116)$.
Then compute 707 targets:
$708P + Q = (342867, 153817)$,
$2 \cdot 708P + nP = (430321, 994742)$,
$3 \cdot 708P + nP = (423151, 635197)$,
$\ldots$, $706 \cdot 708P + nP = (534170, 450849)$.

Build a s
600·708
$27 \cdot 708P$
219·708
...
242·708
...
317·708

Look up

$620P =$
596·708
in the ta
so 620 =
deduce

ADDs

$r_i P$:

ach i.

$= i r_1$

ach new i.

00 ADDs

P.

Faster: Increase 100 to $\approx \sqrt{\ell}$.
Only $\approx 2\sqrt{\ell}$ ADDs
to solve one DL problem!
"Shanks baby-step-giant-step
discrete-logarithm algorithm."

Example: $p = 1000003, \ell =$
500002, $P = (101384, 614510)$,
$Q = nP = (670366, 740819)$.
Compute $708P = (393230, 421116)$.
Then compute 707 targets:
$708P + Q = (342867, 153817)$,
$2 \cdot 708P + nP = (430321, 994742)$,
$3 \cdot 708P + nP = (423151, 635197)$,
$\ldots$, $706 \cdot 708P + nP =$
$(534170, 450849)$.

Build a sorted tab
$600 \cdot 708P + Q = ($
$27 \cdot 708P + Q = ($
$219 \cdot 708P + Q = ($
...
$242 \cdot 708P + Q = ($
...
$317 \cdot 708P + Q = ($

Look up P, $2P$, 3

$620P = (950652,$
$596 \cdot 708P + Q = ($
in the table of tar
so $620 = 596 \cdot 708$
deduce $n = 7865$

Faster: Increase 100 to $\approx \sqrt{\ell}$.
Only $\approx 2\sqrt{\ell}$ ADDs
to solve one DL problem!
"Shanks baby-step-giant-step
discrete-logarithm algorithm."

Example: $p = 1000003, \ell = 500002$, $P = (101384, 614510)$, $Q = nP = (670366, 740819)$.
Compute $708P{=}(393230, 421116)$.
Then compute 707 targets:
$708P + Q = (342867, 153817)$,
$2 \cdot 708P + nP = (430321, 994742)$,
$3 \cdot 708P + nP = (423151, 635197)$,
$\ldots$, $706 \cdot 708P + nP = (534170, 450849)$.

Build a sorted table of targe
$600{\cdot}708P{+}Q = (799978, 92$
$27 \cdot 708P + Q = (785344, 83$
$219{\cdot}708P{+}Q = (425475, 79$
...
$242{\cdot}708P{+}Q = (262804, 34$
...
$317{\cdot}708P{+}Q = (599785, 18$

Look up P, $2P$, $3P$, etc. in

$620P = (950652, 688508)$; f
$596 \cdot 708P + Q = (950652, 68$
in the table of targets;
so $620 = 596{\cdot}708{+}n \bmod 5$
deduce $n = 78654$.

Faster: Increase 100 to $\approx \sqrt{\ell}$.
Only $\approx 2\sqrt{\ell}$ ADDs
to solve one DL problem!
"Shanks baby-step-giant-step
discrete-logarithm algorithm."

Example: $p = 1000003, \ell = 500002$, $P = (101384, 614510)$, $Q = nP = (670366, 740819)$.
Compute $708P = (393230, 421116)$.
Then compute 707 targets:
$708P + Q = (342867, 153817)$,
$2 \cdot 708P + nP = (430321, 994742)$,
$3 \cdot 708P + nP = (423151, 635197)$,
$\ldots$, $706 \cdot 708P + nP = (534170, 450849)$.

Build a sorted table of targets:
$600 \cdot 708P + Q = (799978, 929249)$,
$27 \cdot 708P + Q = (785344, 831127)$,
$219 \cdot 708P + Q = (425475, 793466)$,
...
$242 \cdot 708P + Q = (262804, 347755)$,
...
$317 \cdot 708P + Q = (599785, 189116)$.

Look up P, $2P$, $3P$, etc. in table.

$620P = (950652, 688508)$; find
$596 \cdot 708P + Q = (950652, 688508)$
in the table of targets;
so $620 = 596 \cdot 708 + n \bmod 500002$;
deduce $n = 78654$.

Increase 100 to $\approx \sqrt{\ell}$.
$2\sqrt{\ell}$ ADDs
one DL problem!
baby-step-giant-step
-logarithm algorithm."

e: $p = 1000003, \ell =$
$P = (101384, 614510)$,
$= (670366, 740819)$.
e $708P = (393230, 421116)$.
mpute 707 targets:
$Q = (342867, 153817)$,
$+ nP = (430321, 994742)$,
$+ nP = (423151, 635197)$,
$\cdot 708P + nP =$
$, 450849)$.

Build a sorted table of targets:
$600 \cdot 708P + Q = (799978, 929249)$,
$27 \cdot 708P + Q = (785344, 831127)$,
$219 \cdot 708P + Q = (425475, 793466)$,
...
$242 \cdot 708P + Q = (262804, 347755)$,
...
$317 \cdot 708P + Q = (599785, 189116)$.

Look up P, $2P$, $3P$, etc. in table.

$620P = (950652, 688508)$; find
$596 \cdot 708P + Q = (950652, 688508)$
in the table of targets;
so $620 = 596 \cdot 708 + n \bmod 500002$;
deduce $n = 78654$.

Factors

P has o

Given Q

$R = (53$
$S = (53$
Comput

$R = (2$
and
$S = (2 \cdot$
Comput
$n_2 = \log$
This is a
of size 5

00 to $\approx \sqrt{\ell}$.
s
roblem!
o-giant-step
algorithm."

00003, ℓ =
384, 614510),
6, 740819).
393230, 421116).
7 targets:
867, 153817),
430321, 994742),
423151, 635197),
– nP =

Build a sorted table of targets:
$600 \cdot 708P + Q = (799978, 929249)$,
$27 \cdot 708P + Q = (785344, 831127)$,
$219 \cdot 708P + Q = (425475, 793466)$,
...
$242 \cdot 708P + Q = (262804, 347755)$,
...
$317 \cdot 708P + Q = (599785, 189116)$.

Look up P, $2P$, $3P$, etc. in table.

$620P = (950652, 688508)$; find
$596 \cdot 708P + Q = (950652, 688508)$
in the table of targets;
so $620 = 596 \cdot 708 + n \bmod 500002$;
deduce $n = 78654$.

Factors of the gro

P has order $2 \cdot 53^2$

Given $Q = nP$, fin

$R = (53^2 \cdot 89)P$ h
$S = (53^2 \cdot 89)Q$ is
Compute $n_1 = \log$

$R = (2 \cdot 53 \cdot 89)P$
and
$S = (2 \cdot 53 \cdot 89)Q$
Compute
$n_2 = \log_R S \equiv n$ m
This is a DLP in a
of size 53.

$\sqrt{\ell}$.

p

.”

=

10),

).

21116).

7),

4742),

5197),

Build a sorted table of targets:

$600 \cdot 708P + Q = (799978, 929249)$,

$27 \cdot 708P + Q = (785344, 831127)$,

$219 \cdot 708P + Q = (425475, 793466)$,

...

$242 \cdot 708P + Q = (262804, 347755)$,

...

$317 \cdot 708P + Q = (599785, 189116)$.

Look up P, $2P$, $3P$, etc. in table.

$620P = (950652, 688508)$; find $596 \cdot 708P + Q = (950652, 688508)$ in the table of targets; so $620 = 596 \cdot 708 + n \bmod 500002$; deduce $n = 78654$.

Factors of the group order

P has order $2 \cdot 53^2 \cdot 89$.

Given $Q = nP$, find $n = \log$

$R = (53^2 \cdot 89)P$ has order 2

$S = (53^2 \cdot 89)Q$ is multiple

Compute $n_1 = \log_R S \equiv n$

$R = (2 \cdot 53 \cdot 89)P$ has orde

and

$S = (2 \cdot 53 \cdot 89)Q$ is multipl

Compute

$n_2 = \log_R S \equiv n \bmod 53$.

This is a DLP in a group of size 53.

Build a sorted table of targets:

$600 \cdot 708P + Q = (799978, 929249)$,

$27 \cdot 708P + Q = (785344, 831127)$,

$219 \cdot 708P + Q = (425475, 793466)$,

...

$242 \cdot 708P + Q = (262804, 347755)$,

...

$317 \cdot 708P + Q = (599785, 189116)$.

Look up P, $2P$, $3P$, etc. in table.

$620P = (950652, 688508)$; find $596 \cdot 708P + Q = (950652, 688508)$ in the table of targets; so $620 = 596 \cdot 708 + n \bmod 500002$; deduce $n = 78654$.

<u>Factors of the group order</u>

P has order $2 \cdot 53^2 \cdot 89$.

Given $Q = nP$, find $n = \log_P Q$:

$R = (53^2 \cdot 89)P$ has order 2, and $S = (53^2 \cdot 89)Q$ is multiple of R. Compute $n_1 = \log_R S \equiv n \bmod 2$.

$R = (2 \cdot 53 \cdot 89)P$ has order 53, and $S = (2 \cdot 53 \cdot 89)Q$ is multiple of R. Compute $n_2 = \log_R S \equiv n \bmod 53$. This is a DLP in a group of size 53.

sorted table of targets:

$P+Q = (799978, 929249)$,

$P+Q = (785344, 831127)$,

$P+Q = (425475, 793466)$,

$P+Q = (262804, 347755)$,

$P+Q = (599785, 189116)$.

P, $2P$, $3P$, etc. in table.

$(950652, 688508)$; find

$P+Q = (950652, 688508)$

able of targets;

$= 596{\cdot}708+n$ mod 500002;

$n = 78654$.

Factors of the group order

P has order $2 \cdot 53^2 \cdot 89$.

Given $Q = nP$, find $n = \log_P Q$:

$R = (53^2 \cdot 89)P$ has order 2, and $S = (53^2 \cdot 89)Q$ is multiple of R.
Compute $n_1 = \log_R S \equiv n \bmod 2$.

$R = (2 \cdot 53 \cdot 89)P$ has order 53, and
$S = (2 \cdot 53 \cdot 89)Q$ is multiple of R.
Compute
$n_2 = \log_R S \equiv n \bmod 53$.
This is a DLP in a group of size 53.

$T = (2$

a multip

Comput

$n_3 = \log$

Now n_2

$R = (2$

$S = (2$

Comput

$n_4 = \log$

Use Chi

$n \equiv n_1$

$n \equiv n_2$

$n \equiv n_4$

to deter

le of targets:

799978, 929249),

785344, 831127),

425475, 793466),

262804, 347755),

599785, 189116).

P, etc. in table.

688508); find

950652, 688508)

gets;

$+n$ mod 500002;

-.

Factors of the group order

P has order $2 \cdot 53^2 \cdot 89$.

Given $Q = nP$, find $n = \log_P Q$:

$R = (53^2 \cdot 89)P$ has order 2, and
$S = (53^2 \cdot 89)Q$ is multiple of R.
Compute $n_1 = \log_R S \equiv n \bmod 2$.

$R = (2 \cdot 53 \cdot 89)P$ has order 53,
and
$S = (2 \cdot 53 \cdot 89)Q$ is multiple of R.
Compute
$n_2 = \log_R S \equiv n \bmod 53$.
This is a DLP in a group
of size 53.

$T = (2 \cdot 89)(Q -$
a multiple of R, i.
Compute
$n_3 = \log_R T \equiv n$
Now $n_2 + 53n_3 \equiv$

$R = (2 \cdot 53^2)P$ ha
$S = (2 \cdot 53^2)Q$ is
Compute
$n_4 = \log_R S \equiv n$

Use Chinese Rema
$n \equiv n_1 \bmod 2$,
$n \equiv n_2 + 53n_3$ m
$n \equiv n_4 \bmod 89$,
to determine n m

ts:
29249),
1127),
3466),
7755),
9116).
table.
ind
38508)
00002;

<u>Factors of the group order</u>

P has order $2 \cdot 53^2 \cdot 89$.

Given $Q = nP$, find $n = \log_P Q$:

$R = (53^2 \cdot 89)P$ has order 2, and
$S = (53^2 \cdot 89)Q$ is multiple of R.
Compute $n_1 = \log_R S \equiv n \bmod 2$.

$R = (2 \cdot 53 \cdot 89)P$ has order 53,
and
$S = (2 \cdot 53 \cdot 89)Q$ is multiple of R.
Compute
$n_2 = \log_R S \equiv n \bmod 53$.
This is a DLP in a group
of size 53.

$T = (2 \cdot 89)(Q - n_2 P)$ is als
a multiple of R, i.e., has ord
Compute
$n_3 = \log_R T \equiv n \bmod 53$.
Now $n_2 + 53n_3 \equiv n \bmod 53$

$R = (2 \cdot 53^2)P$ has order 89
$S = (2 \cdot 53^2)Q$ is multiple o
Compute
$n_4 = \log_R S \equiv n \bmod 89$.

Use Chinese Remainder The
$n \equiv n_1 \bmod 2$,
$n \equiv n_2 + 53n_3 \bmod 53^2$,
$n \equiv n_4 \bmod 89$,
to determine n modulo $2 \cdot 5$

Factors of the group order

P has order $2 \cdot 53^2 \cdot 89$.

Given $Q = nP$, find $n = \log_P Q$:

$R = (53^2 \cdot 89)P$ has order 2, and
$S = (53^2 \cdot 89)Q$ is multiple of R.
Compute $n_1 = \log_R S \equiv n \bmod 2$.

$R = (2 \cdot 53 \cdot 89)P$ has order 53,
and
$S = (2 \cdot 53 \cdot 89)Q$ is multiple of R.
Compute
$n_2 = \log_R S \equiv n \bmod 53$.
This is a DLP in a group
of size 53.

$T = (2 \cdot 89)(Q - n_2 P)$ is also
a multiple of R, i.e., has order 53.
Compute
$n_3 = \log_R T \equiv n \bmod 53$.
Now $n_2 + 53n_3 \equiv n \bmod 53^2$.

$R = (2 \cdot 53^2)P$ has order 89, and
$S = (2 \cdot 53^2)Q$ is multiple of R.
Compute
$n_4 = \log_R S \equiv n \bmod 89$.

Use Chinese Remainder Theorem
$n \equiv n_1 \bmod 2$,
$n \equiv n_2 + 53n_3 \bmod 53^2$,
$n \equiv n_4 \bmod 89$,
to determine n modulo $2 \cdot 53^2 \cdot 89$.

of the group order

rder $2 \cdot 53^2 \cdot 89$.

$= nP$, find $n = \log_P Q$:

$^2 \cdot 89)P$ has order 2, and

$^2 \cdot 89)Q$ is multiple of R.

e $n_1 = \log_R S \equiv n \bmod 2$.

$\cdot 53 \cdot 89)P$ has order 53,

$53 \cdot 89)Q$ is multiple of R.

e

$g_R S \equiv n \bmod 53$.

DLP in a group

3.

$T = (2 \cdot 89)(Q - n_2 P)$ is also a multiple of R, i.e., has order 53.

Compute

$n_3 = \log_R T \equiv n \bmod 53$.

Now $n_2 + 53n_3 \equiv n \bmod 53^2$.

$R = (2 \cdot 53^2)P$ has order 89, and $S = (2 \cdot 53^2)Q$ is multiple of R.

Compute

$n_4 = \log_R S \equiv n \bmod 89$.

Use Chinese Remainder Theorem

$n \equiv n_1 \bmod 2$,

$n \equiv n_2 + 53n_3 \bmod 53^2$,

$n \equiv n_4 \bmod 89$,

to determine n modulo $2 \cdot 53^2 \cdot 89$.

This "Po

converts

an order

and a fe

Here (53

$(53^2 \cdot 89$

$(2 \cdot 53 \cdot$

$(2 \cdot 53 \cdot$

A search

$(2 \cdot 89)($

and n_2

up order

$^2 \cdot 89$.

nd $n = \log_P Q$:

as order 2, and

s multiple of R.

$g_R S \equiv n \bmod 2$.

P has order 53,

is multiple of R.

mod 53.

n group

$T = (2 \cdot 89)(Q - n_2 P)$ is also a multiple of R, i.e., has order 53. Compute $n_3 = \log_R T \equiv n \bmod 53$. Now $n_2 + 53n_3 \equiv n \bmod 53^2$.

$R = (2 \cdot 53^2)P$ has order 89, and $S = (2 \cdot 53^2)Q$ is multiple of R. Compute $n_4 = \log_R S \equiv n \bmod 89$.

Use Chinese Remainder Theorem
$n \equiv n_1 \bmod 2$,
$n \equiv n_2 + 53n_3 \bmod 53^2$,
$n \equiv n_4 \bmod 89$,
to determine n modulo $2 \cdot 53^2 \cdot 89$.

This "Pohlig-Helln
converts an order-
an order-a DL, an
and a few scalar m

Here $(53^2 \cdot 89)P$
$(53^2 \cdot 89)Q = \infty$,

$(2 \cdot 53 \cdot 89)P = (5$
$(2 \cdot 53 \cdot 89)Q = (7$
A search quickly fi
$(2 \cdot 89)(Q - 2P) =$
and $n_2 + 53n_3 =$

$_P\, Q$:

, and

of R.

mod 2.

r 53,

e of R.

$T = (2 \cdot 89)(Q - n_2 P)$ is also a multiple of R, i.e., has order 53.
Compute
$n_3 = \log_R T \equiv n \bmod 53$.
Now $n_2 + 53 n_3 \equiv n \bmod 53^2$.

$R = (2 \cdot 53^2)P$ has order 89, and
$S = (2 \cdot 53^2)Q$ is multiple of R.
Compute
$n_4 = \log_R S \equiv n \bmod 89$.

Use Chinese Remainder Theorem
$n \equiv n_1 \bmod 2$,
$n \equiv n_2 + 53 n_3 \bmod 53^2$,
$n \equiv n_4 \bmod 89$,
to determine n modulo $2 \cdot 53^2 \cdot 89$.

This "Pohlig-Hellman metho
converts an order-ab DL int
an order-a DL, an order-b D
and a few scalar multiplicati

Here $(53^2 \cdot 89)P = (1, 0)$ a
$(53^2 \cdot 89)Q = \infty$, thus $n_1 =$

$(2 \cdot 53 \cdot 89)P = (539296, 48$
$(2 \cdot 53 \cdot 89)Q = (782288, 57$
A search quickly finds $n_2 =$
$(2 \cdot 89)(Q - 2P) = \infty$, thus
and $n_2 + 53 n_3 = 2$.

$T = (2 \cdot 89)(Q - n_2 P)$ is also a multiple of R, i.e., has order 53. Compute $n_3 = \log_R T \equiv n \bmod 53$. Now $n_2 + 53n_3 \equiv n \bmod 53^2$.

$R = (2 \cdot 53^2)P$ has order 89, and $S = (2 \cdot 53^2)Q$ is multiple of R. Compute $n_4 = \log_R S \equiv n \bmod 89$.

Use Chinese Remainder Theorem
$n \equiv n_1 \bmod 2$,
$n \equiv n_2 + 53n_3 \bmod 53^2$,
$n \equiv n_4 \bmod 89$,
to determine n modulo $2 \cdot 53^2 \cdot 89$.

This "Pohlig-Hellman method" converts an order-ab DL into an order-a DL, an order-b DL, and a few scalar multiplications.

Here $(53^2 \cdot 89)P = (1, 0)$ and $(53^2 \cdot 89)Q = \infty$, thus $n_1 = 0$.

$(2 \cdot 53 \cdot 89)P = (539296, 488875)$,
$(2 \cdot 53 \cdot 89)Q = (782288, 572333)$.
A search quickly finds $n_2 = 2$.
$(2 \cdot 89)(Q - 2P) = \infty$, thus $n_3 = 0$ and $n_2 + 53n_3 = 2$.

$89)(Q - n_2 P)$ is also
le of R, i.e., has order 53.
e
$_R T \equiv n$ mod 53.
$+ 53n_3 \equiv n$ mod 53^2.

$53^2)P$ has order 89, and
$53^2)Q$ is multiple of R.
e
$_R S \equiv n$ mod 89.

nese Remainder Theorem
mod 2,
$+ 53n_3$ mod 53^2,
mod 89,
mine n modulo $2 \cdot 53^2 \cdot 89$.

This "Pohlig-Hellman method" converts an order-ab DL into an order-a DL, an order-b DL, and a few scalar multiplications.

Here $(53^2 \cdot 89)P = (1, 0)$ and $(53^2 \cdot 89)Q = \infty$, thus $n_1 = 0$.

$(2 \cdot 53 \cdot 89)P = (539296, 488875)$,
$(2 \cdot 53 \cdot 89)Q = (782288, 572333)$.
A search quickly finds $n_2 = 2$.
$(2 \cdot 89)(Q - 2P) = \infty$, thus $n_3 = 0$ and $n_2 + 53n_3 = 2$.

$(2 \cdot 53^2)$
$(2 \cdot 53^2)$
Comput
e.g. usin

Use Chi
$n \equiv 0$ m
$n \equiv 2$ m
$n \equiv 67$
to deter

Pohlig-H
security
problem
to secur
subgrou

$n_2 P$) is also
e., has order 53.

mod 53.
$\equiv n$ mod 53^2.

s order 89, and
multiple of R.

mod 89.

inder Theorem

od 53^2,

odulo $2 \cdot 53^2 \cdot 89$.

This "Pohlig-Hellman method" converts an order-ab DL into an order-a DL, an order-b DL, and a few scalar multiplications.

Here $(53^2 \cdot 89)P = (1, 0)$ and $(53^2 \cdot 89)Q = \infty$, thus $n_1 = 0$.

$(2 \cdot 53 \cdot 89)P = (539296, 488875)$,
$(2 \cdot 53 \cdot 89)Q = (782288, 572333)$.
A search quickly finds $n_2 = 2$.
$(2 \cdot 89)(Q - 2P) = \infty$, thus $n_3 = 0$ and $n_2 + 53n_3 = 2$.

$(2 \cdot 53^2)P = (8775$
$(2 \cdot 53^2)Q = (822$
Compute $n_4 = 67$
e.g. using BSGS.

Use Chinese Rema
$n \equiv 0$ mod 2,
$n \equiv 2$ mod 53^2,
$n \equiv 67$ mod 89,
to determine $n =$

Pohlig-Hellman me
security of discrete
problem in group g
to security of large
subgroup.

so

ler 53.

3^2.

, and

f R.

orem

$3^2 \cdot 89$.

This "Pohlig-Hellman method" converts an order-ab DL into an order-a DL, an order-b DL, and a few scalar multiplications.

Here $(53^2 \cdot 89)P = (1, 0)$ and $(53^2 \cdot 89)Q = \infty$, thus $n_1 = 0$.

$(2 \cdot 53 \cdot 89)P = (539296, 488875)$,
$(2 \cdot 53 \cdot 89)Q = (782288, 572333)$.
A search quickly finds $n_2 = 2$.
$(2 \cdot 89)(Q - 2P) = \infty$, thus $n_3 = 0$ and $n_2 + 53n_3 = 2$.

$(2 \cdot 53^2)P = (877560, 94784$
$(2 \cdot 53^2)Q = (822491, 1182$
Compute $n_4 = 67$, e.g. using BSGS.

Use Chinese Remainder The
$n \equiv 0 \bmod 2$,
$n \equiv 2 \bmod 53^2$,
$n \equiv 67 \bmod 89$,
to determine $n = 78654$.

Pohlig-Hellman method redu
security of discrete logarithm
problem in group generated
to security of largest prime o
subgroup.

This "Pohlig-Hellman method" converts an order-ab DL into an order-a DL, an order-b DL, and a few scalar multiplications.

Here $(53^2 \cdot 89)P = (1, 0)$ and $(53^2 \cdot 89)Q = \infty$, thus $n_1 = 0$.

$(2 \cdot 53 \cdot 89)P = (539296, 488875)$,
$(2 \cdot 53 \cdot 89)Q = (782288, 572333)$.
A search quickly finds $n_2 = 2$.
$(2 \cdot 89)(Q - 2P) = \infty$, thus $n_3 = 0$ and $n_2 + 53n_3 = 2$.

$(2 \cdot 53^2)P = (877560, 947848)$ and $(2 \cdot 53^2)Q = (822491, 118220)$.
Compute $n_4 = 67$, e.g. using BSGS.

Use Chinese Remainder Theorem
$n \equiv 0 \bmod 2$,
$n \equiv 2 \bmod 53^2$,
$n \equiv 67 \bmod 89$,
to determine $n = 78654$.

Pohlig-Hellman method reduces security of discrete logarithm problem in group generated by P to security of largest prime order subgroup.

ohlig-Hellman method"
an order-ab DL into
-a DL, an order-b DL,
w scalar multiplications.

$3^2 \cdot 89)P = (1,0)$ and
$)Q = \infty$, thus $n_1 = 0$.

$89)P = (539296, 488875)$,
$89)Q = (782288, 572333)$.
quickly finds $n_2 = 2$.
$Q - 2P) = \infty$, thus $n_3 = 0$
$+ 53n_3 = 2$.

$(2 \cdot 53^2)P = (877560, 947848)$ and
$(2 \cdot 53^2)Q = (822491, 118220)$.
Compute $n_4 = 67$,
e.g. using BSGS.

Use Chinese Remainder Theorem
$n \equiv 0 \bmod 2$,
$n \equiv 2 \bmod 53^2$,
$n \equiv 67 \bmod 89$,
to determine $n = 78654$.

Pohlig-Hellman method reduces security of discrete logarithm problem in group generated by P to security of largest prime order subgroup.

The rho

Simplifie

Make a
in the gr
where th
on curre

Birthday
Random
elements
after abo

The wal
Cycle-fin
(e.g., Fl

man method"
ab DL into
order-b DL,
multiplications.

$= (1, 0)$ and
thus $n_1 = 0$.

$539296, 488875)$,
$782288, 572333)$.
inds $n_2 = 2$.
∞, thus $n_3 = 0$
2.

$(2 \cdot 53^2)P = (877560, 947848)$ and
$(2 \cdot 53^2)Q = (822491, 118220)$.
Compute $n_4 = 67$,
e.g. using BSGS.

Use Chinese Remainder Theorem
$n \equiv 0 \bmod 2$,
$n \equiv 2 \bmod 53^2$,
$n \equiv 67 \bmod 89$,
to determine $n = 78654$.

Pohlig-Hellman method reduces security of discrete logarithm problem in group generated by P to security of largest prime order subgroup.

The rho method

Simplified, non-pa

Make a pseudo-ra
in the group $\langle P \rangle$,
where the next ste
on current point:

Birthday paradox:
Randomly choosin
elements picks one
after about $\sqrt{\pi \ell /}$

The walk now ent
Cycle-finding algo
(e.g., Floyd) quick

od"
o
L,
ons.

and
= 0.

8875),
2333).
2.
$n_3 = 0$

$(2 \cdot 53^2)P = (877560, 947848)$ and
$(2 \cdot 53^2)Q = (822491, 118220)$.
Compute $n_4 = 67$,
e.g. using BSGS.

Use Chinese Remainder Theorem
$n \equiv 0 \bmod 2$,
$n \equiv 2 \bmod 53^2$,
$n \equiv 67 \bmod 89$,
to determine $n = 78654$.

Pohlig-Hellman method reduces security of discrete logarithm problem in group generated by P to security of largest prime order subgroup.

The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f$

Birthday paradox:
Randomly choosing from ℓ
elements picks one element
after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle
Cycle-finding algorithm
(e.g., Floyd) quickly detects

$(2 \cdot 53^2)P = (877560, 947848)$ and $(2 \cdot 53^2)Q = (822491, 118220)$.
Compute $n_4 = 67$,
e.g. using BSGS.

Use Chinese Remainder Theorem
$n \equiv 0 \bmod 2$,
$n \equiv 2 \bmod 53^2$,
$n \equiv 67 \bmod 89$,
to determine $n = 78654$.

Pohlig-Hellman method reduces security of discrete logarithm problem in group generated by P to security of largest prime order subgroup.

The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f(W_i)$.

Birthday paradox:
Randomly choosing from ℓ
elements picks one element twice
after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm
(e.g., Floyd) quickly detects this.

$P = (877560, 947848)$ and
$)Q = (822491, 118220)$.
e $n_4 = 67$,
g BSGS.

nese Remainder Theorem
nod 2,
nod 53^2,
mod 89,
mine $n = 78654$.

Hellman method reduces
of discrete logarithm
in group generated by P
ity of largest prime order
p.

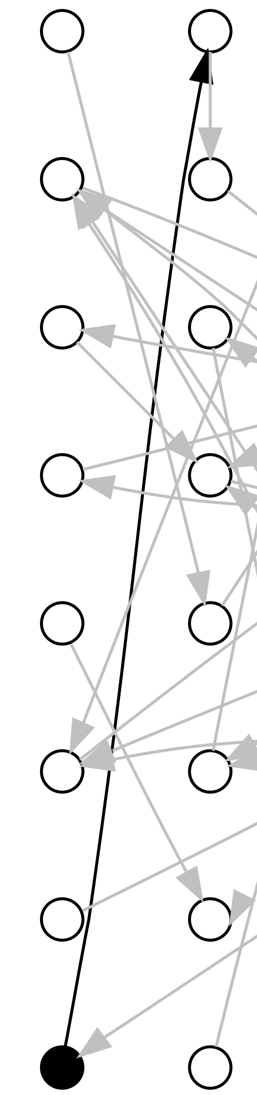
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f(W_i)$.

Birthday paradox:
Randomly choosing from ℓ
elements picks one element twice
after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm
(e.g., Floyd) quickly detects this.



560, 947848) and
2491, 118220).
,

inder Theorem

78654.

ethod reduces
e logarithm
generated by P
est prime order

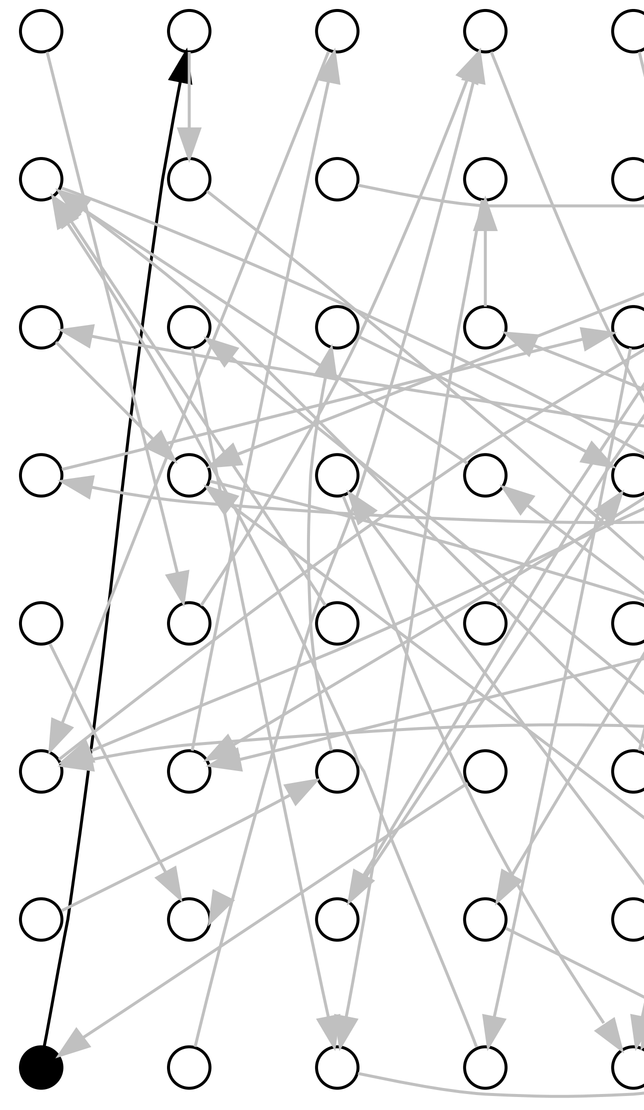
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



8) and
20).

orem

uces
n
by P
order

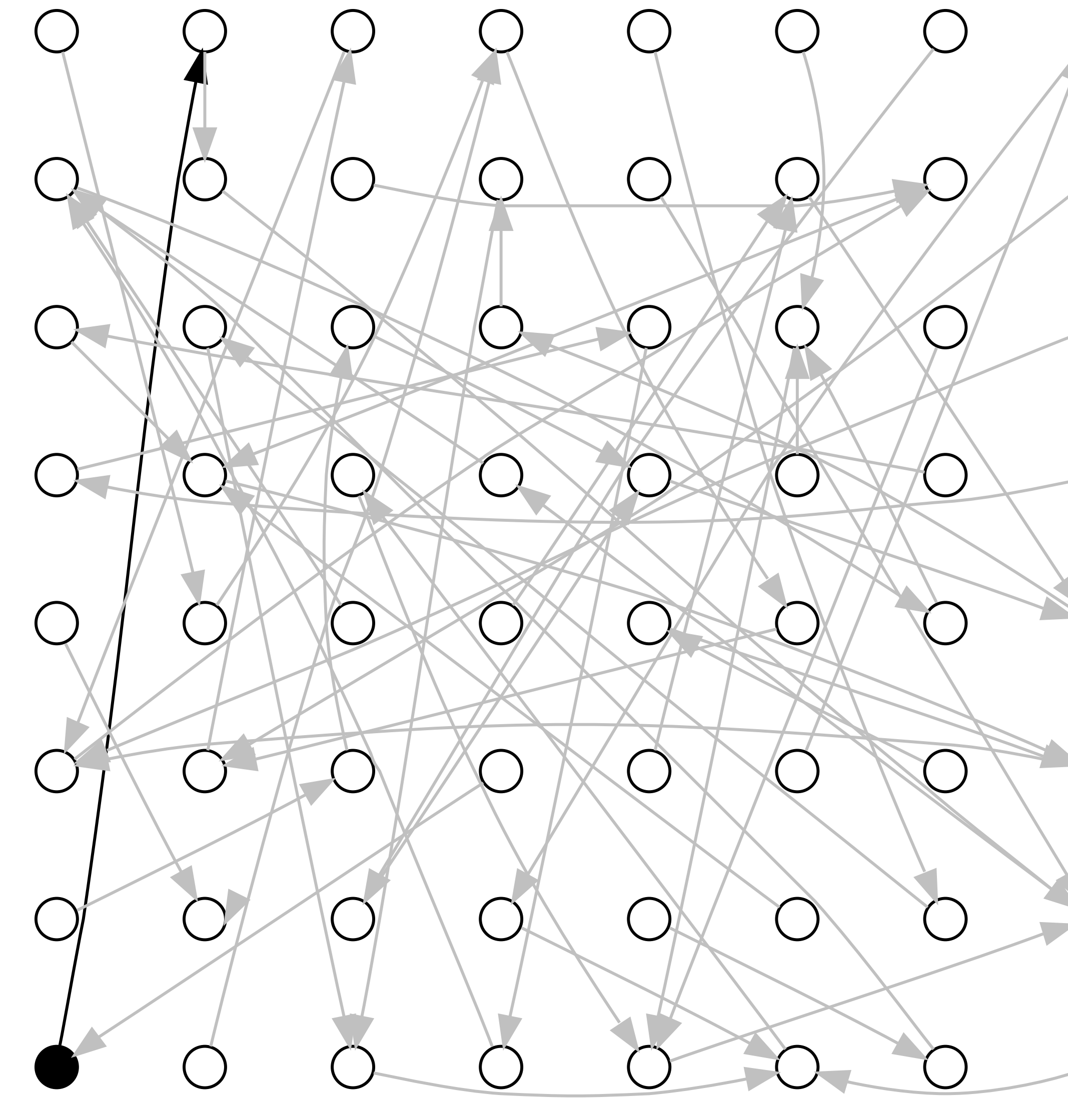
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



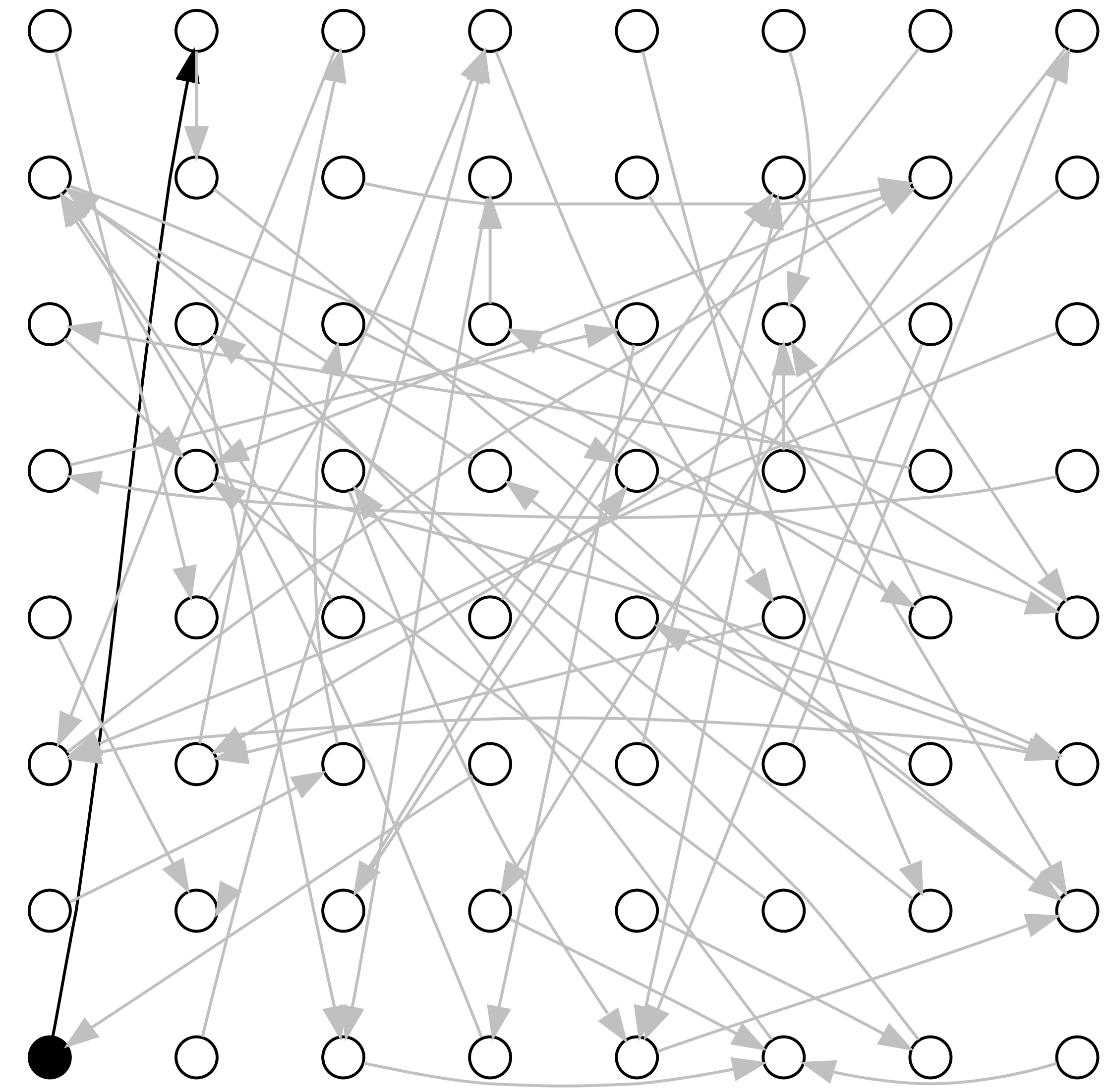
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f(W_i)$.

Birthday paradox:
Randomly choosing from ℓ
elements picks one element twice
after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm
(e.g., Floyd) quickly detects this.



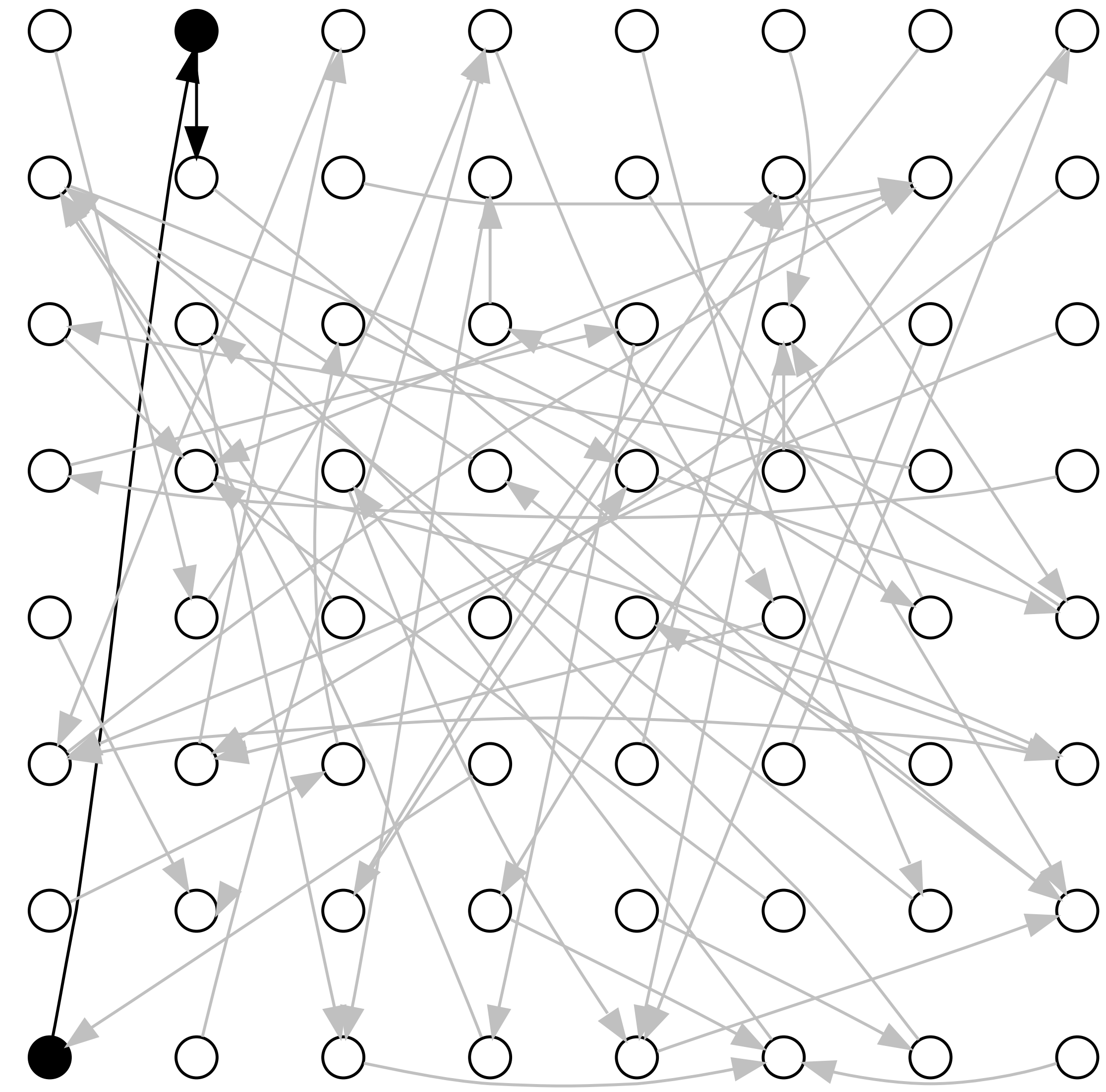
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



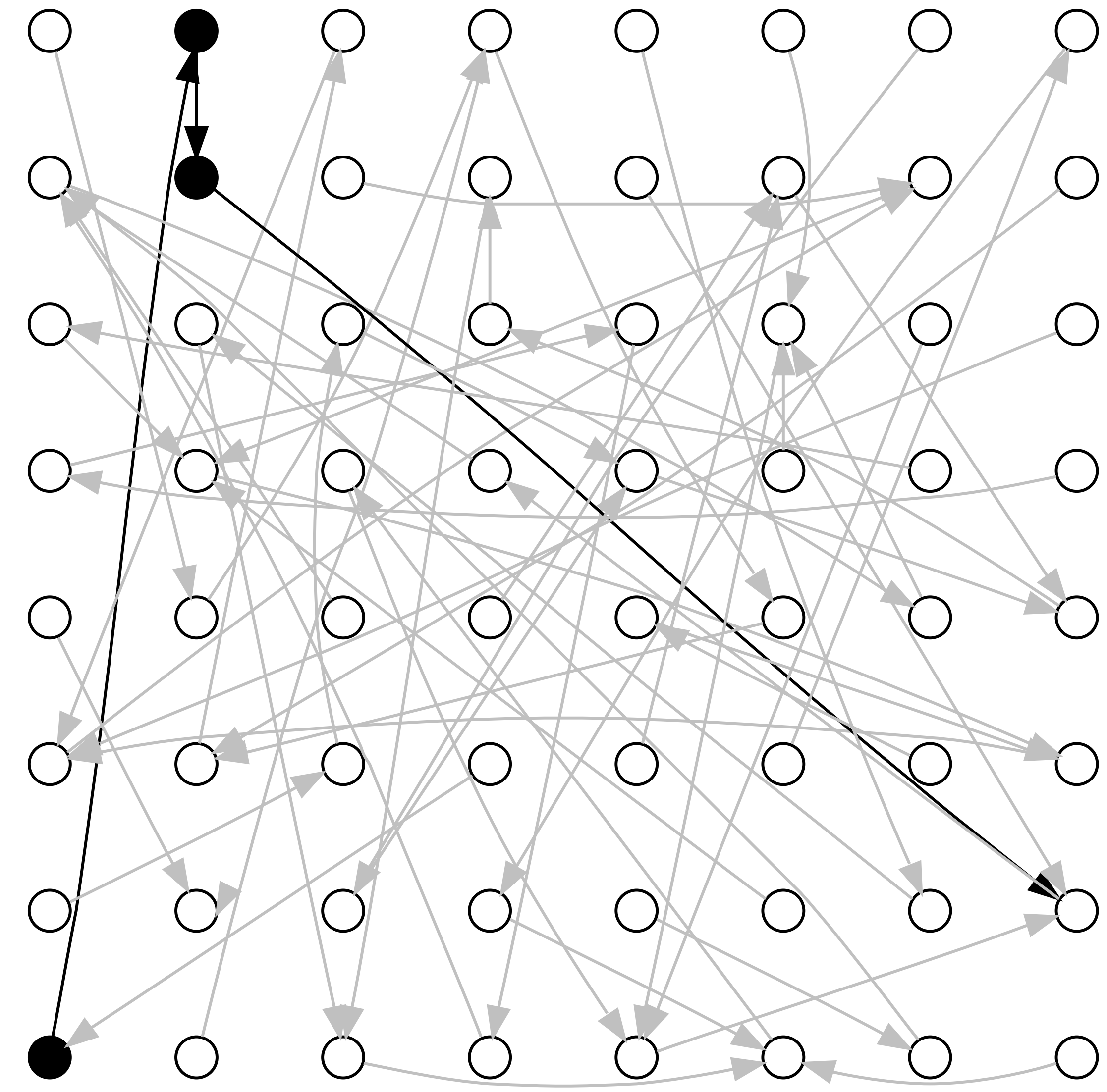
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



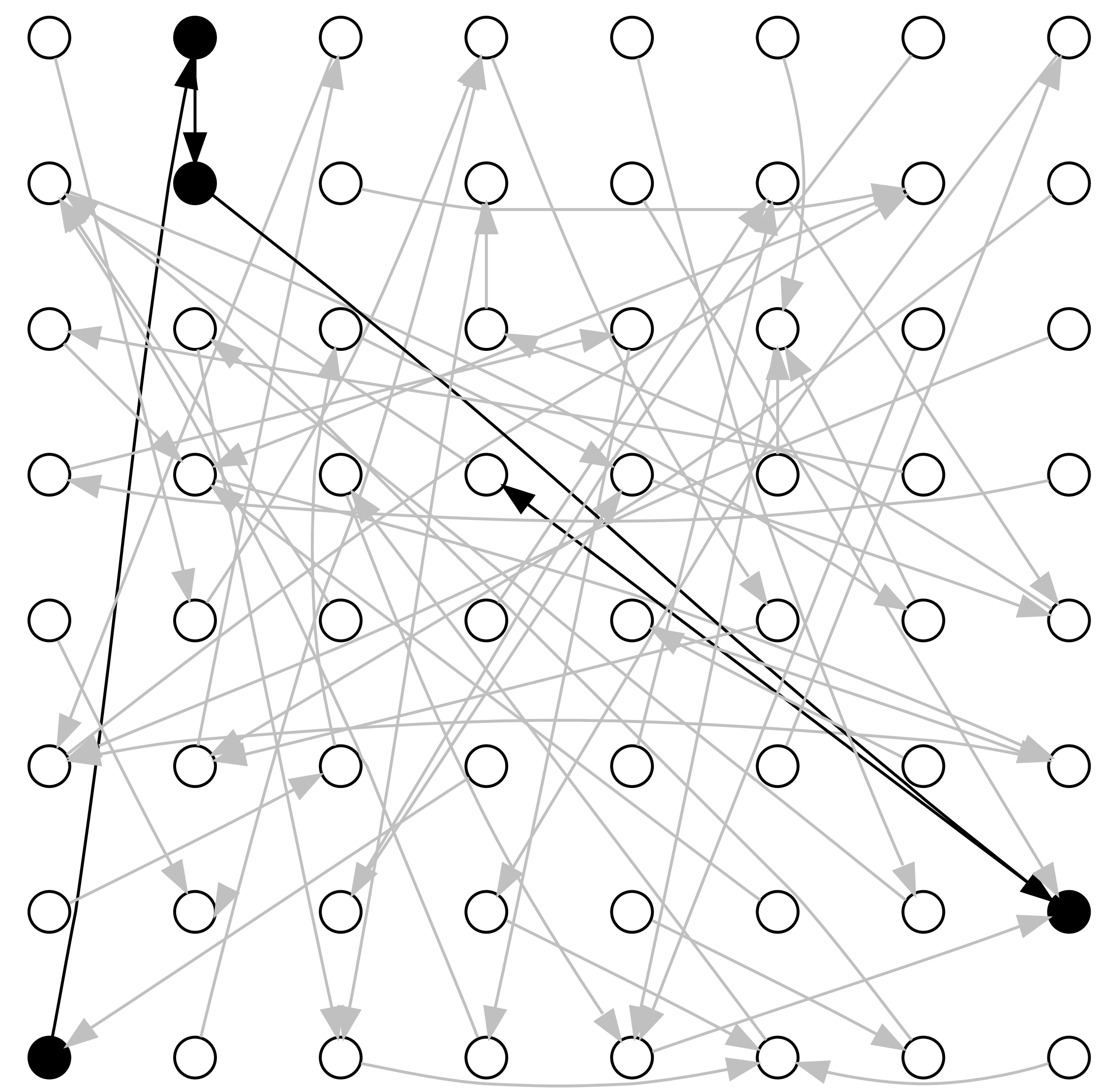
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



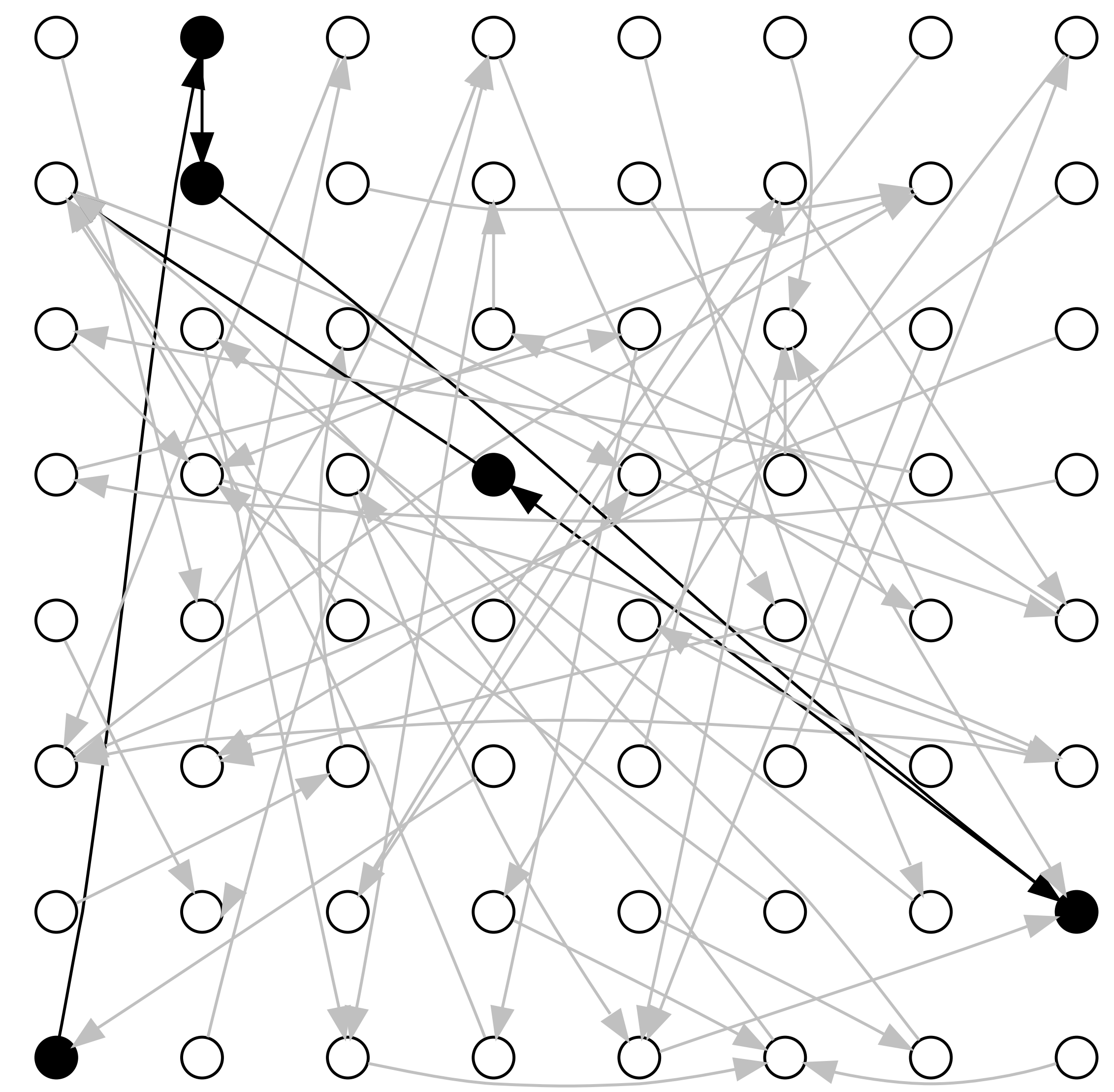
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



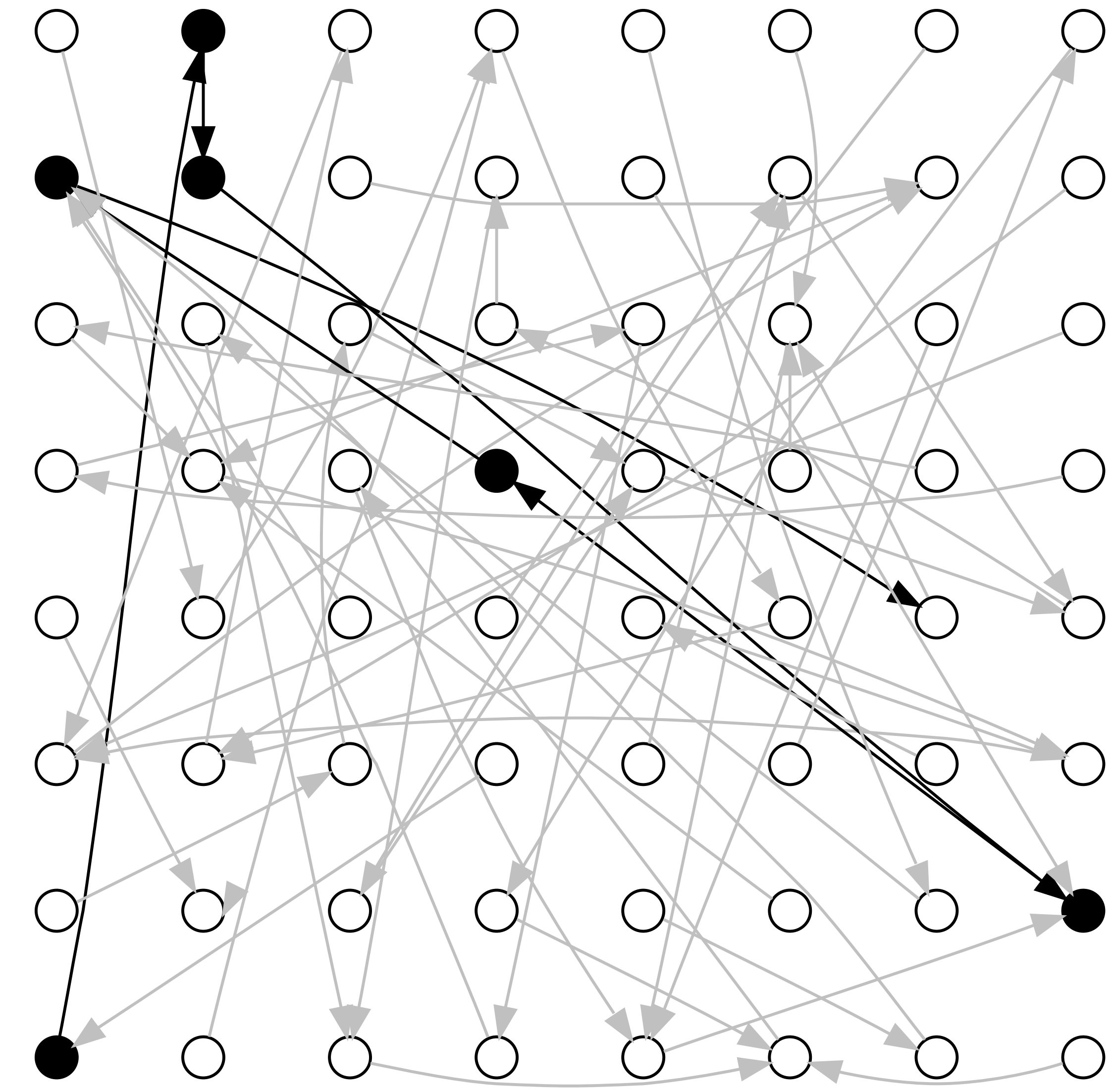
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



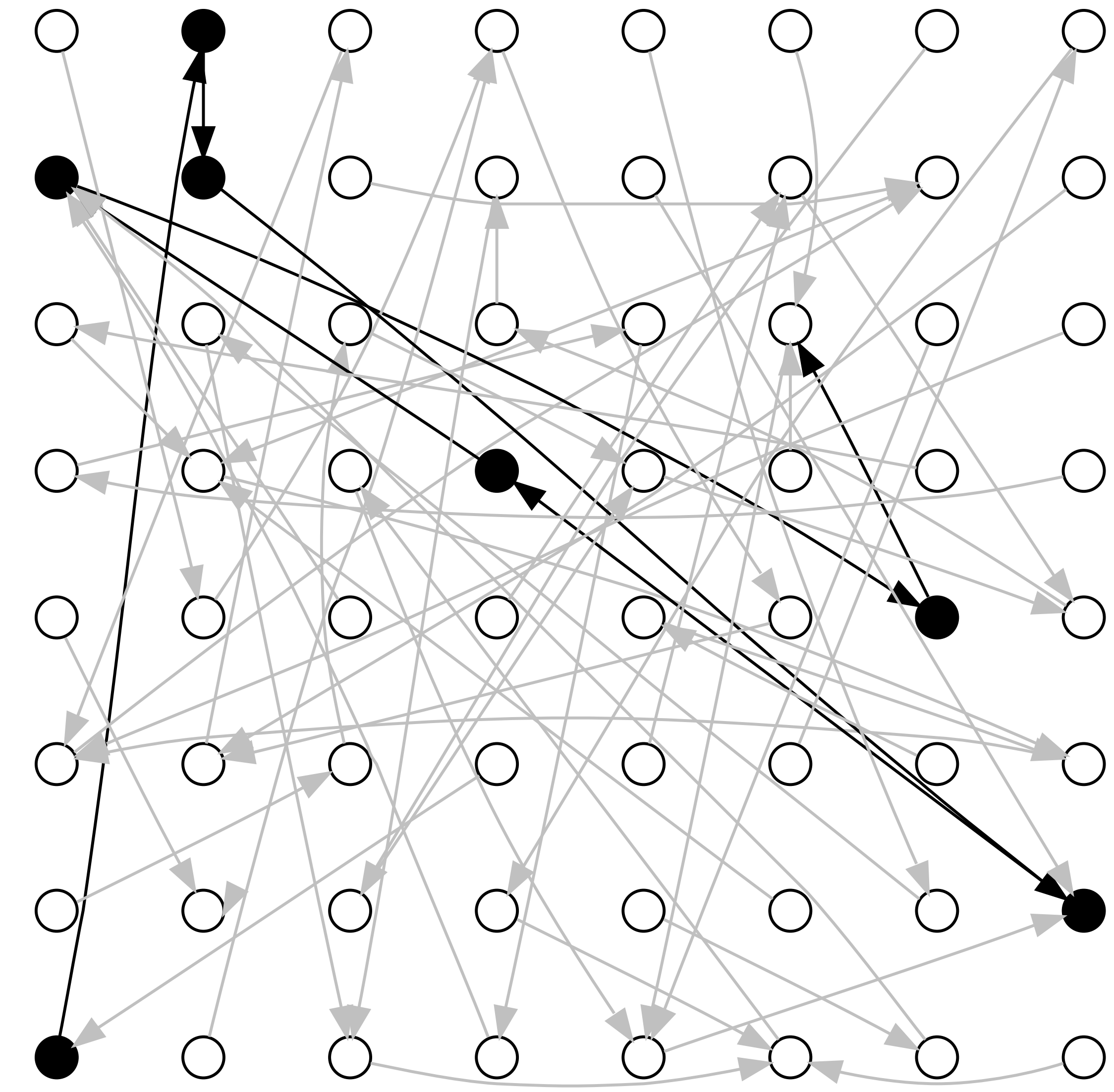
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



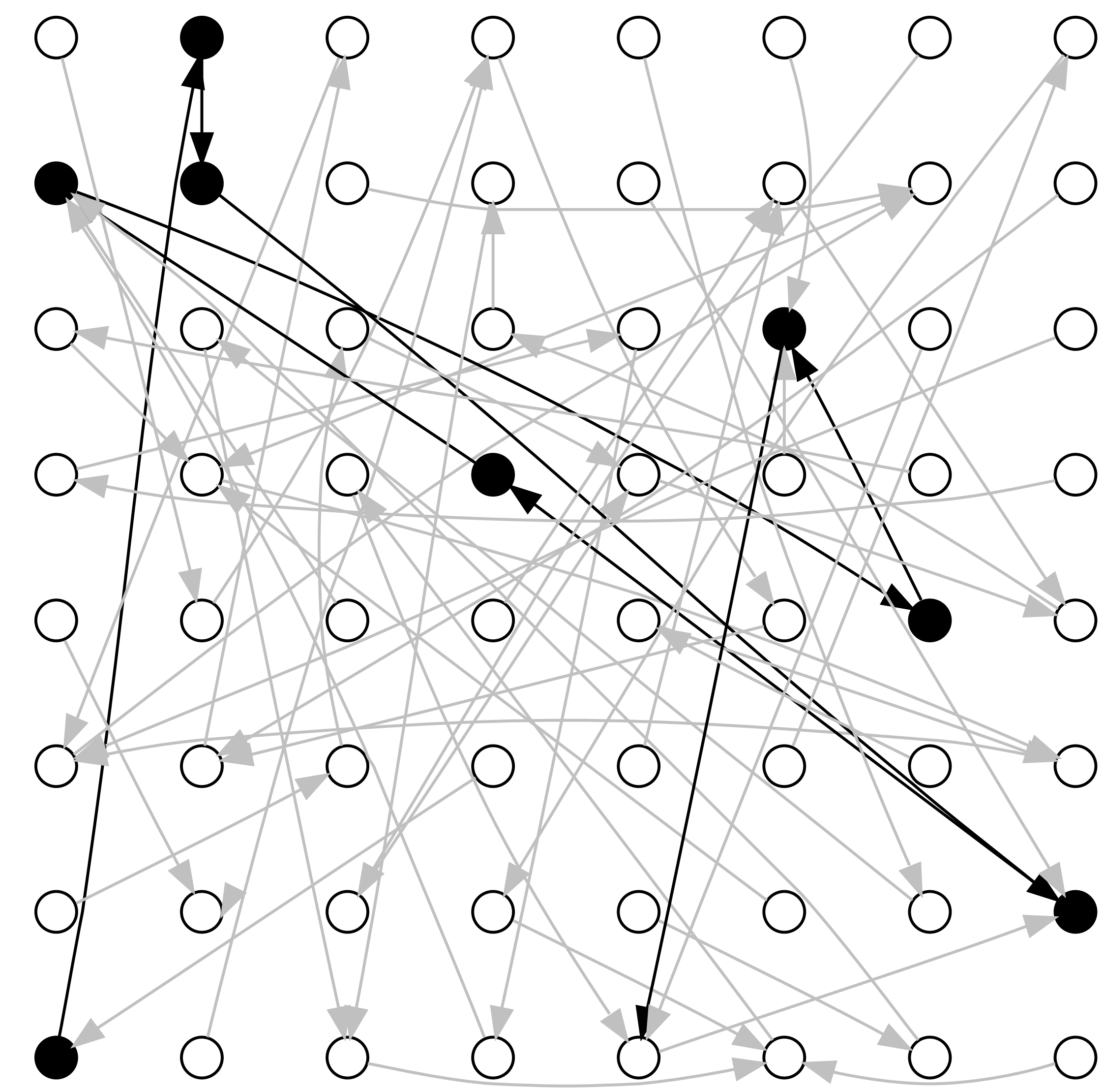
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



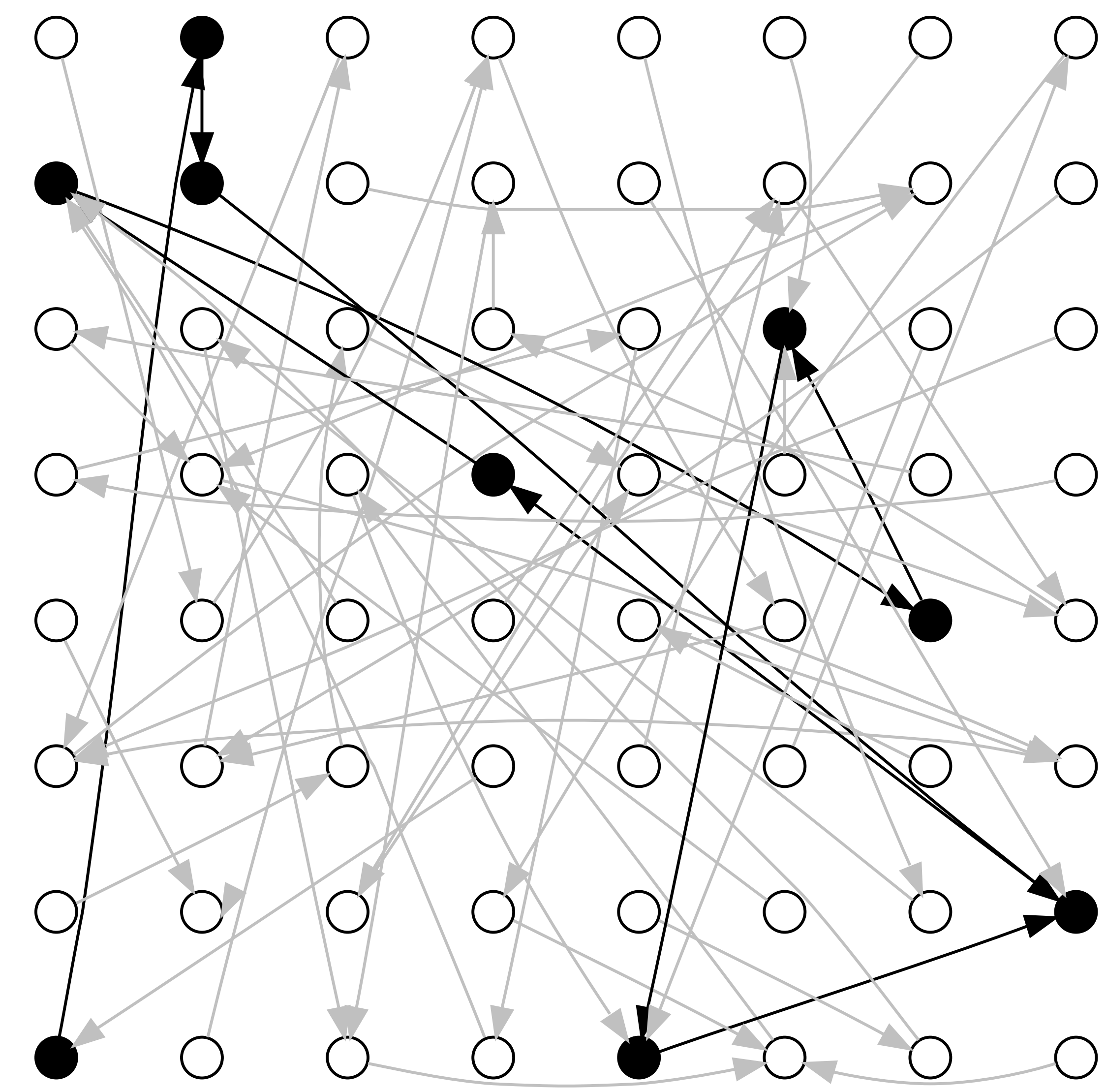
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



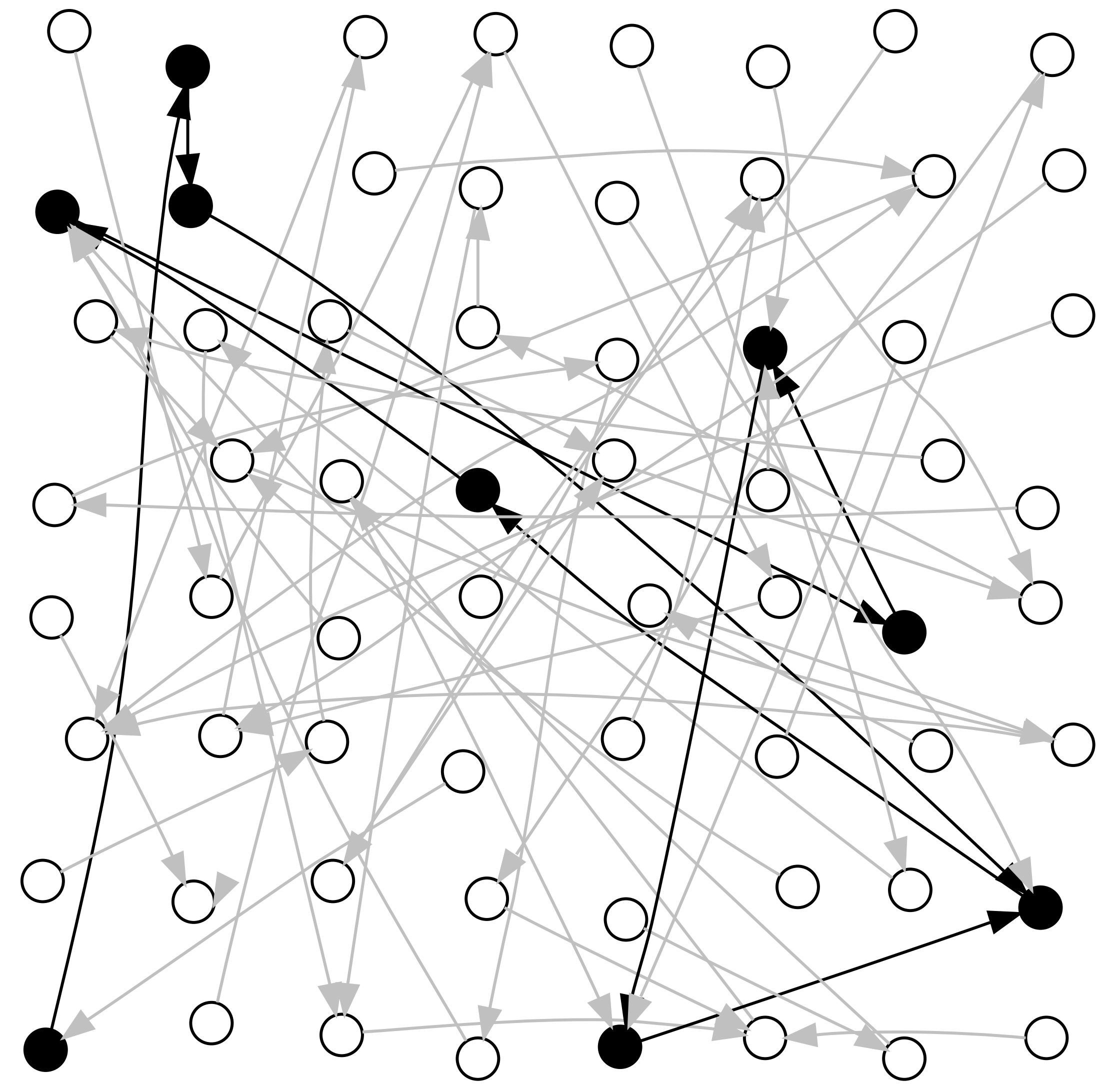
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



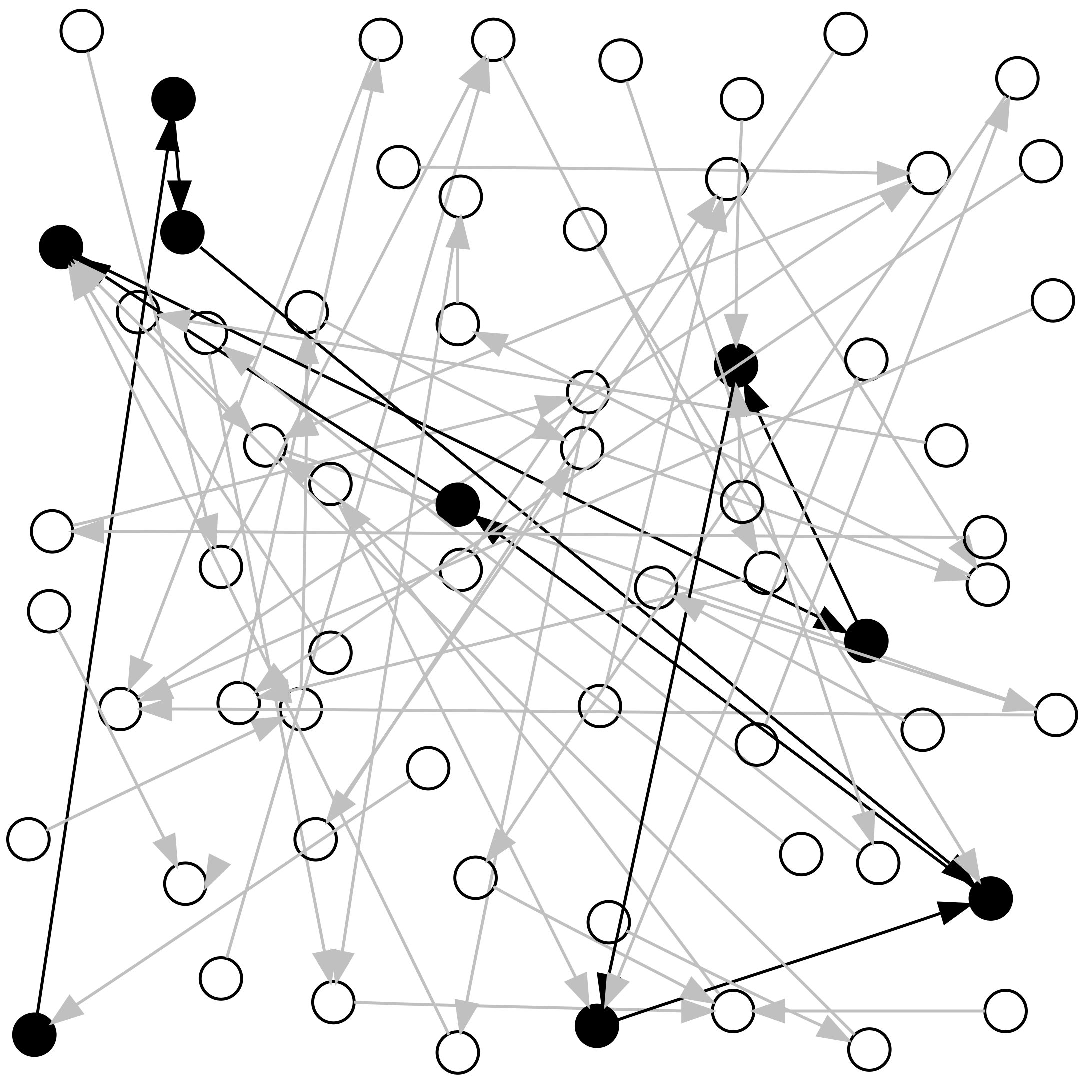
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



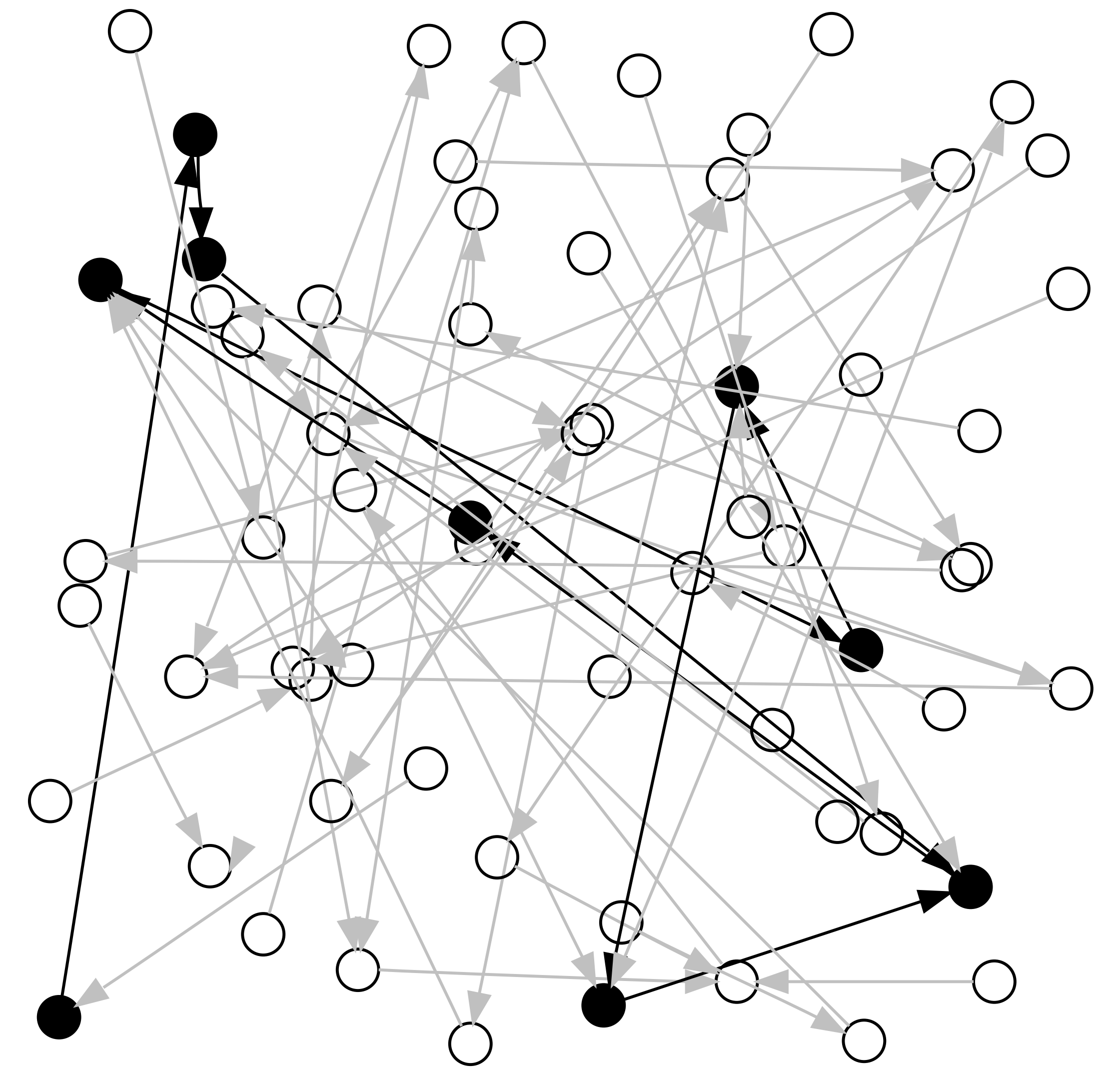
<u>The rho method</u>

Simplified, non-parallel rho:

Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f(W_i)$.

Birthday paradox:
Randomly choosing from ℓ
elements picks one element twice
after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm
(e.g., Floyd) quickly detects this.



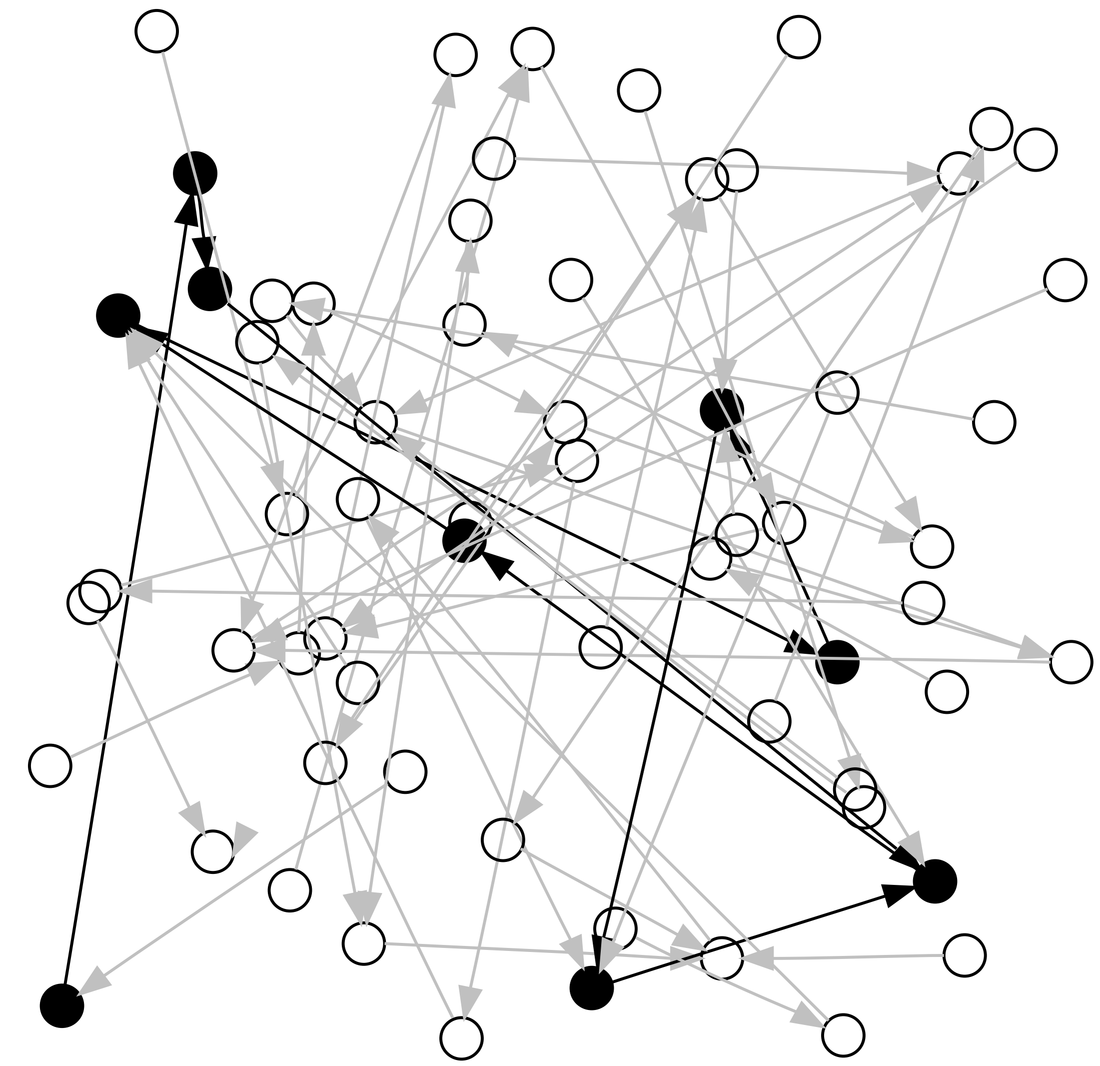
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



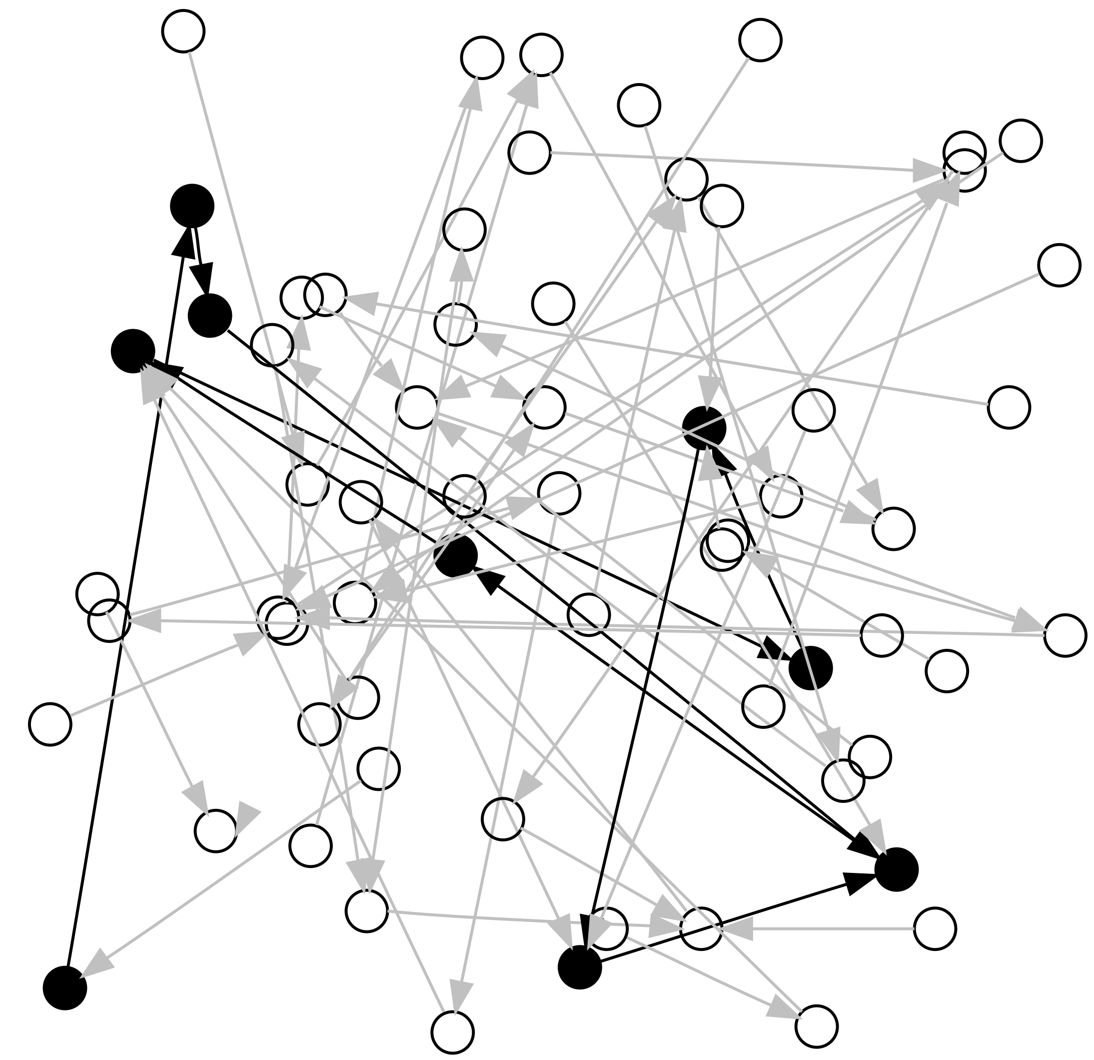
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



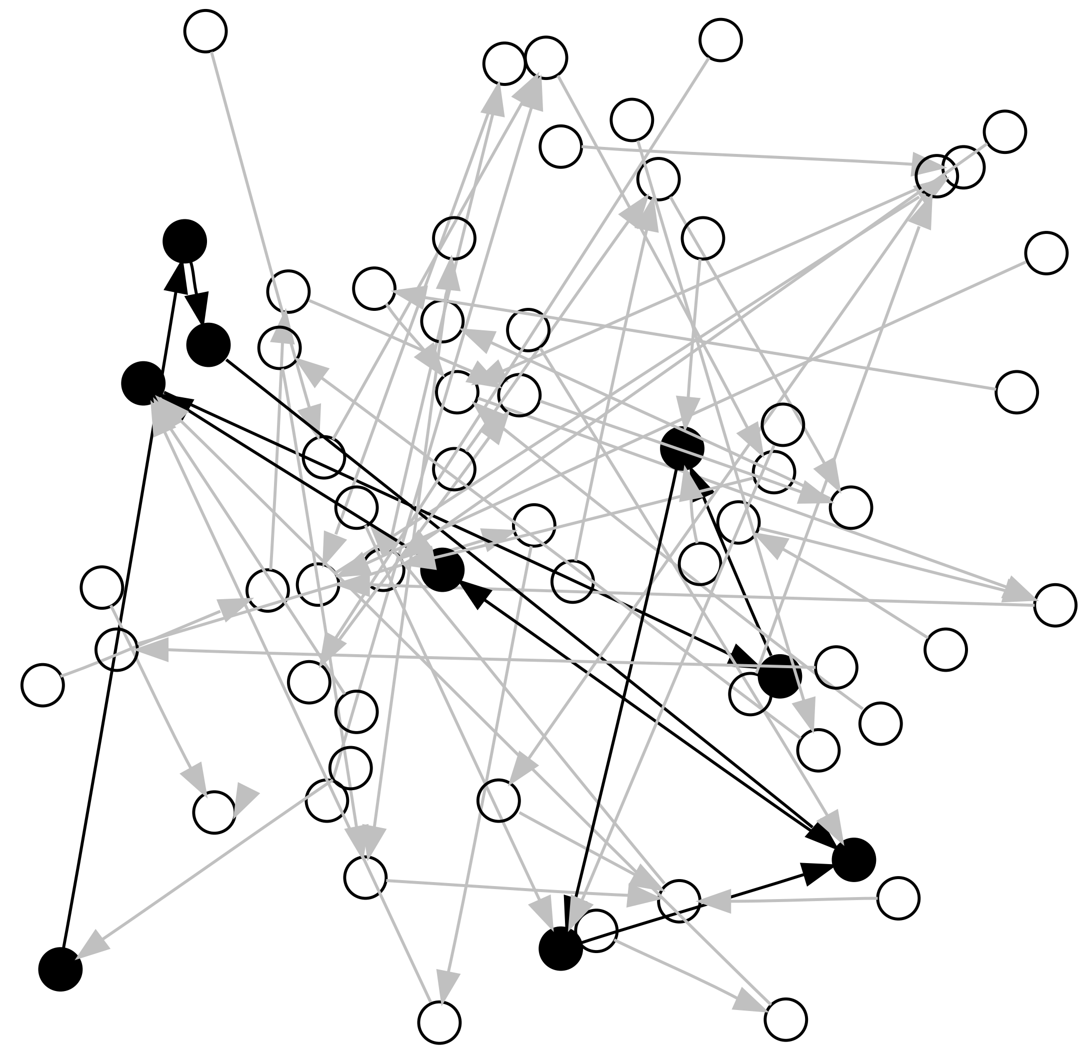
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



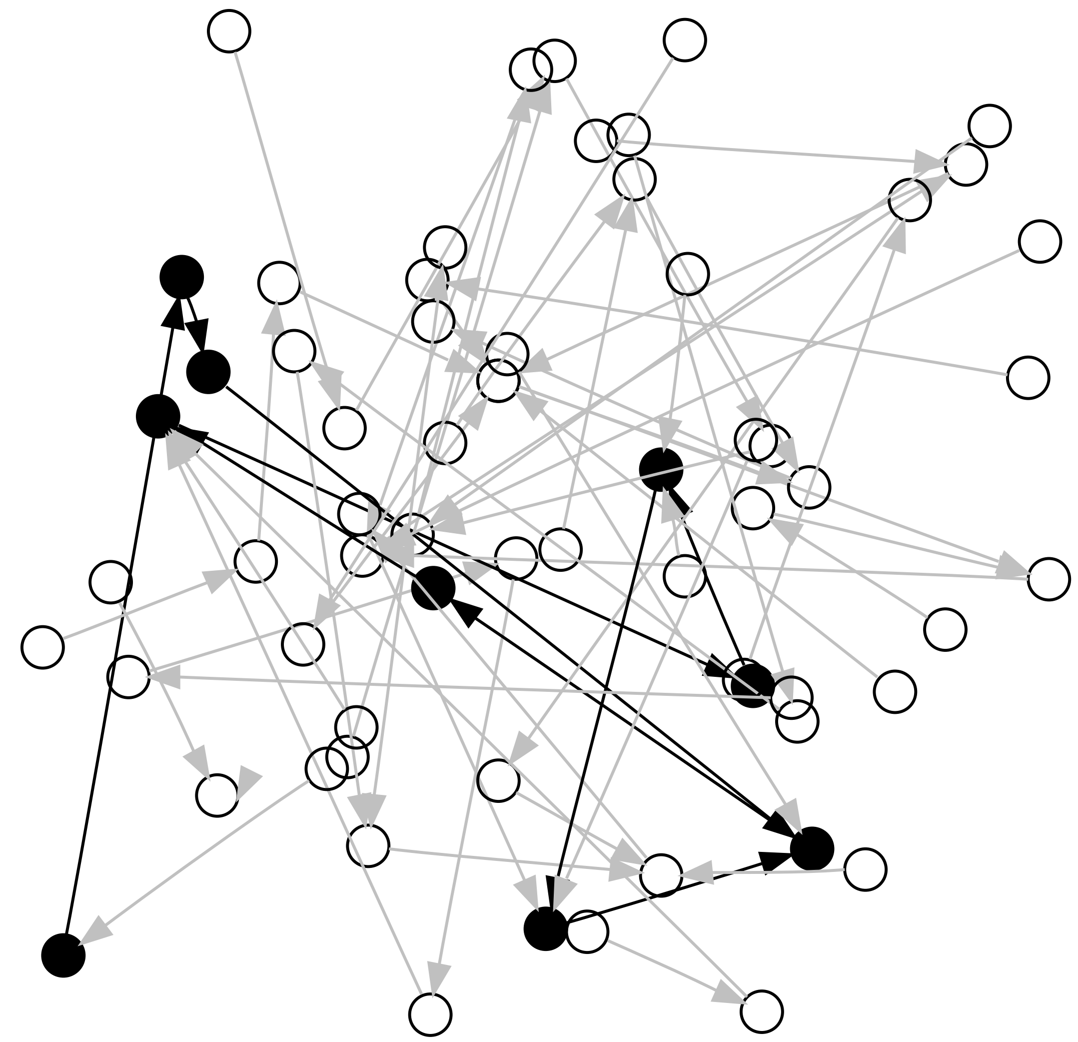
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



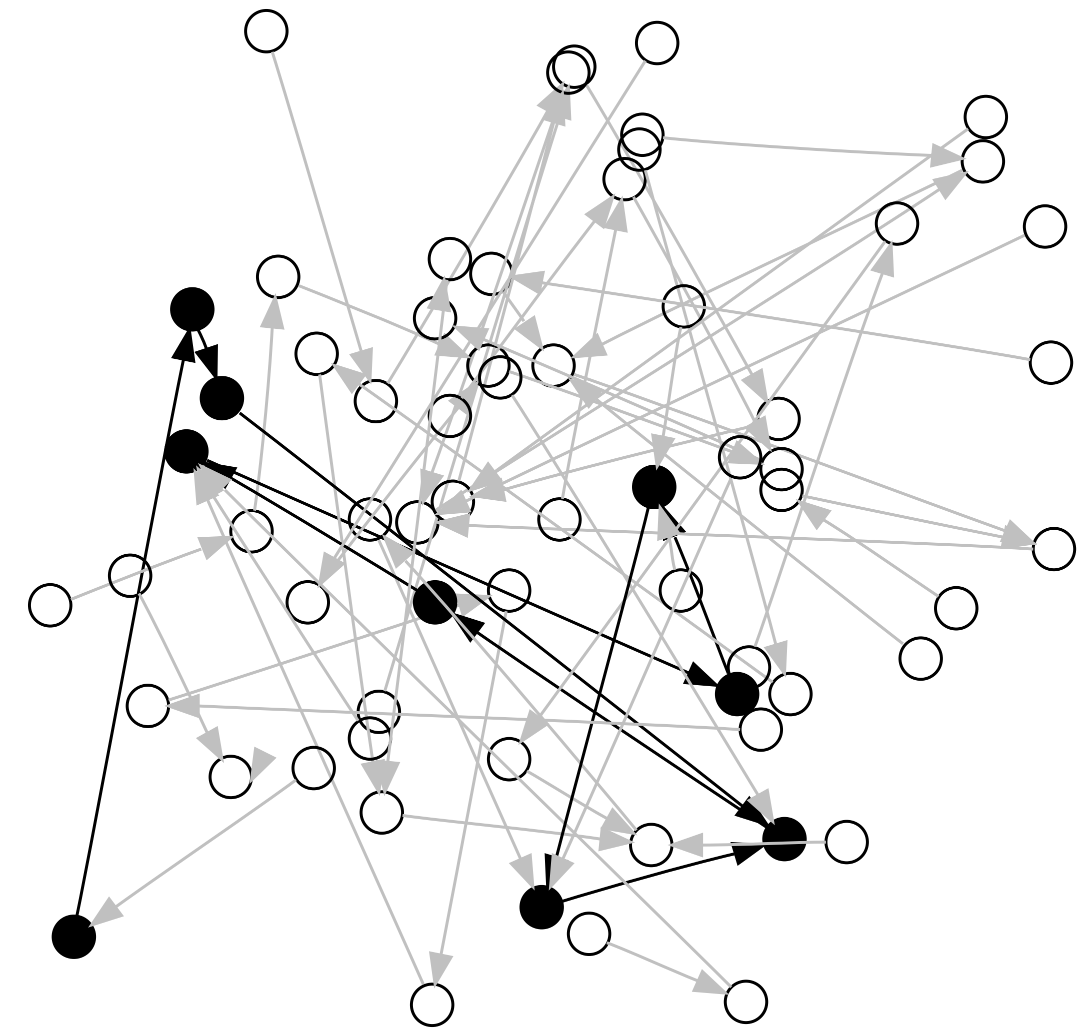
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f(W_i)$.

Birthday paradox:
Randomly choosing from ℓ
elements picks one element twice
after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm
(e.g., Floyd) quickly detects this.



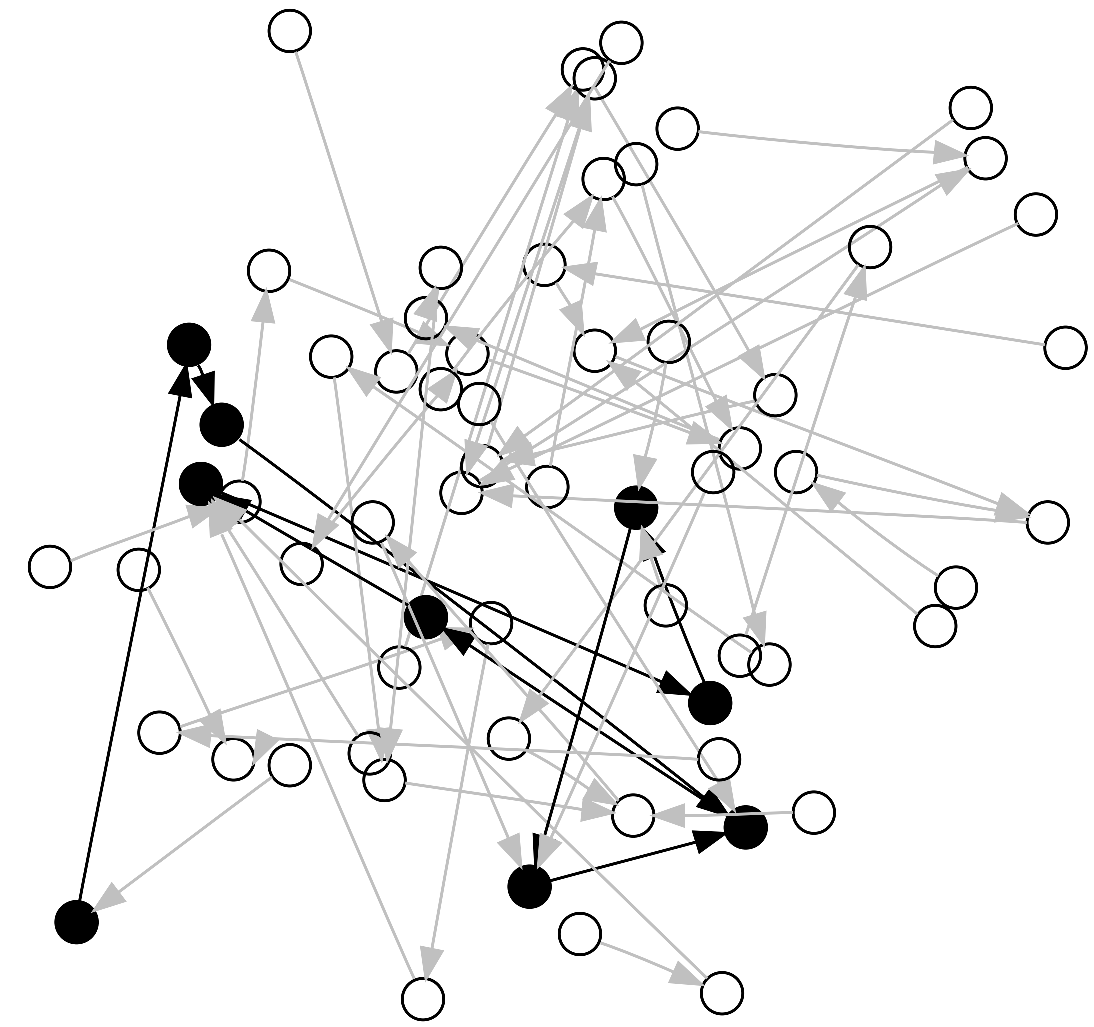
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



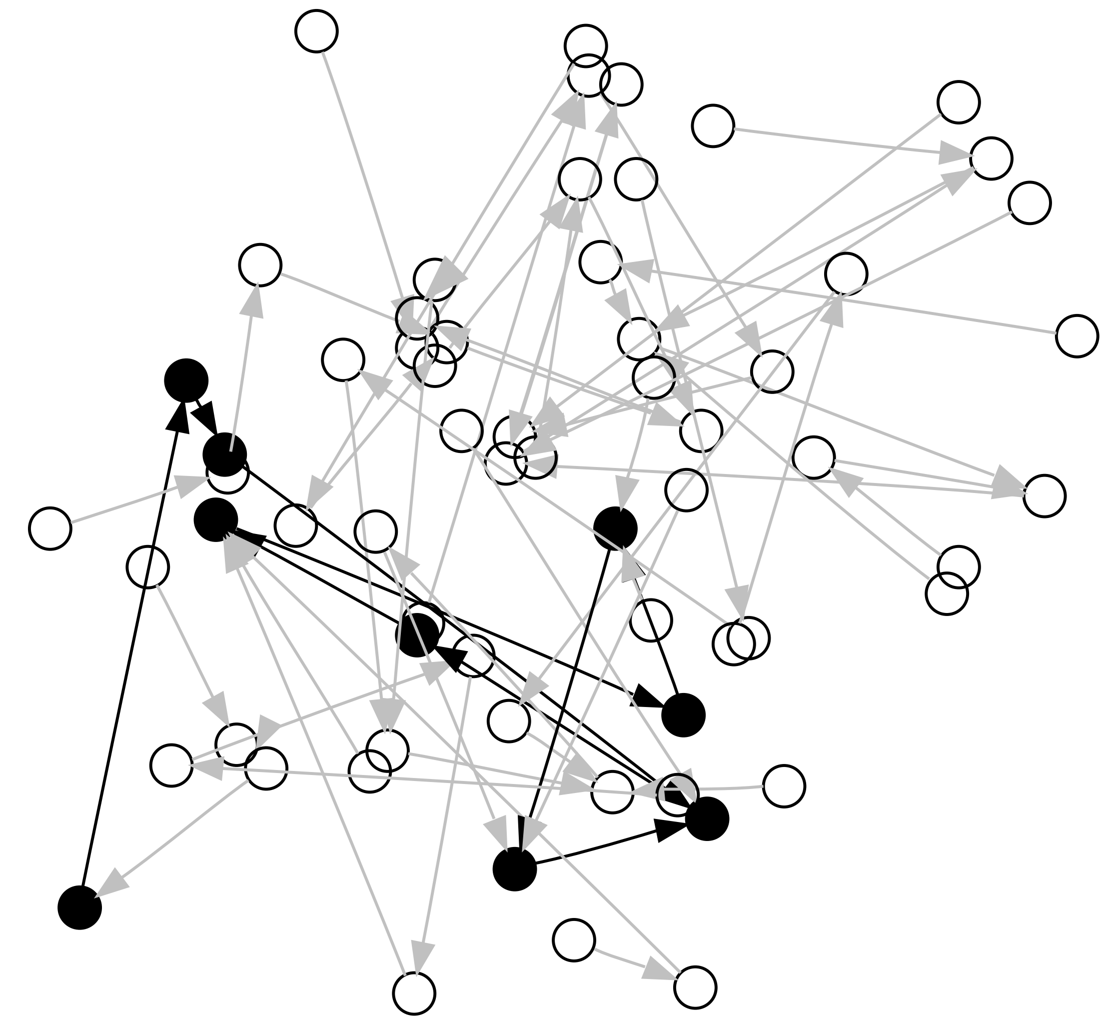
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



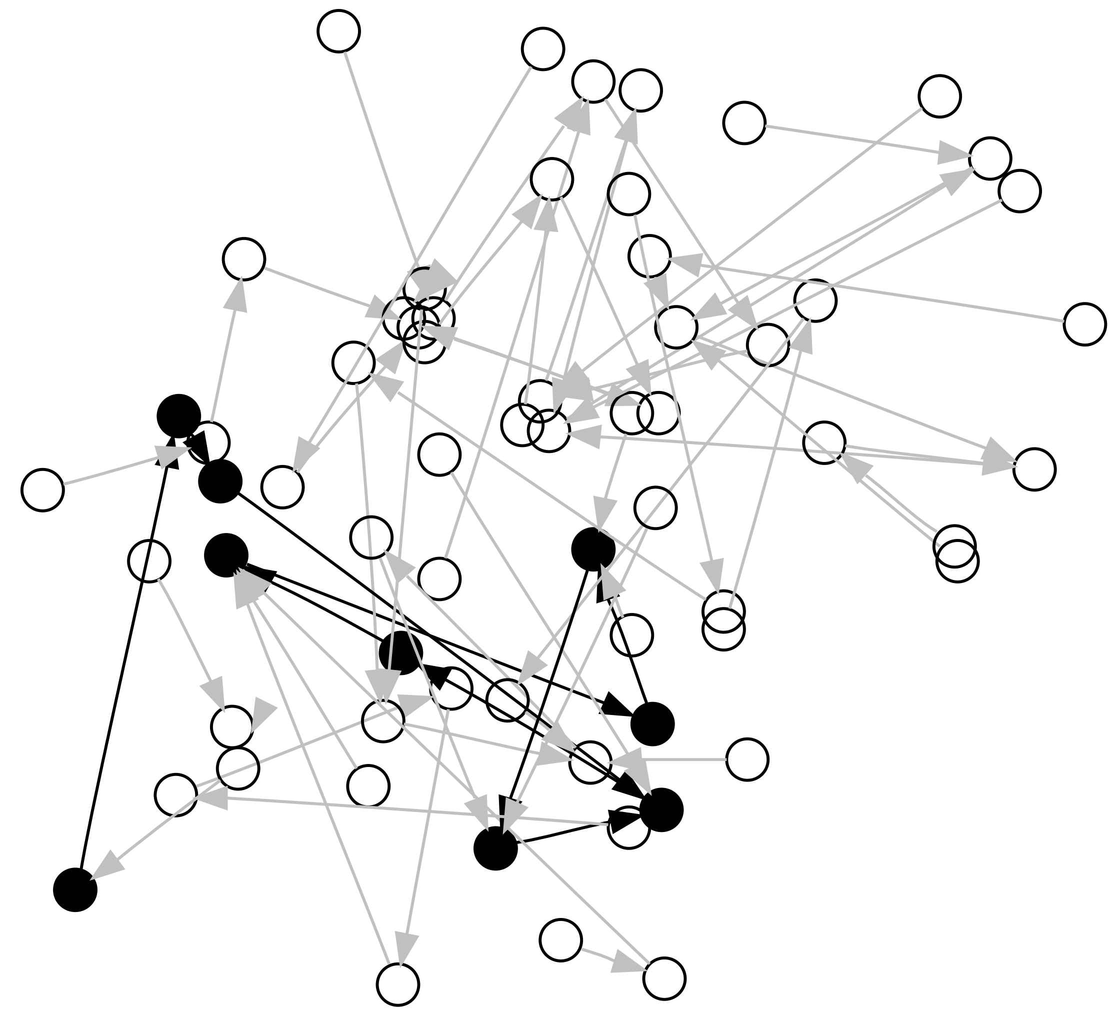
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



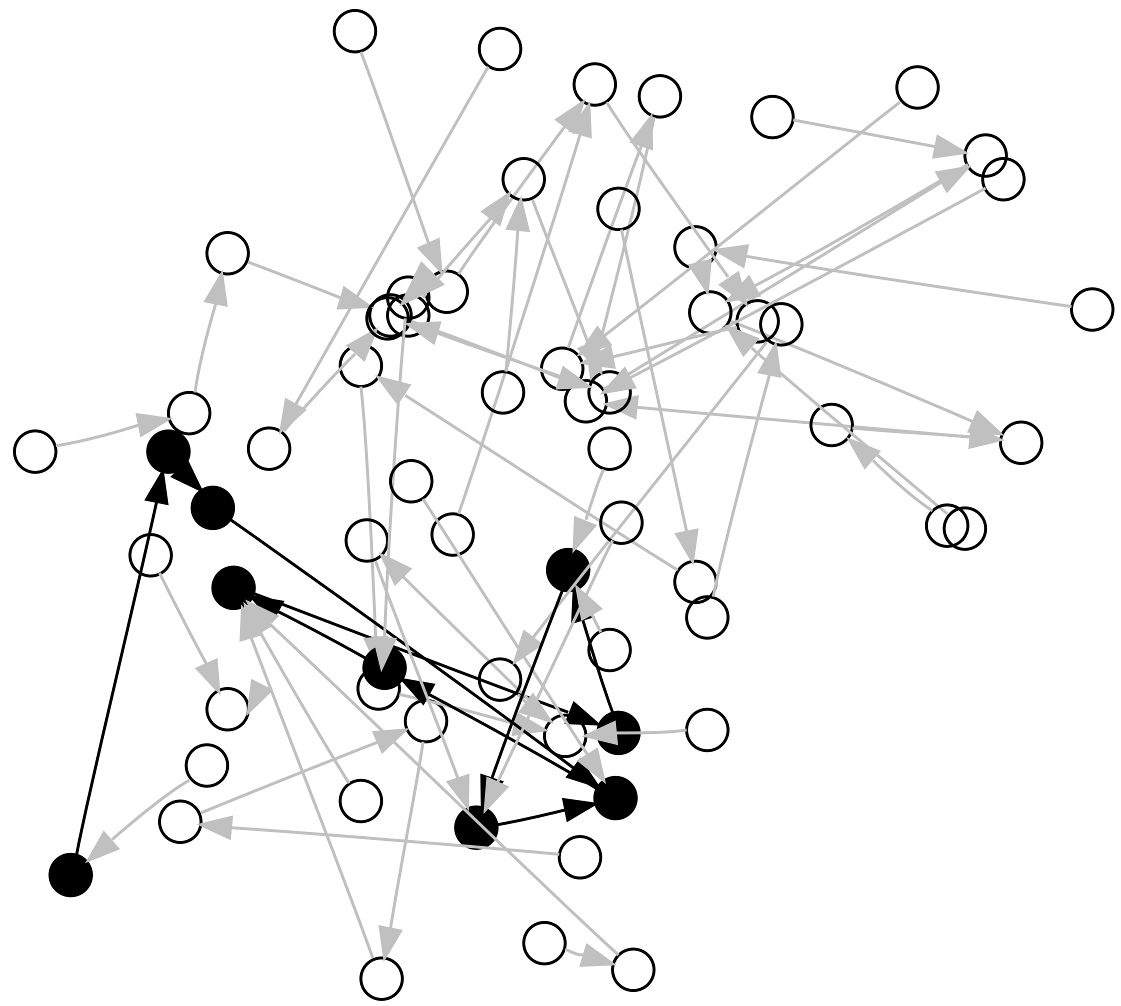
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f(W_i)$.

Birthday paradox:
Randomly choosing from $\boldsymbol{\ell}$
elements picks one element twice
after about $\sqrt{\pi \boldsymbol{\ell}/2}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm
(e.g., Floyd) quickly detects this.



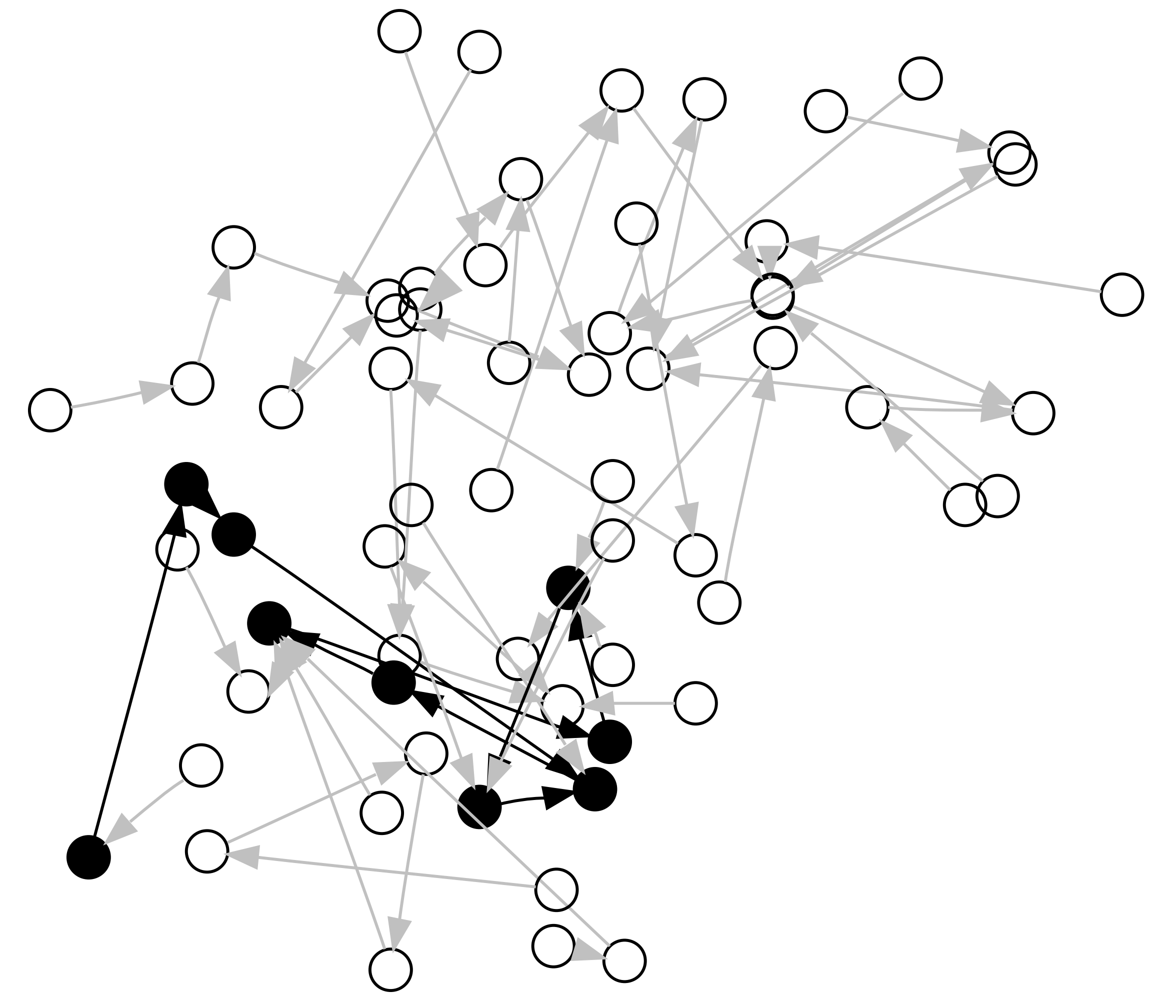
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



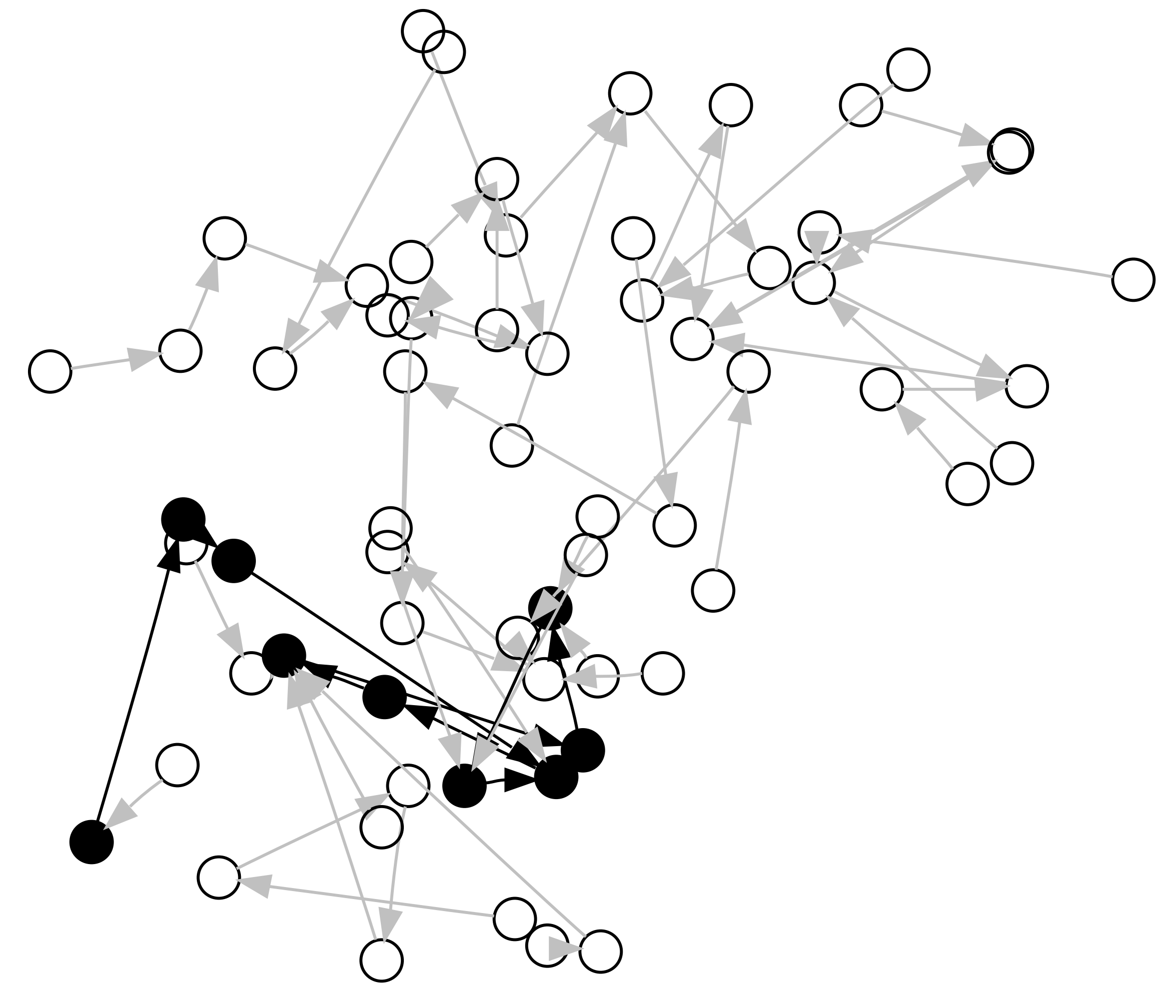
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



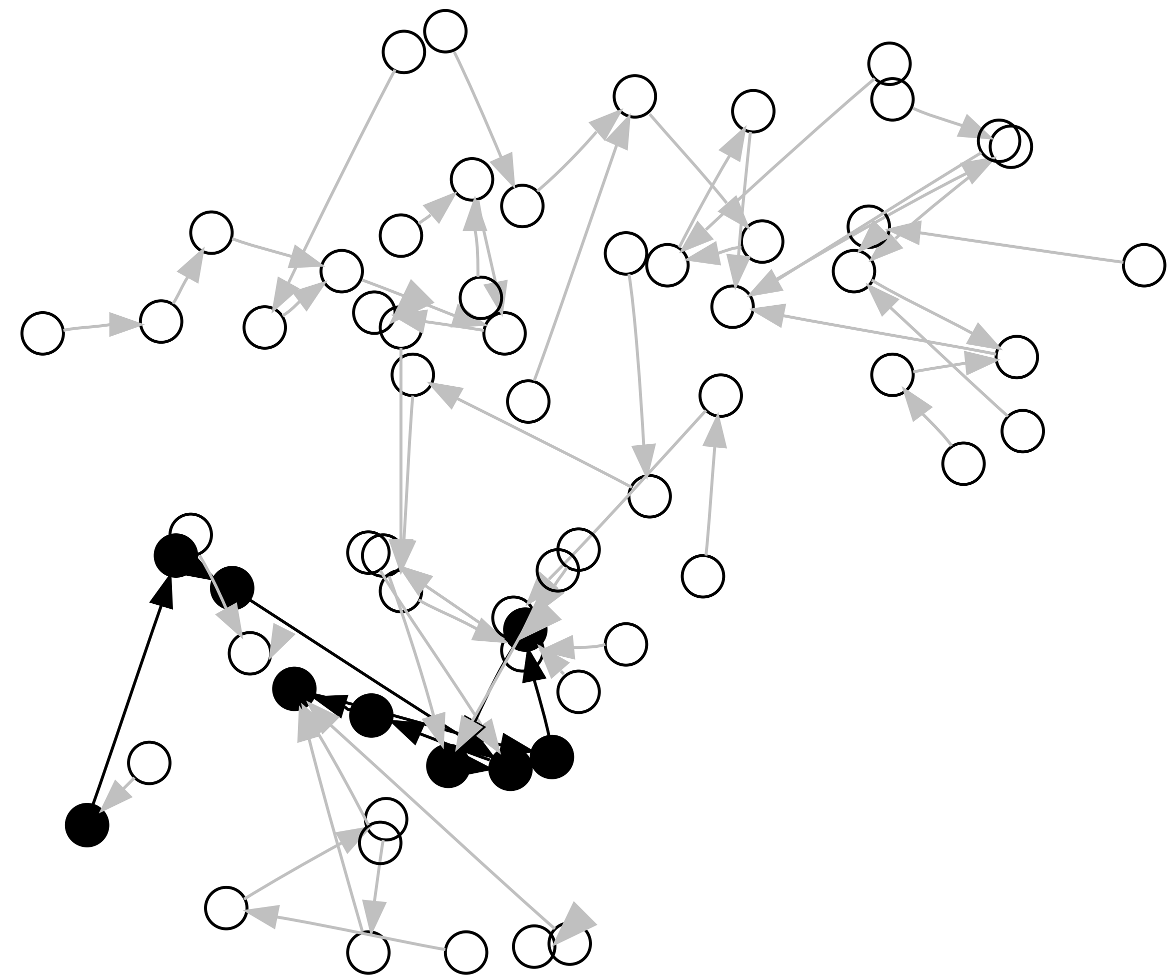
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



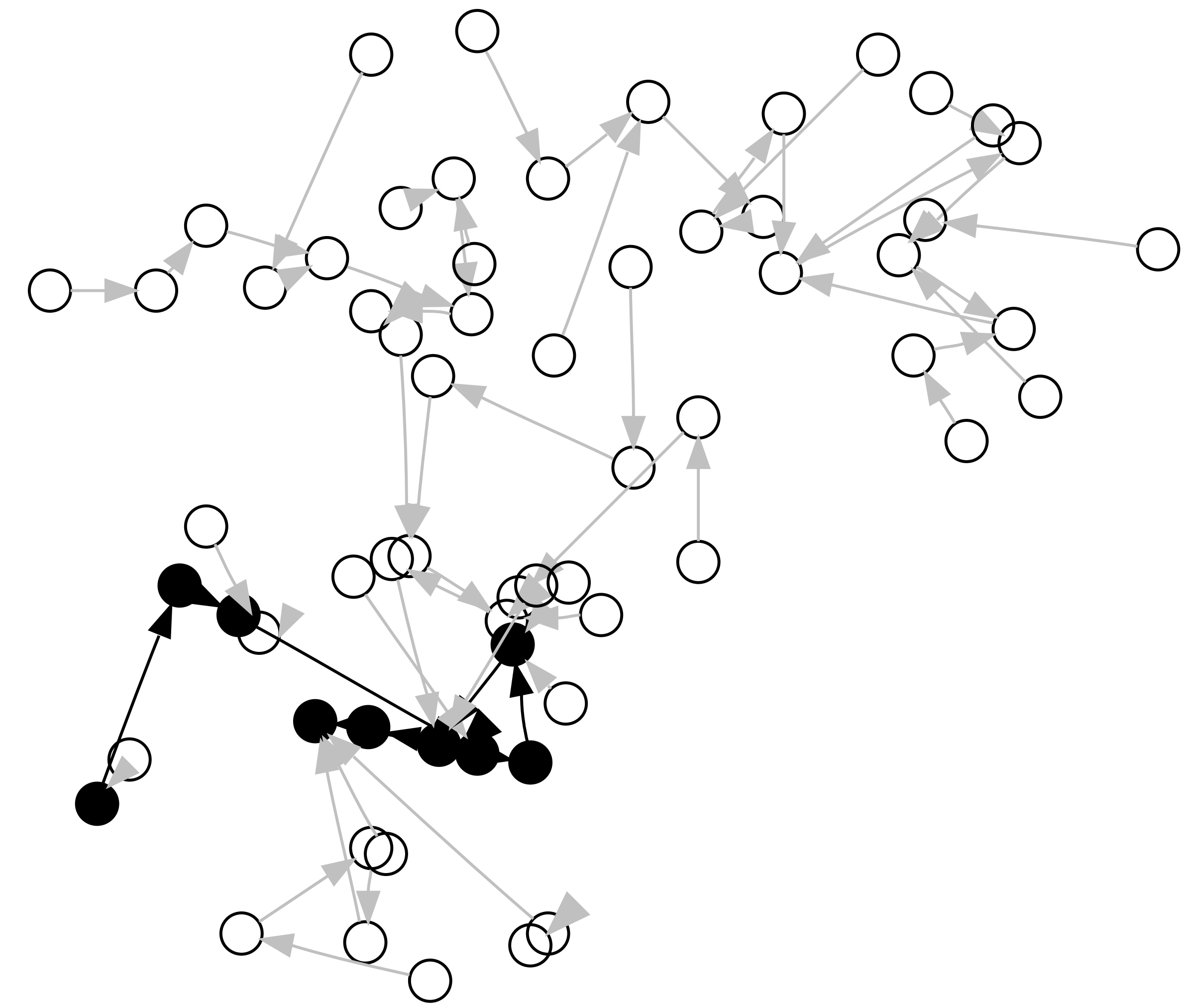
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f(W_i)$.

Birthday paradox:
Randomly choosing from ℓ
elements picks one element twice
after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm
(e.g., Floyd) quickly detects this.



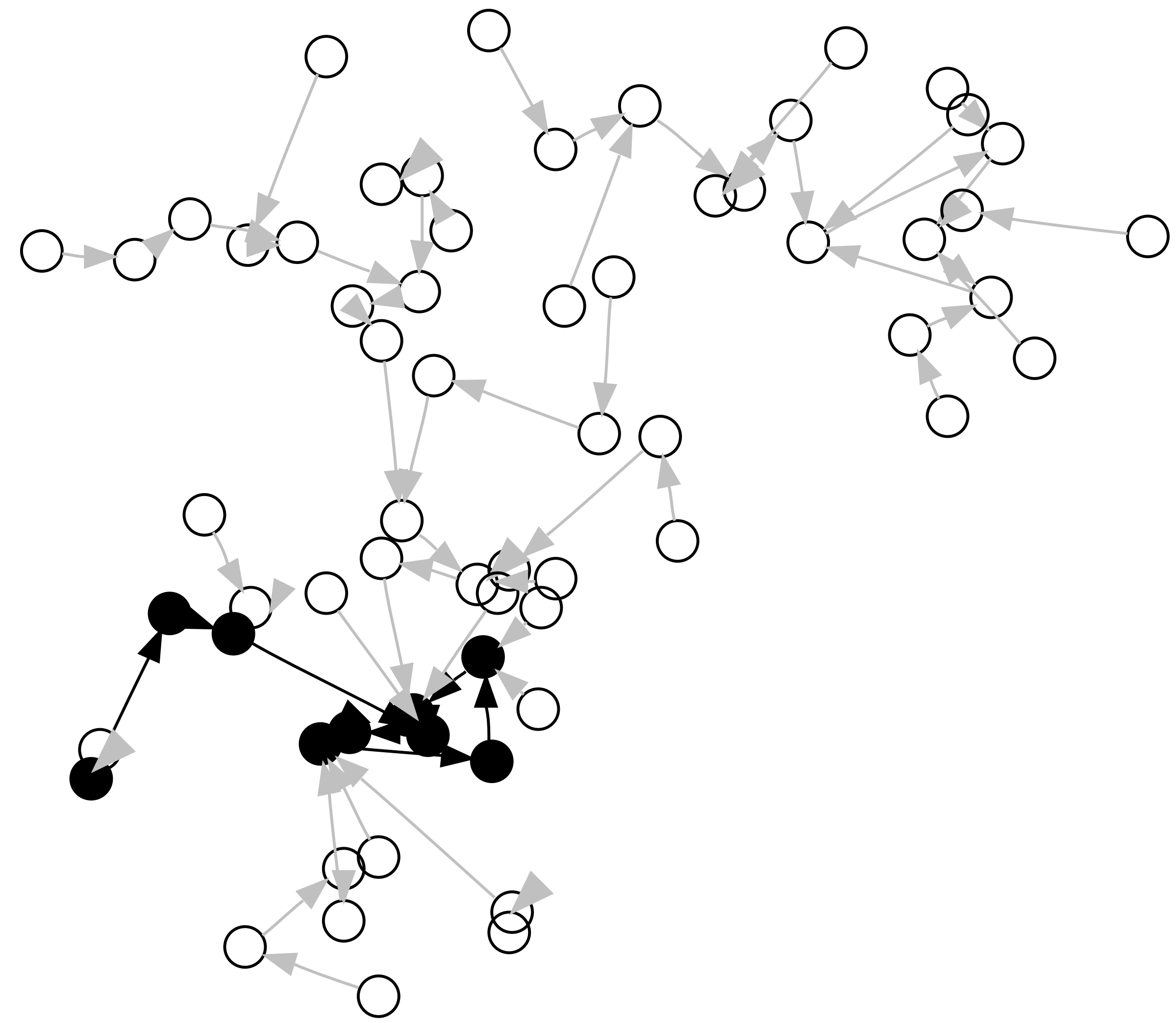
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



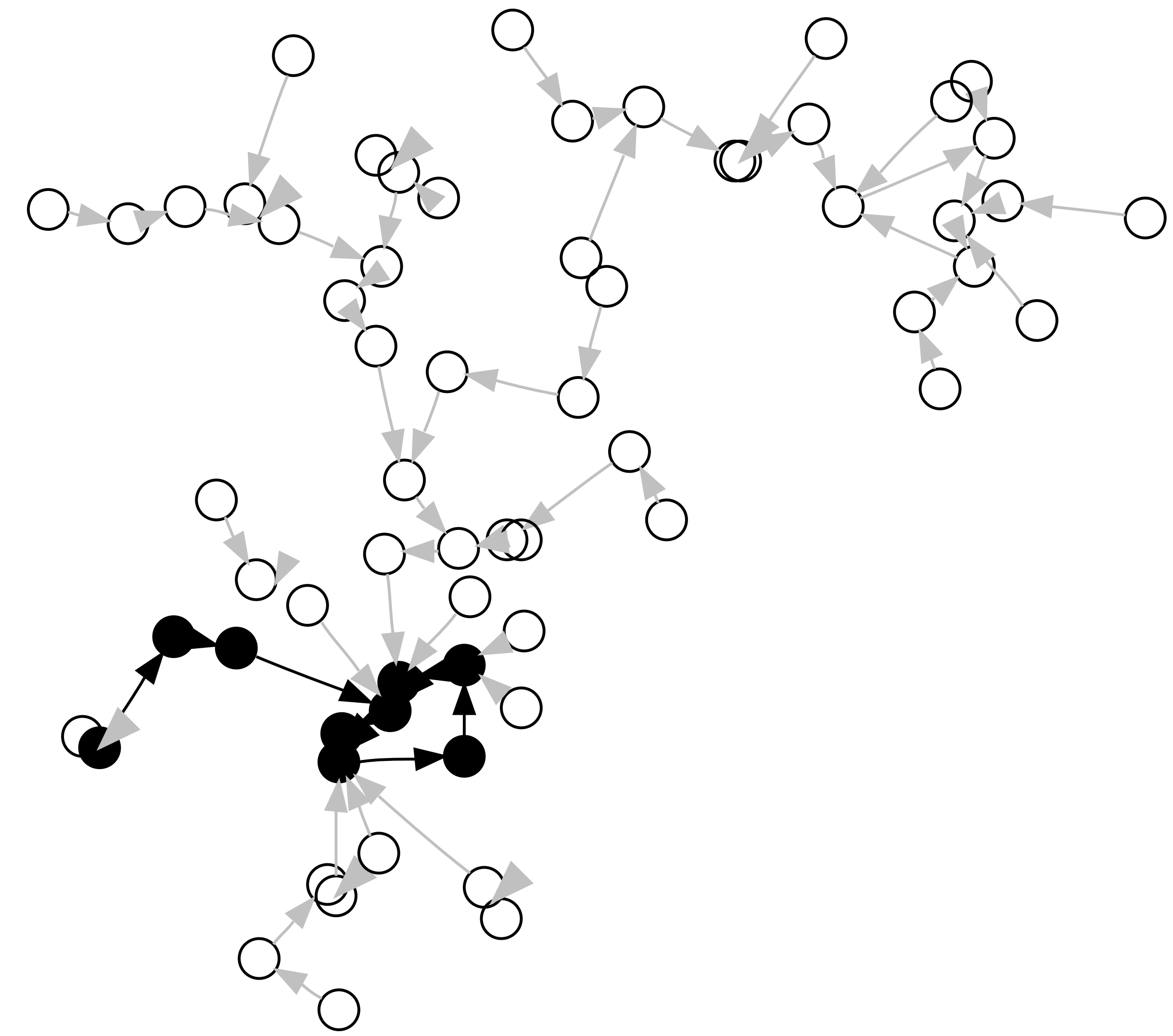
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.



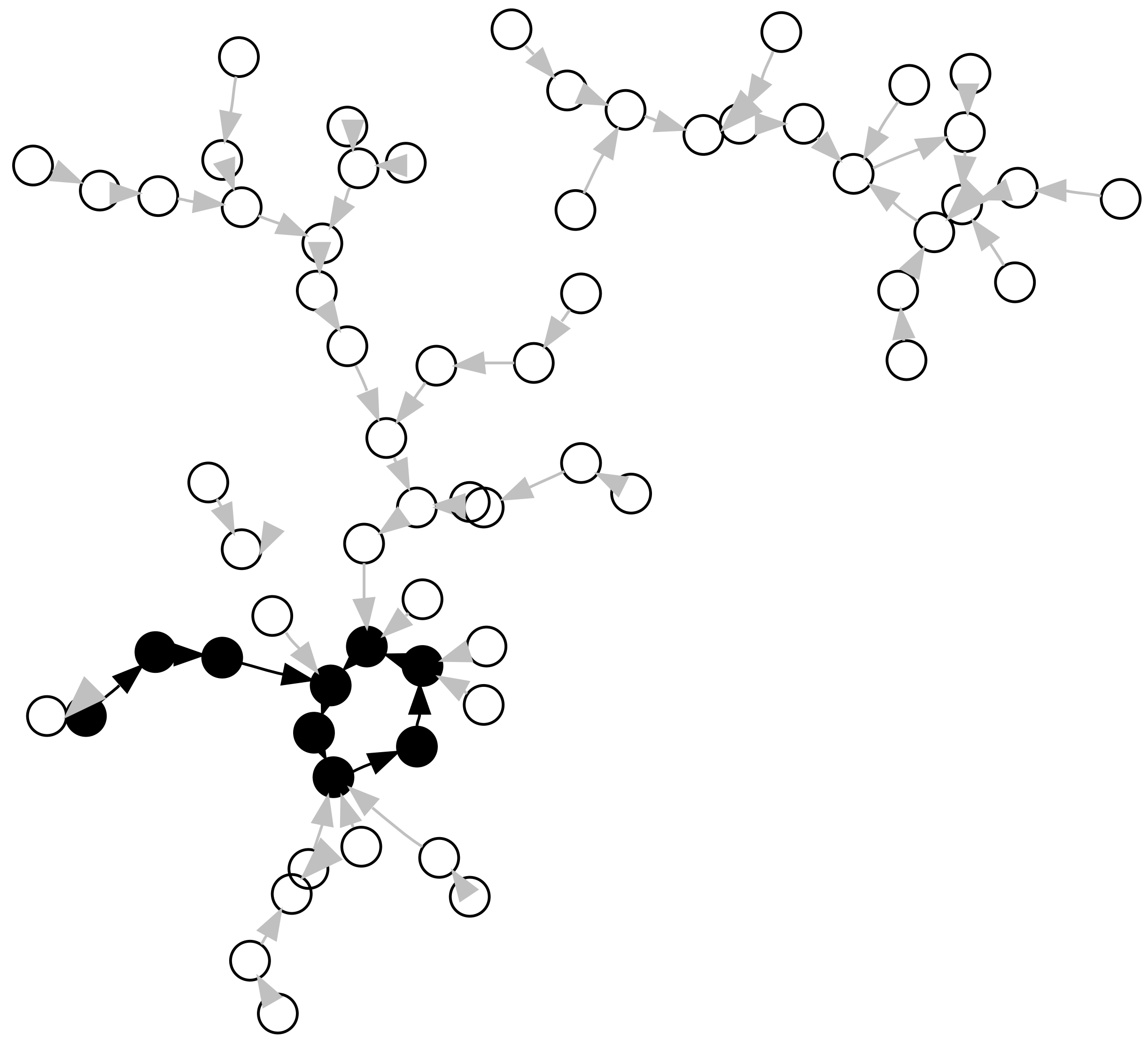
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f(W_i)$.

Birthday paradox:
Randomly choosing from ℓ
elements picks one element twice
after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm
(e.g., Floyd) quickly detects this.



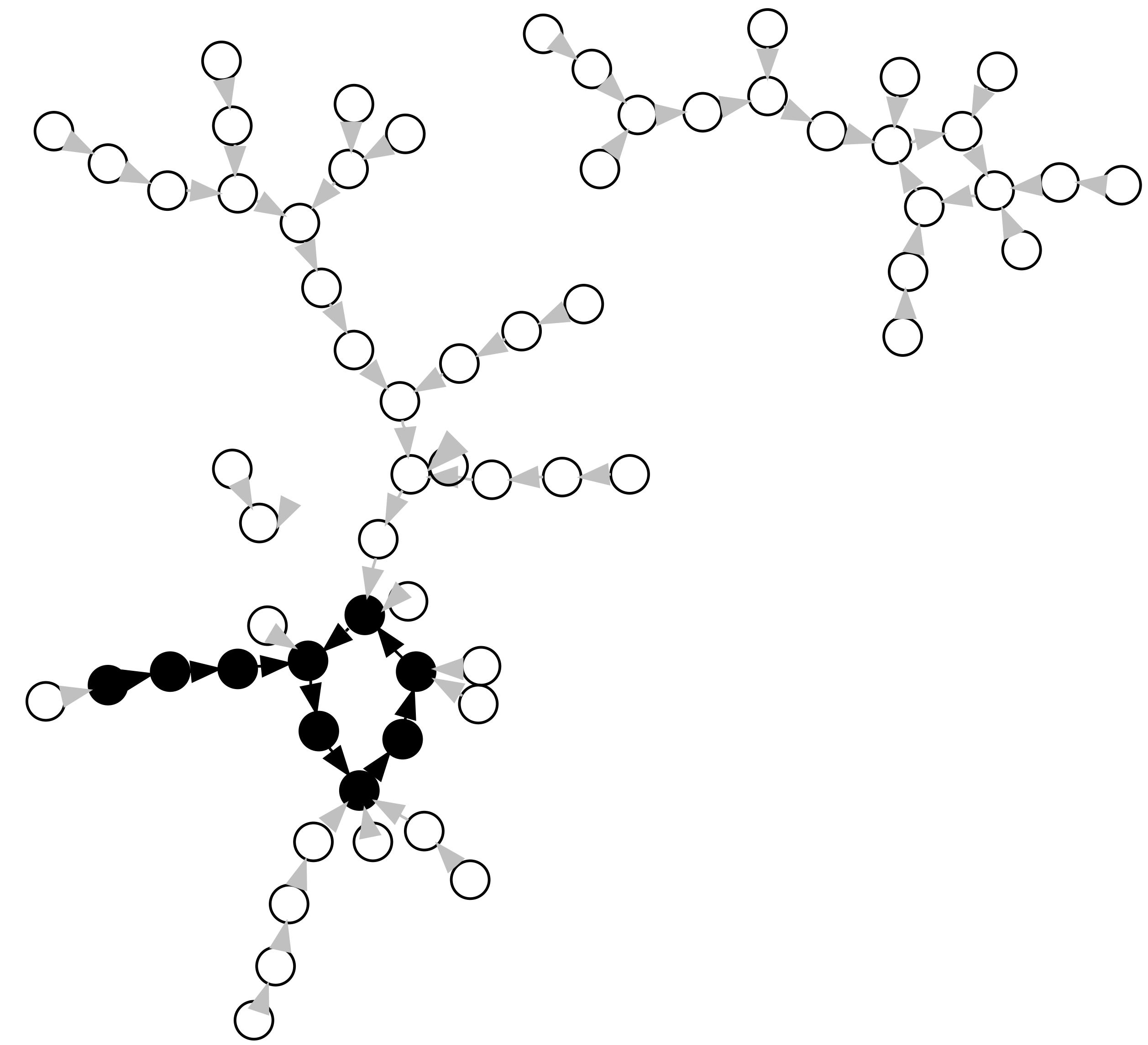
The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f(W_i)$.

Birthday paradox:
Randomly choosing from ℓ
elements picks one element twice
after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm
(e.g., Floyd) quickly detects this.



method

ed, non-parallel rho:

pseudo-random walk

roup $\langle P \rangle$,

he next step depends

nt point: $W_{i+1} = f(W_i)$.

paradox:

ly choosing from ℓ

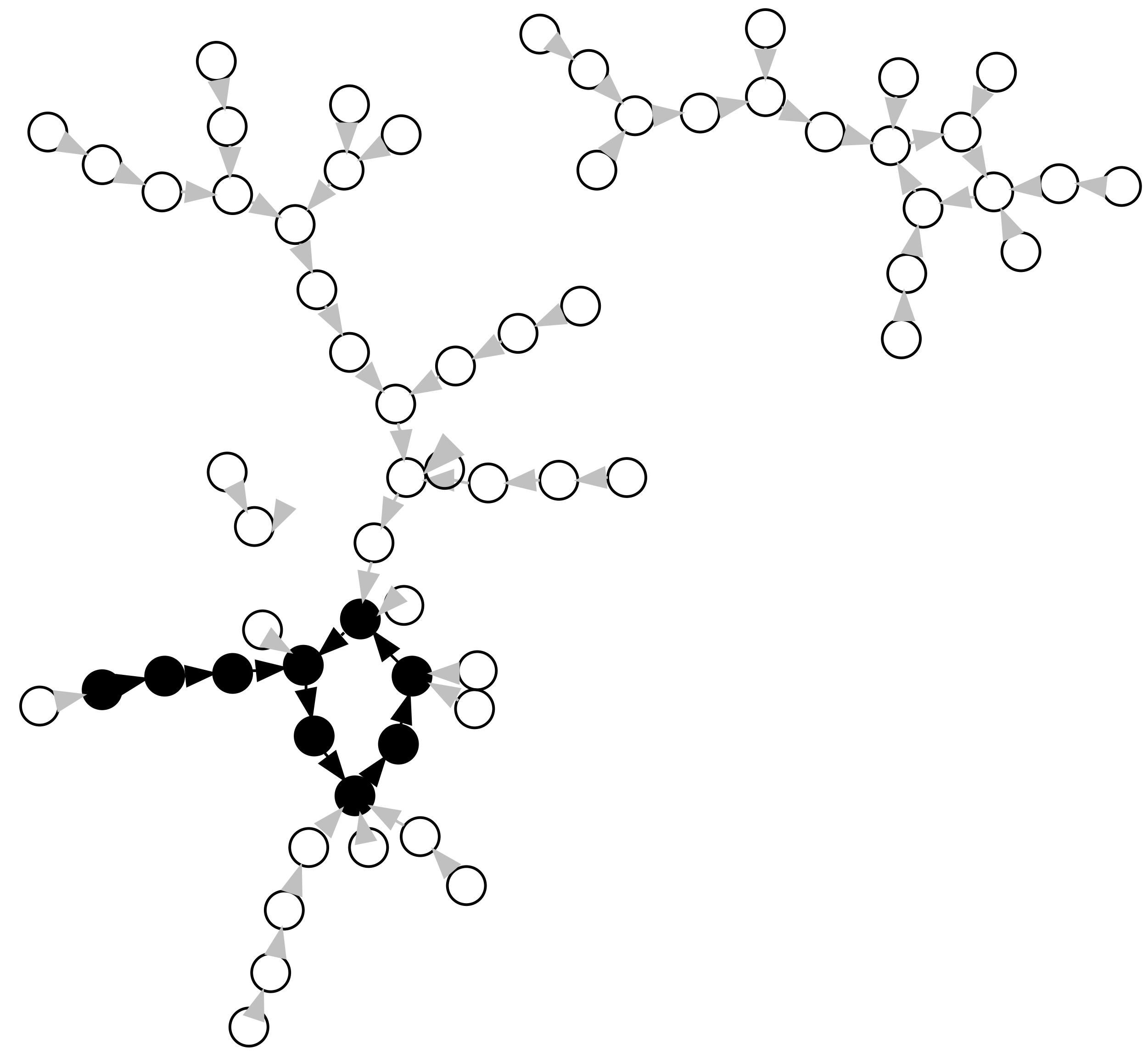
s picks one element twice

out $\sqrt{\pi\ell/2}$ draws.

k now enters a cycle.

nding algorithm

oyd) quickly detects this.



Assume

we know

so that

Then W

$a_i P + b$

so $(b_i -$

If $b_i \neq b$

$n = (a_j$

rallel rho:

ndom walk

ep depends

$W_{i+1} = f(W_i)$.

g from ℓ

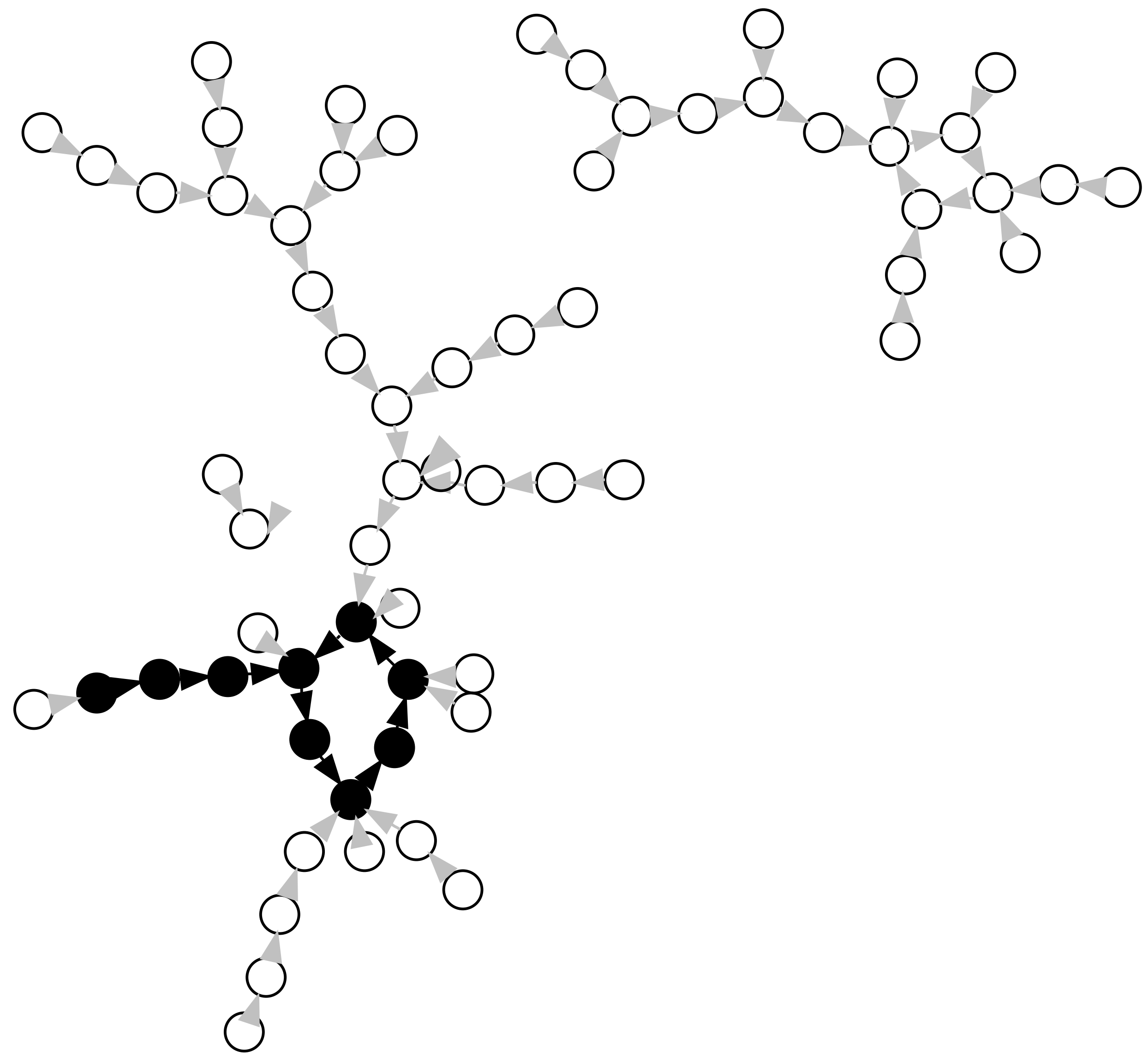
e element twice

2 draws.

ers a cycle.

rithm

kly detects this.



Assume that for ea

we know $a_i, b_i \in \mathbf{Z}$

so that $W_i = a_i P$

Then $W_i = W_j$ me

$a_i P + b_i Q = a_j P$

so $(b_i - b_j)Q = (a$

If $b_i \neq b_j$ the DLP

$n = (a_j - a_i)/(b_i$

x

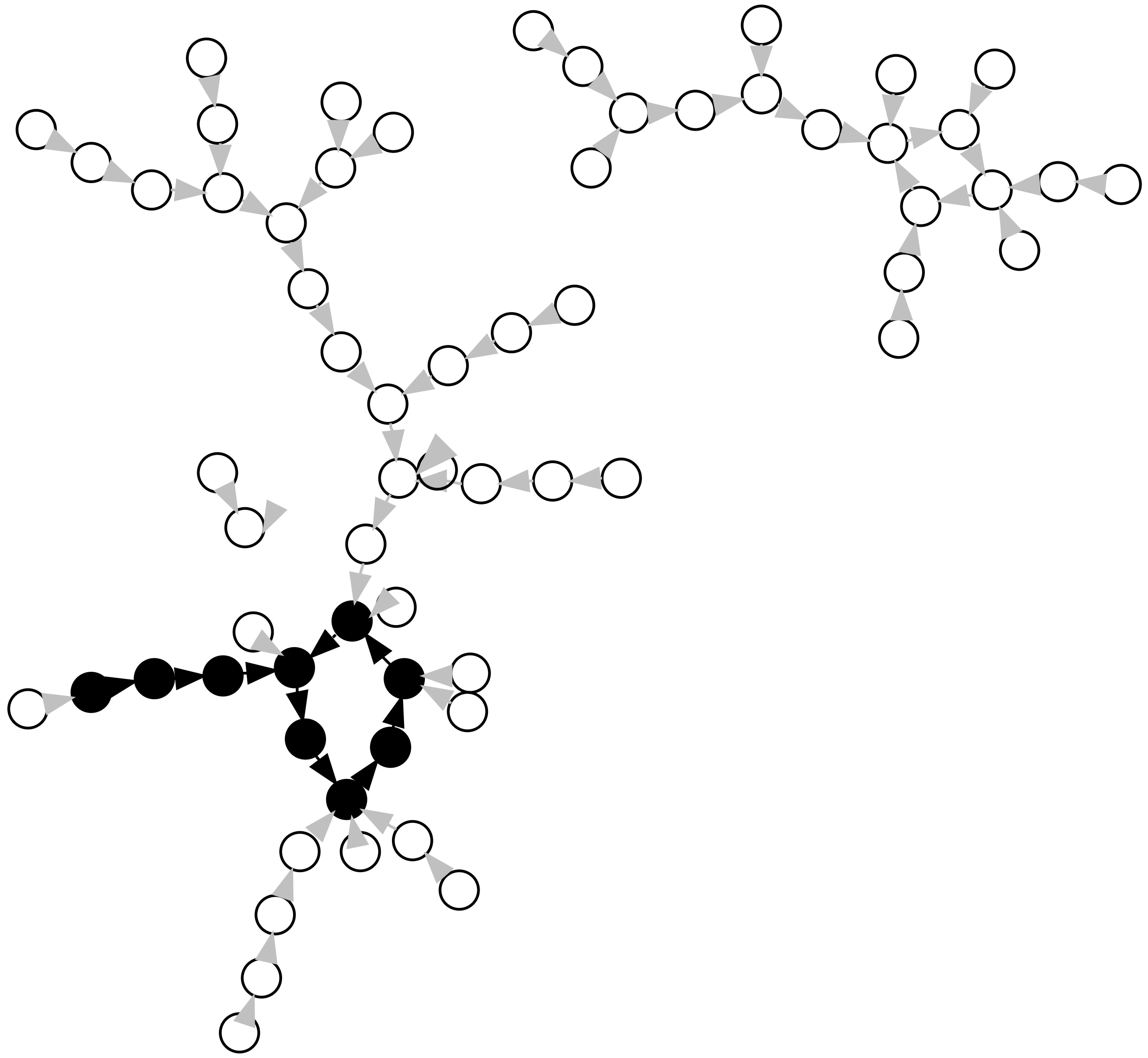
s

(W_i).

twice

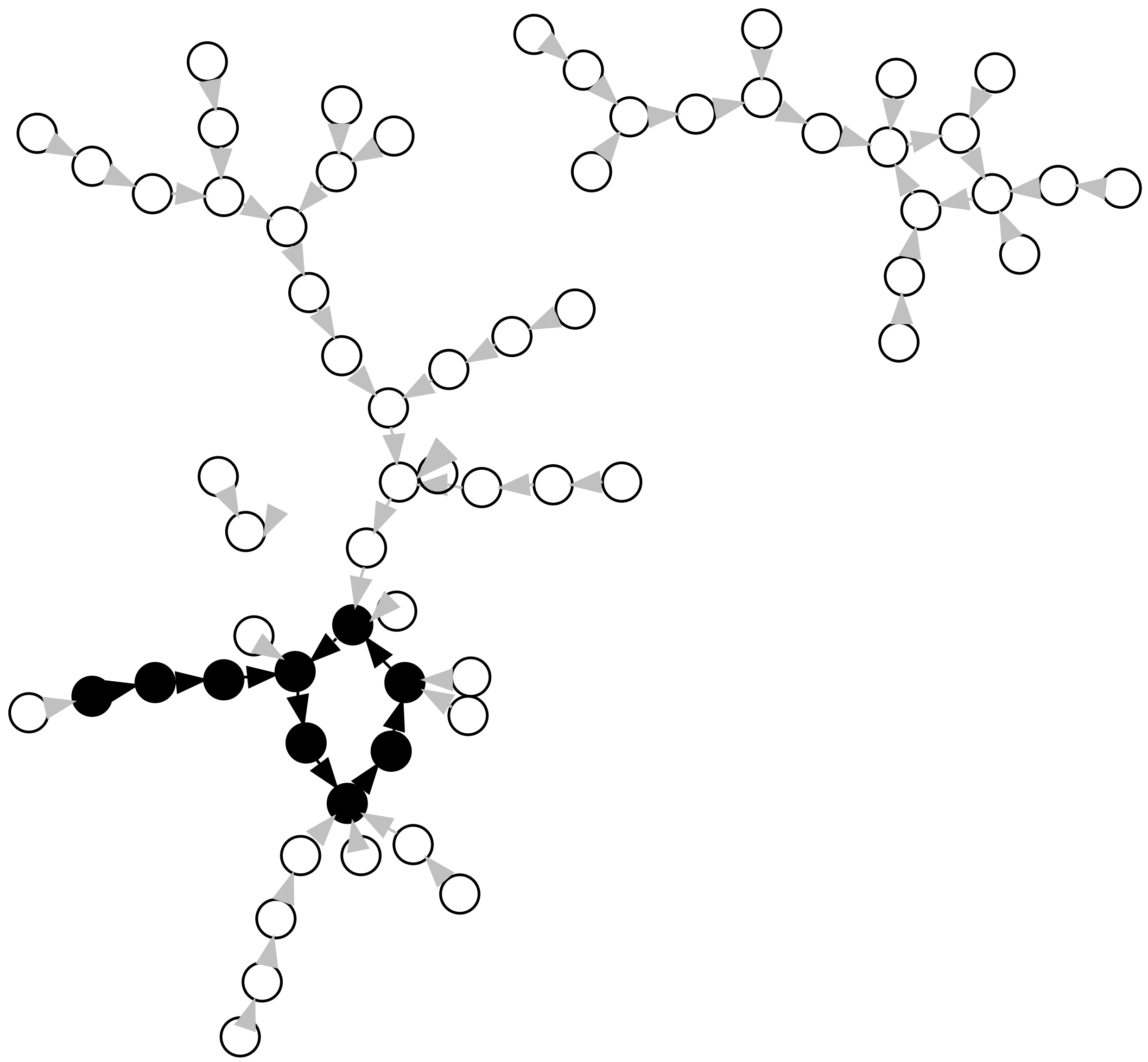
e.

this.



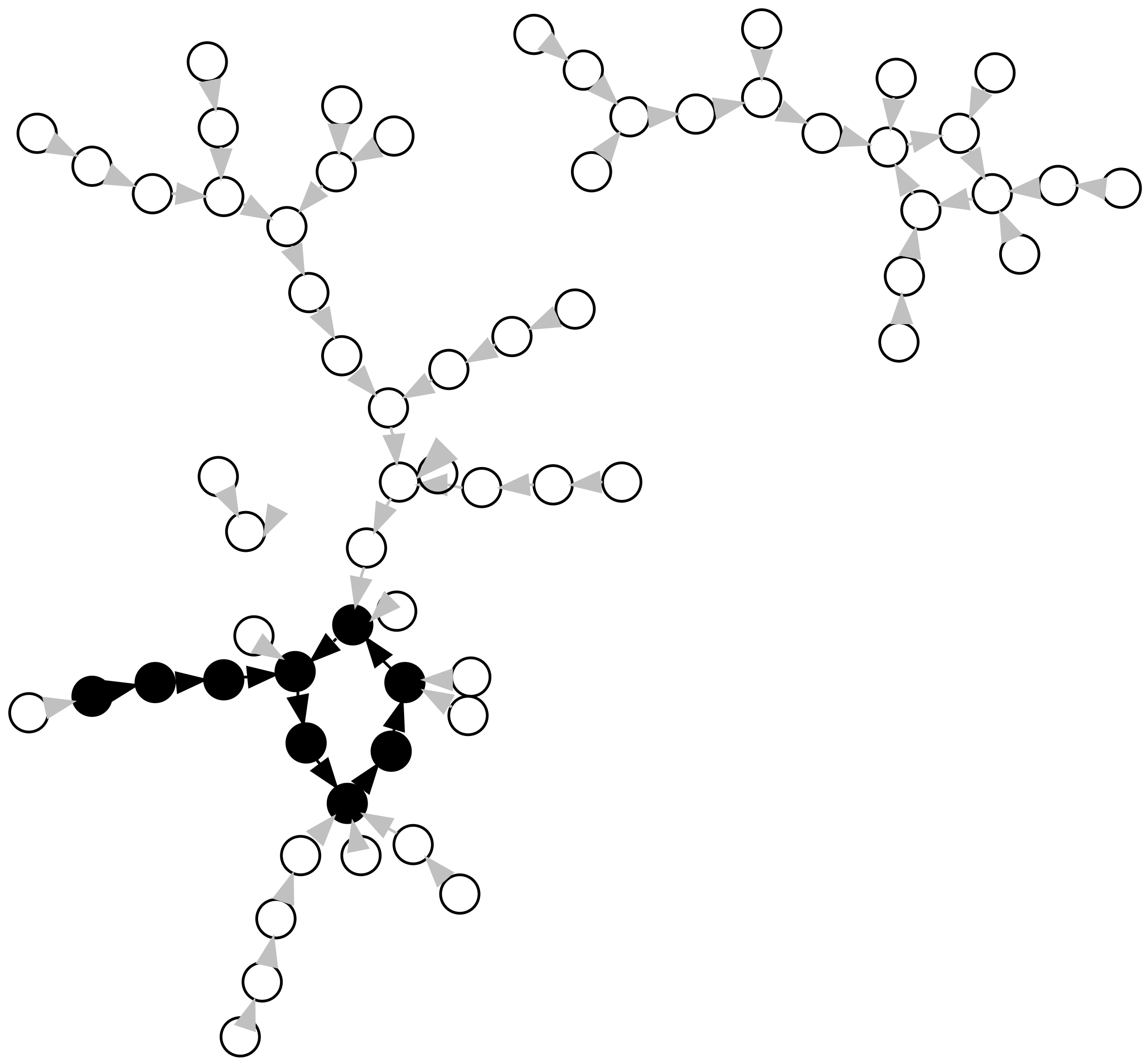
Assume that for each point
we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $W_i = a_i P + b_i Q$.

Then $W_i = W_j$ means that
$a_i P + b_i Q = a_j P + b_j Q$
so $(b_i - b_j)Q = (a_j - a_i)P$.
If $b_i \neq b_j$ the DLP is solved:
$n = (a_j - a_i)/(b_i - b_j)$.



Assume that for each point we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $W_i = a_i P + b_i Q$.

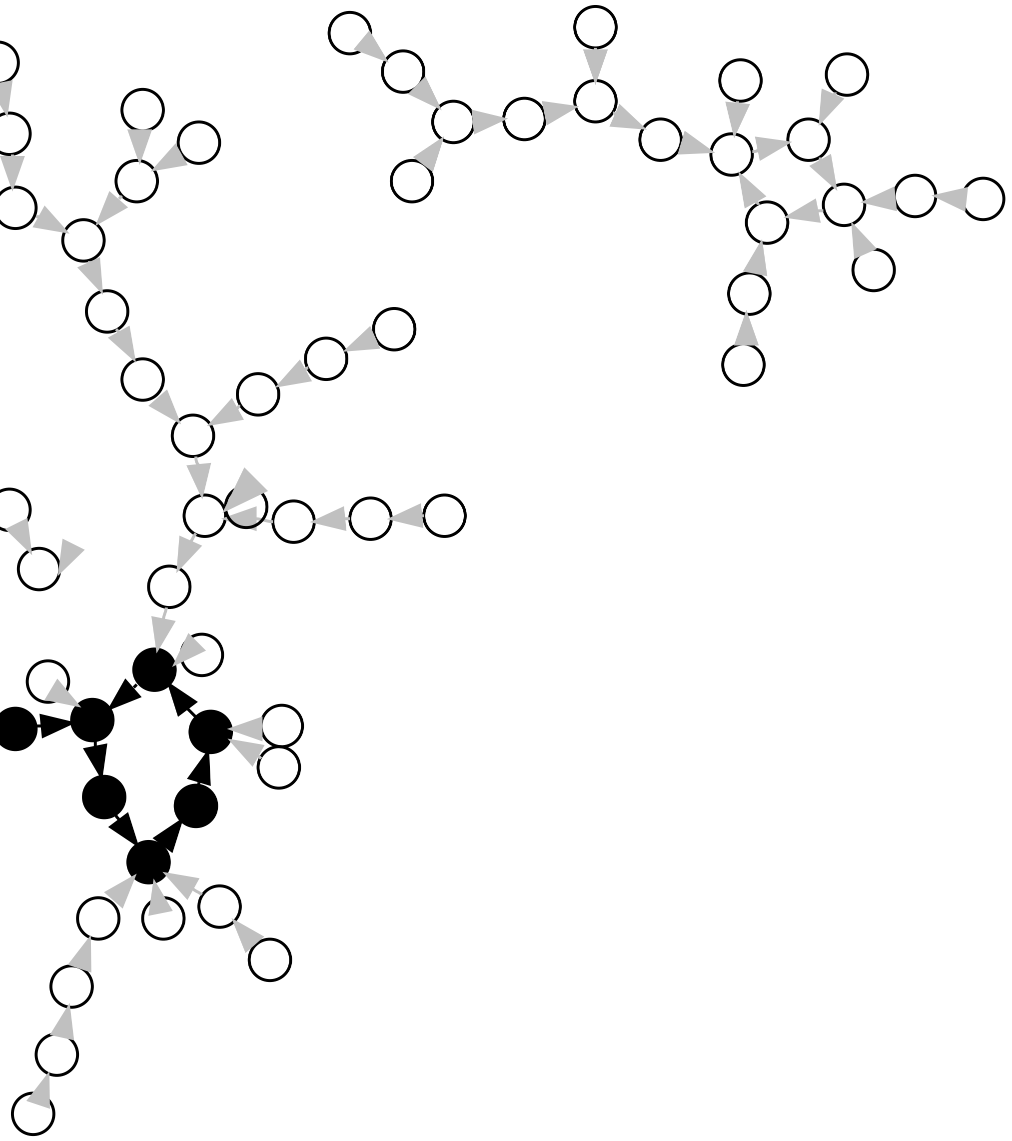
Then $W_i = W_j$ means that $a_i P + b_i Q = a_j P + b_j Q$ so $(b_i - b_j)Q = (a_j - a_i)P$. If $b_i \neq b_j$ the DLP is solved: $n = (a_j - a_i)/(b_i - b_j)$.



Assume that for each point we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $W_i = a_i P + b_i Q$.

Then $W_i = W_j$ means that $a_i P + b_i Q = a_j P + b_j Q$ so $(b_i - b_j)Q = (a_j - a_i)P$. If $b_i \neq b_j$ the DLP is solved: $n = (a_j - a_i)/(b_i - b_j)$.

e.g. $f(W_i) = a(W_i)P + b(W_i)Q$, starting from some initial combination $W_0 = a_0 P + b_0 Q$. If any W_i and W_j collide then $W_{i+1} = W_{j+1}$, $W_{i+2} = W_{j+2}$, etc.



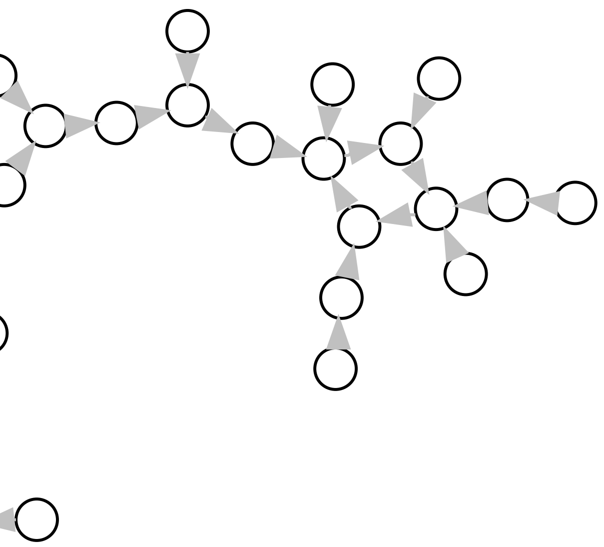
Assume that for each point we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $W_i = a_i P + b_i Q$.

Then $W_i = W_j$ means that $a_i P + b_i Q = a_j P + b_j Q$ so $(b_i - b_j)Q = (a_j - a_i)P$. If $b_i \neq b_j$ the DLP is solved: $n = (a_j - a_i)/(b_i - b_j)$.

e.g. $f(W_i) = a(W_i)P + b(W_i)Q$, starting from some initial combination $W_0 = a_0 P + b_0 Q$. If any W_i and W_j collide then $W_{i+1} = W_{j+1}$, $W_{i+2} = W_{j+2}$, etc.

If functi
random
perform
If a and
$f(W_i) =$
is define
under $\pm$
There ar
classes.
number
of almos

In gener
can be c
compute
small orc



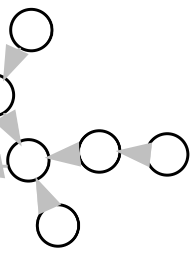
Assume that for each point we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $W_i = a_i P + b_i Q$.

Then $W_i = W_j$ means that $a_i P + b_i Q = a_j P + b_j Q$ so $(b_i - b_j)Q = (a_j - a_i)P$. If $b_i \neq b_j$ the DLP is solved: $n = (a_j - a_i)/(b_i - b_j)$.

e.g. $f(W_i) = a(W_i)P + b(W_i)Q$, starting from some initial combination $W_0 = a_0 P + b_0 Q$. If any W_i and W_j collide then $W_{i+1} = W_{j+1}$, $W_{i+2} = W_{j+2}$, etc.

If functions $a(W)$
random modulo ℓ,
perform a random
If a and b are chos
$f(W_i) = f(-W_i)$
is defined on *equiv*
under $\pm$.
There are only $\lceil \ell/$
classes. This redu
number of iteratio
of almost exactly

In general, Pollard
can be combined w
computed group a
small order.



Assume that for each point we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $W_i = a_i P + b_i Q$.

Then $W_i = W_j$ means that $a_i P + b_i Q = a_j P + b_j Q$ so $(b_i - b_j)Q = (a_j - a_i)P$. If $b_i \neq b_j$ the DLP is solved: $n = (a_j - a_i)/(b_i - b_j)$.

e.g. $f(W_i) = a(W_i)P + b(W_i)Q$, starting from some initial combination $W_0 = a_0 P + b_0 Q$. If any W_i and W_j collide then $W_{i+1} = W_{j+1}$, $W_{i+2} = W_{j+2}$, etc.

If functions $a(W)$ and $b(W)$
random modulo ℓ, iterations
perform a random walk in $\langle$
If a and b are chosen such t
$f(W_i) = f(-W_i)$ then the
is defined on *equivalence cla*
under $\pm$.
There are only $\lceil \ell/2 \rceil$ differe
classes. This reduces the av
number of iterations by a fa
of almost exactly $\sqrt{2}$.

In general, Pollard's rho met
can be combined with any e
computed group automorphi
small order.

Assume that for each point we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $W_i = a_i P + b_i Q$.

Then $W_i = W_j$ means that $a_i P + b_i Q = a_j P + b_j Q$ so $(b_i - b_j)Q = (a_j - a_i)P$. If $b_i \neq b_j$ the DLP is solved: $n = (a_j - a_i)/(b_i - b_j)$.

e.g. $f(W_i) = a(W_i)P + b(W_i)Q$, starting from some initial combination $W_0 = a_0 P + b_0 Q$. If any W_i and W_j collide then $W_{i+1} = W_{j+1}$, $W_{i+2} = W_{j+2}$, etc.

If functions $a(W)$ and $b(W)$ are random modulo ℓ, iterations perform a random walk in $\langle P \rangle$. If a and b are chosen such that $f(W_i) = f(-W_i)$ then the walk is defined on *equivalence classes* under $\pm$. There are only $\lceil \ell/2 \rceil$ different classes. This reduces the average number of iterations by a factor of almost exactly $\sqrt{2}$.

In general, Pollard's rho method can be combined with any easily computed group automorphism of small order.

that for each point
$a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$
$W_i = a_i P + b_i Q$.

$_i = W_j$ means that
$_i Q = a_j P + b_j Q$
$b_j)Q = (a_j - a_i)P$.
$_j$ the DLP is solved:
$- a_i)/(b_i - b_j)$.

$W_i) = a(W_i)P + b(W_i)Q$,
from some initial
tion $W_0 = a_0 P + b_0 Q$.
$_i$ and W_j collide then
$W_{j+1}, W_{i+2} = W_{j+2}$,

If functions $a(W)$ and $b(W)$ are random modulo ℓ, iterations perform a random walk in $\langle P \rangle$.
If a and b are chosen such that $f(W_i) = f(-W_i)$ then the walk is defined on *equivalence classes* under $\pm$.
There are only $\lceil \ell/2 \rceil$ different classes. This reduces the average number of iterations by a factor of almost exactly $\sqrt{2}$.

In general, Pollard's rho method can be combined with any easily computed group automorphism of small order.

Parallel

Running
N comp
$\approx \sqrt{N}$ f
of findin

Want be
computa
to find c
on *differ*
frequent

Better m
Oorscho
Declare
be *distin*

ach point

$\mathbf{Z}/\ell\mathbf{Z}$

$+ b_i Q$.

eans that

$+ b_j Q$

$a_j - a_i)P$.

is solved:

$- b_j)$.

$_i)P + b(W_i)Q$,

e initial

$= a_0 P + b_0 Q$.

collide then

$_{i+2} = W_{j+2}$,

If functions $a(W)$ and $b(W)$ are random modulo ℓ, iterations perform a random walk in $\langle P \rangle$.
If a and b are chosen such that $f(W_i) = f(-W_i)$ then the walk is defined on *equivalence classes* under $\pm$.
There are only $\lceil \ell/2 \rceil$ different classes. This reduces the average number of iterations by a factor of almost exactly $\sqrt{2}$.

In general, Pollard's rho method can be combined with any easily computed group automorphism of small order.

Parallel collision se

Running Pollard's
N computers give
$\approx \sqrt{N}$ from increa
of finding collision

Want better way t
computation acros
to find collisions b
on *different* machi
frequent synchroni

Better method du
Oorschot and Wie
Declare some subs
be *distinguished p*

$W_i)Q$,

Q.

en

$i+2$,

If functions $a(W)$ and $b(W)$ are random modulo ℓ, iterations perform a random walk in $\langle P \rangle$.
If a and b are chosen such that $f(W_i) = f(-W_i)$ then the walk is defined on *equivalence classes* under $\pm$.
There are only $\lceil \ell/2 \rceil$ different classes. This reduces the average number of iterations by a factor of almost exactly $\sqrt{2}$.

In general, Pollard's rho method can be combined with any easily computed group automorphism of small order.

Parallel collision search

Running Pollard's rho metho
N computers gives speedup
$\approx \sqrt{N}$ from increased likelih
of finding collision.

Want better way to spread
computation across clients.
to find collisions between wa
on *different* machines, witho
frequent synchronization!

Better method due to van
Oorschot and Wiener (1999)
Declare some subset of $\langle P \rangle$
be *distinguished points*.

If functions $a(W)$ and $b(W)$ are random modulo ℓ, iterations perform a random walk in $\langle P \rangle$. If a and b are chosen such that $f(W_i) = f(-W_i)$ then the walk is defined on *equivalence classes* under $\pm$.

There are only $\lceil \ell/2 \rceil$ different classes. This reduces the average number of iterations by a factor of almost exactly $\sqrt{2}$.

In general, Pollard's rho method can be combined with any easily computed group automorphism of small order.

Parallel collision search

Running Pollard's rho method on N computers gives speedup of $\approx \sqrt{N}$ from increased likelihood of finding collision.

Want better way to spread computation across clients. Want to find collisions between walks on *different* machines, without frequent synchronization!

Better method due to van Oorschot and Wiener (1999). Declare some subset of $\langle P \rangle$ to be *distinguished points*.

ons $a(W)$ and $b(W)$ are
modulo ℓ, iterations
a random walk in $\langle P \rangle$.
b are chosen such that
$= f(-W_i)$ then the walk
d on *equivalence classes*
.

re only $\lceil \ell/2 \rceil$ different
This reduces the average
of iterations by a factor
st exactly $\sqrt{2}$.

al, Pollard's rho method
combined with any easily
ed group automorphism of
der.

Parallel collision search

Running Pollard's rho method on N computers gives speedup of $\approx \sqrt{N}$ from increased likelihood of finding collision.

Want better way to spread computation across clients. Want to find collisions between walks on *different* machines, without frequent synchronization!

Better method due to van Oorschot and Wiener (1999). Declare some subset of $\langle P \rangle$ to be *distinguished points*.

Parallel
with diff
but sam
If two di
find the
their sub

Termina
a disting
the poin
server.
Server re
all distin
Two wal
distingui
This coll

and $b(W)$ are
iterations
walk in $\langle P \rangle$.
sen such that
then the walk
valence classes

$/2\rceil$ different
ces the average
ns by a factor
$\sqrt{2}$.

's rho method
with any easily
utomorphism of

Parallel collision search

Running Pollard's rho method on N computers gives speedup of $\approx \sqrt{N}$ from increased likelihood of finding collision.

Want better way to spread computation across clients. Want to find collisions between walks on *different* machines, without frequent synchronization!

Better method due to van Oorschot and Wiener (1999). Declare some subset of $\langle P \rangle$ to be *distinguished points*.

Parallel rho: Perfo
with different start
but same update f
If two different wa
find the same poin
their subsequent s

Terminate each wa
a distinguished po
the point along wi
server.
Server receives, sto
all distinguished p
Two walks reachin
distinguished poin
This collision solve

are
s
$P\rangle$.
hat
walk
asses

nt
erage
ctor

thod
asily
ism of

Parallel collision search

Running Pollard's rho method on N computers gives speedup of $\approx \sqrt{N}$ from increased likelihood of finding collision.

Want better way to spread computation across clients. Want to find collisions between walks on *different* machines, without frequent synchronization!

Better method due to van Oorschot and Wiener (1999). Declare some subset of $\langle P \rangle$ to be *distinguished points*.

Parallel rho: Perform many
with different starting points
but same update function f
If two different walks
find the same point then
their subsequent steps will n

Terminate each walk once it
a distinguished point and re
the point along with a_i and
server.
Server receives, stores, and s
all distinguished points.
Two walks reaching same
distinguished point give colli
This collision solves the DLF

Parallel collision search

Running Pollard's rho method on N computers gives speedup of $\approx \sqrt{N}$ from increased likelihood of finding collision.

Want better way to spread computation across clients. Want to find collisions between walks on *different* machines, without frequent synchronization!

Better method due to van Oorschot and Wiener (1999). Declare some subset of $\langle P \rangle$ to be *distinguished points*.

Parallel rho: Perform many walks with different starting points but same update function f. If two different walks find the same point then their subsequent steps will match.

Terminate each walk once it hits a distinguished point and report the point along with a_i and b_i to server. Server receives, stores, and sorts all distinguished points. Two walks reaching same distinguished point give collision. This collision solves the DLP.

collision search

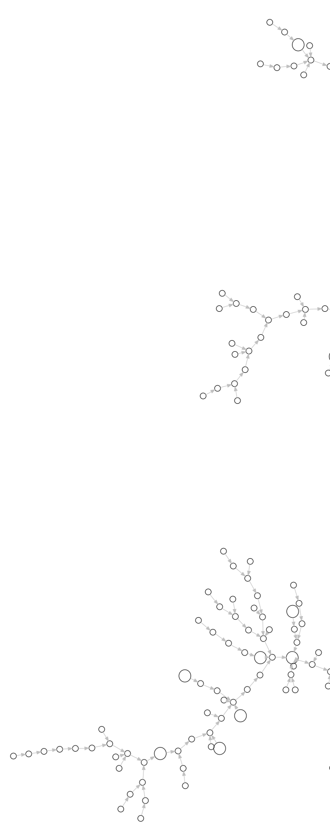
Pollard's rho method on
uters gives speedup of
from increased likelihood
g collision.

etter way to spread
ation across clients. Want
collisions between walks
rent machines, without
synchronization!

method due to van
t and Wiener (1999).
some subset of $\langle P \rangle$ to
nguished points.

Parallel rho: Perform many walks
with different starting points
but same update function f.
If two different walks
find the same point then
their subsequent steps will match.

Terminate each walk once it hits
a distinguished point and report
the point along with a_i and b_i to
server.
Server receives, stores, and sorts
all distinguished points.
Two walks reaching same
distinguished point give collision.
This collision solves the DLP.



earch

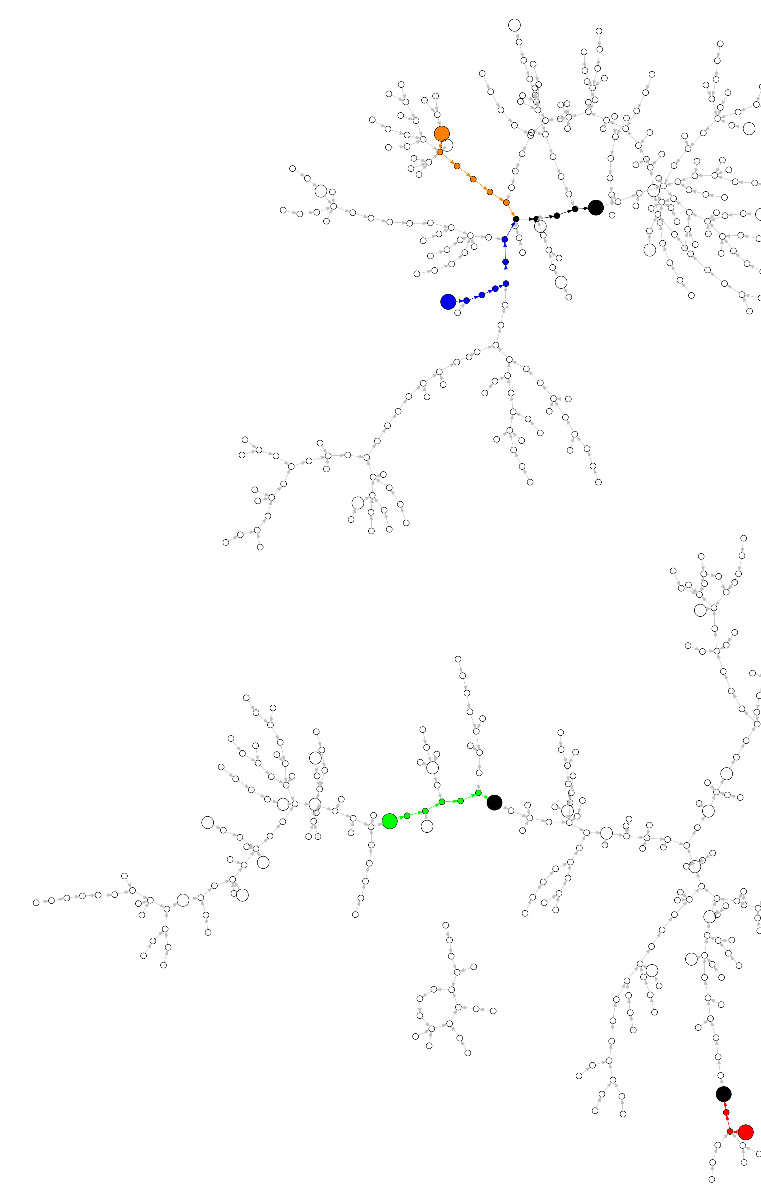
rho method on
s speedup of
ased likelihood
.

o spread
ss clients. Want
etween walks
nes, without
ization!

e to van
ner (1999).
set of $\langle P \rangle$ to
oints.

Parallel rho: Perform many walks
with different starting points
but same update function f.
If two different walks
find the same point then
their subsequent steps will match.

Terminate each walk once it hits
a distinguished point and report
the point along with a_i and b_i to
server.
Server receives, stores, and sorts
all distinguished points.
Two walks reaching same
distinguished point give collision.
This collision solves the DLP.



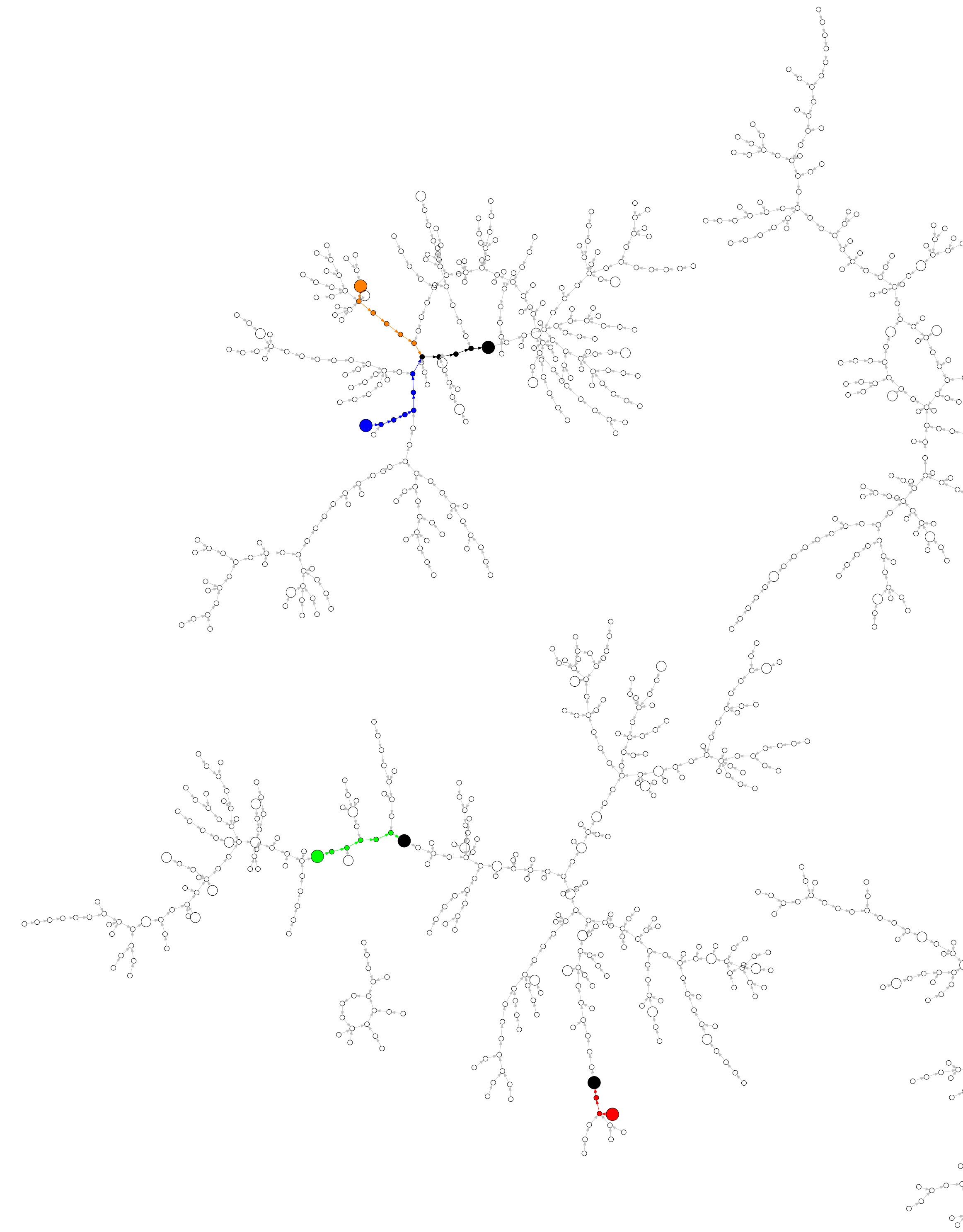
od on
of
hood

Want
alks
out

).
to

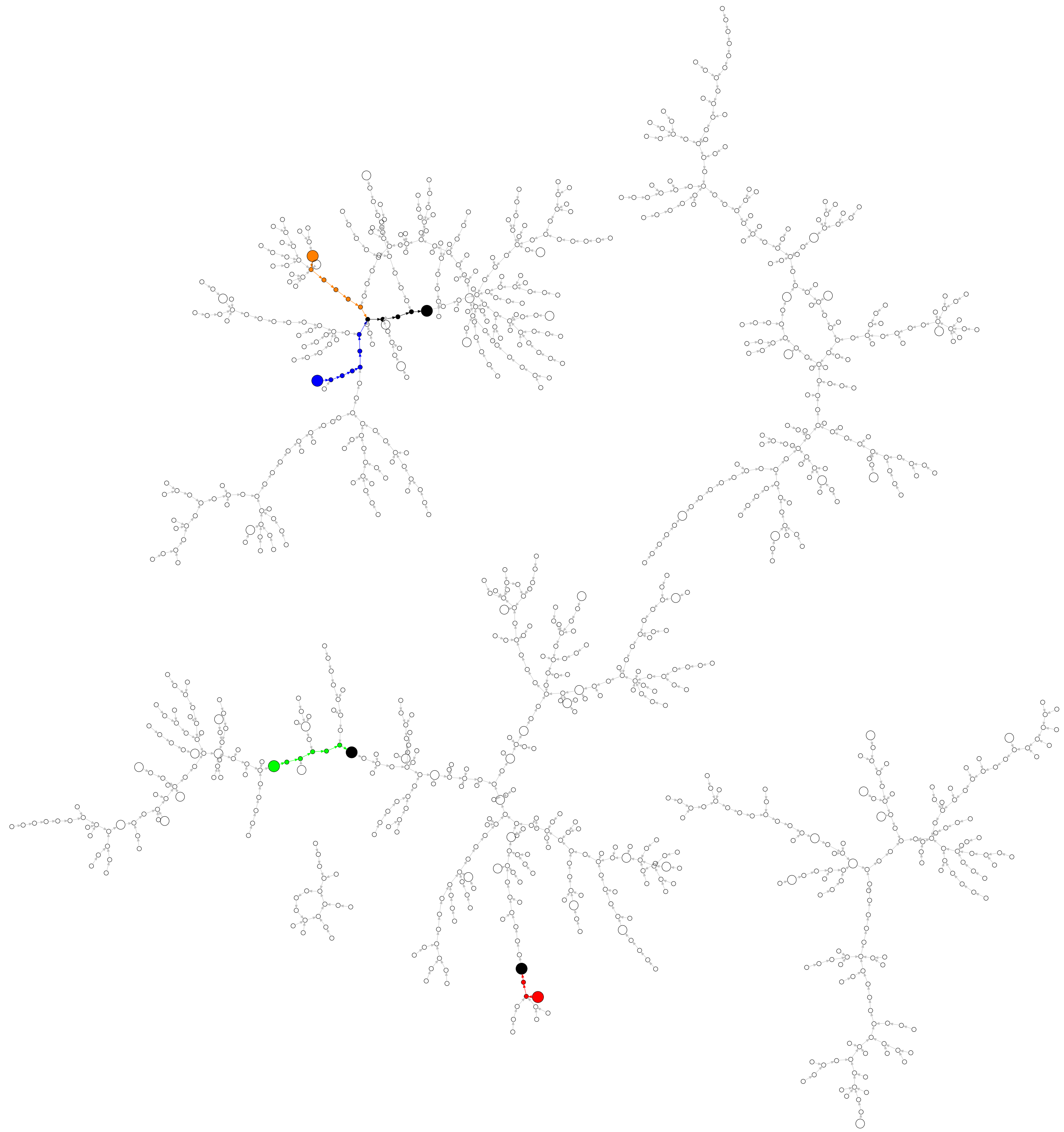
Parallel rho: Perform many walks with different starting points but same update function f.
If two different walks find the same point then their subsequent steps will match.

Terminate each walk once it hits a distinguished point and report the point along with a_i and b_i to server.
Server receives, stores, and sorts all distinguished points.
Two walks reaching same distinguished point give collision.
This collision solves the DLP.



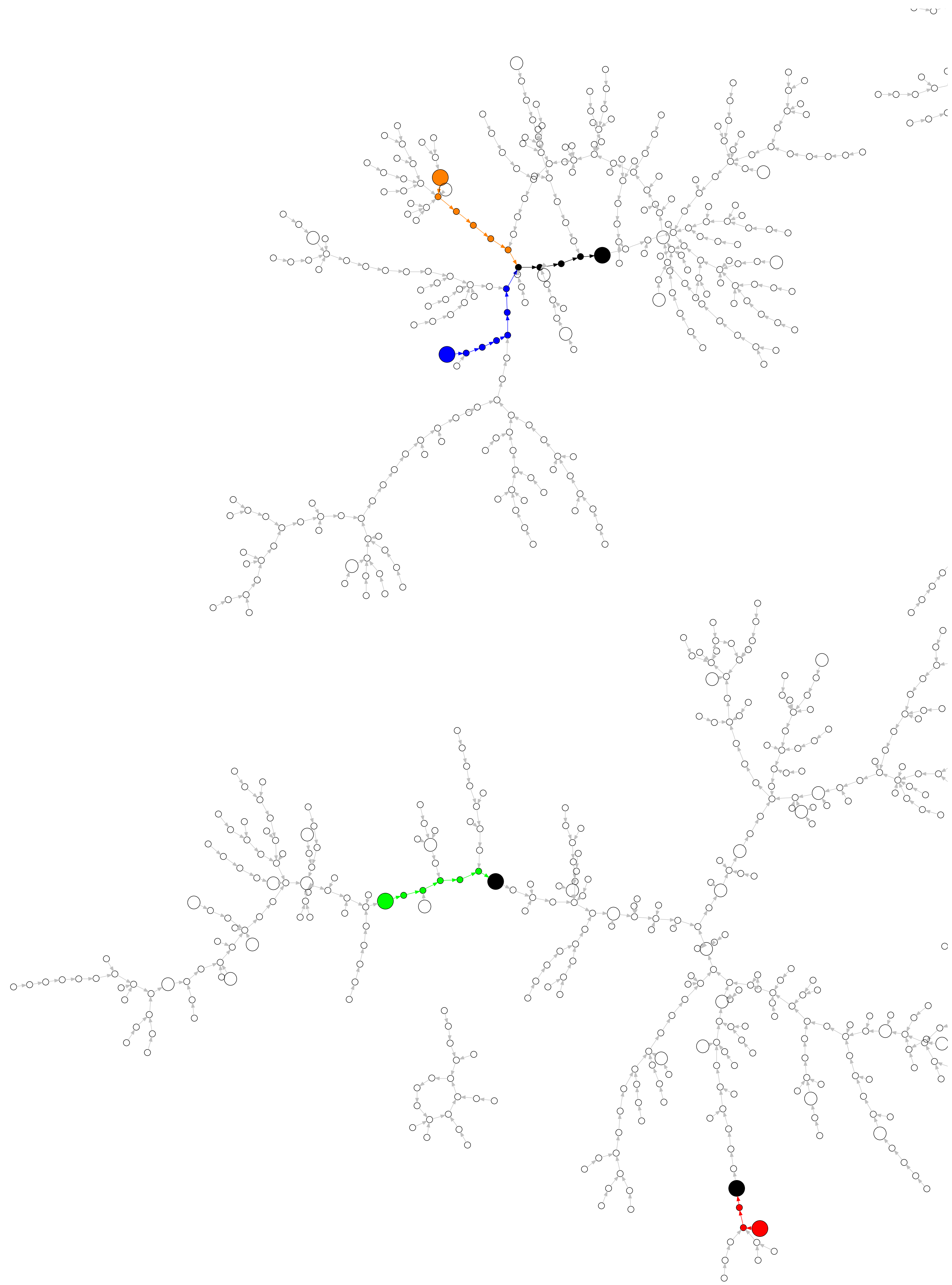
Parallel rho: Perform many walks with different starting points but same update function f.
If two different walks find the same point then their subsequent steps will match.

Terminate each walk once it hits a distinguished point and report the point along with a_i and b_i to server.
Server receives, stores, and sorts all distinguished points.
Two walks reaching same distinguished point give collision.
This collision solves the DLP.



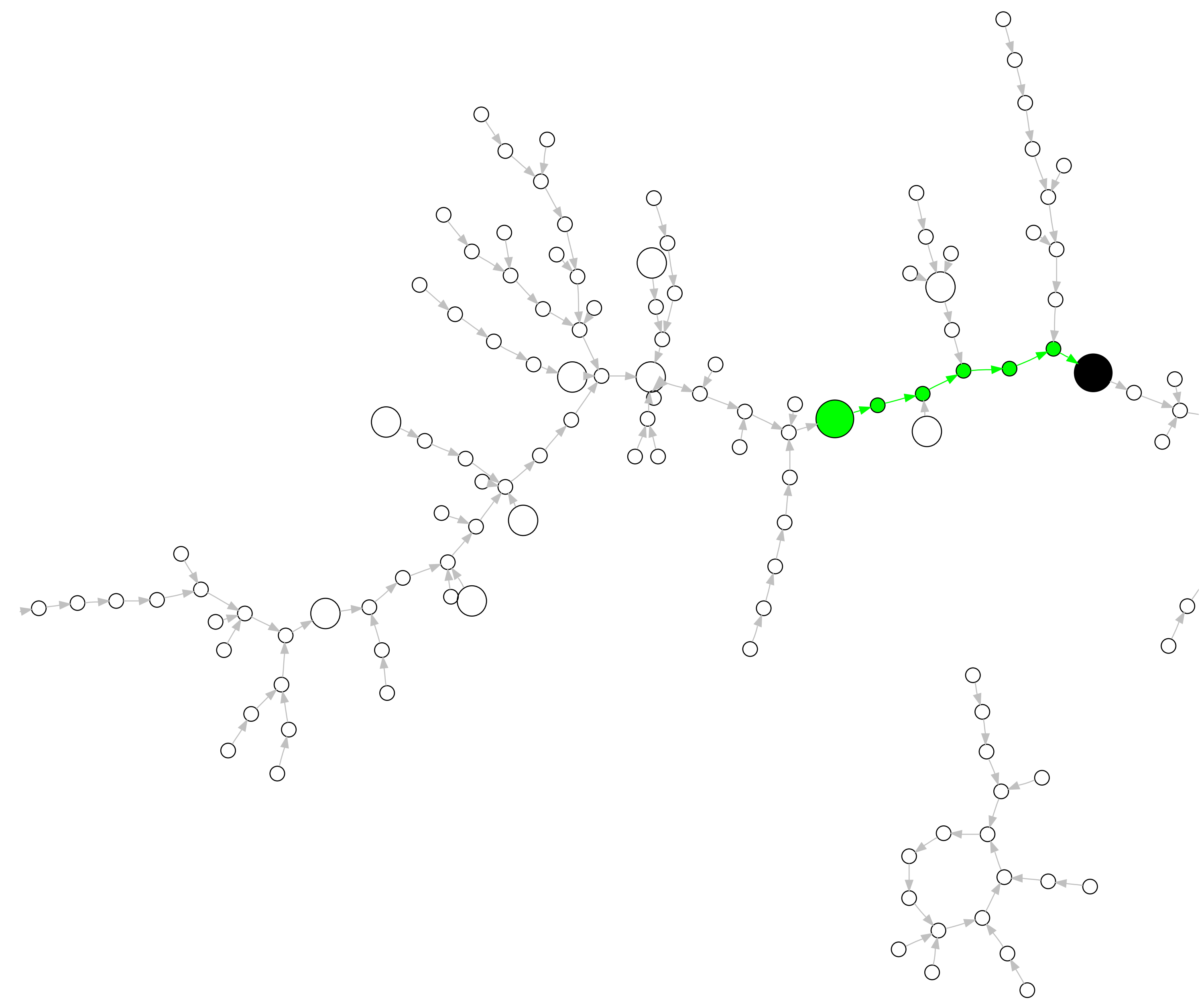
Parallel rho: Perform many walks with different starting points but same update function f.
If two different walks find the same point then their subsequent steps will match.

Terminate each walk once it hits a distinguished point and report the point along with a_i and b_i to server.
Server receives, stores, and sorts all distinguished points.
Two walks reaching same distinguished point give collision.
This collision solves the DLP.



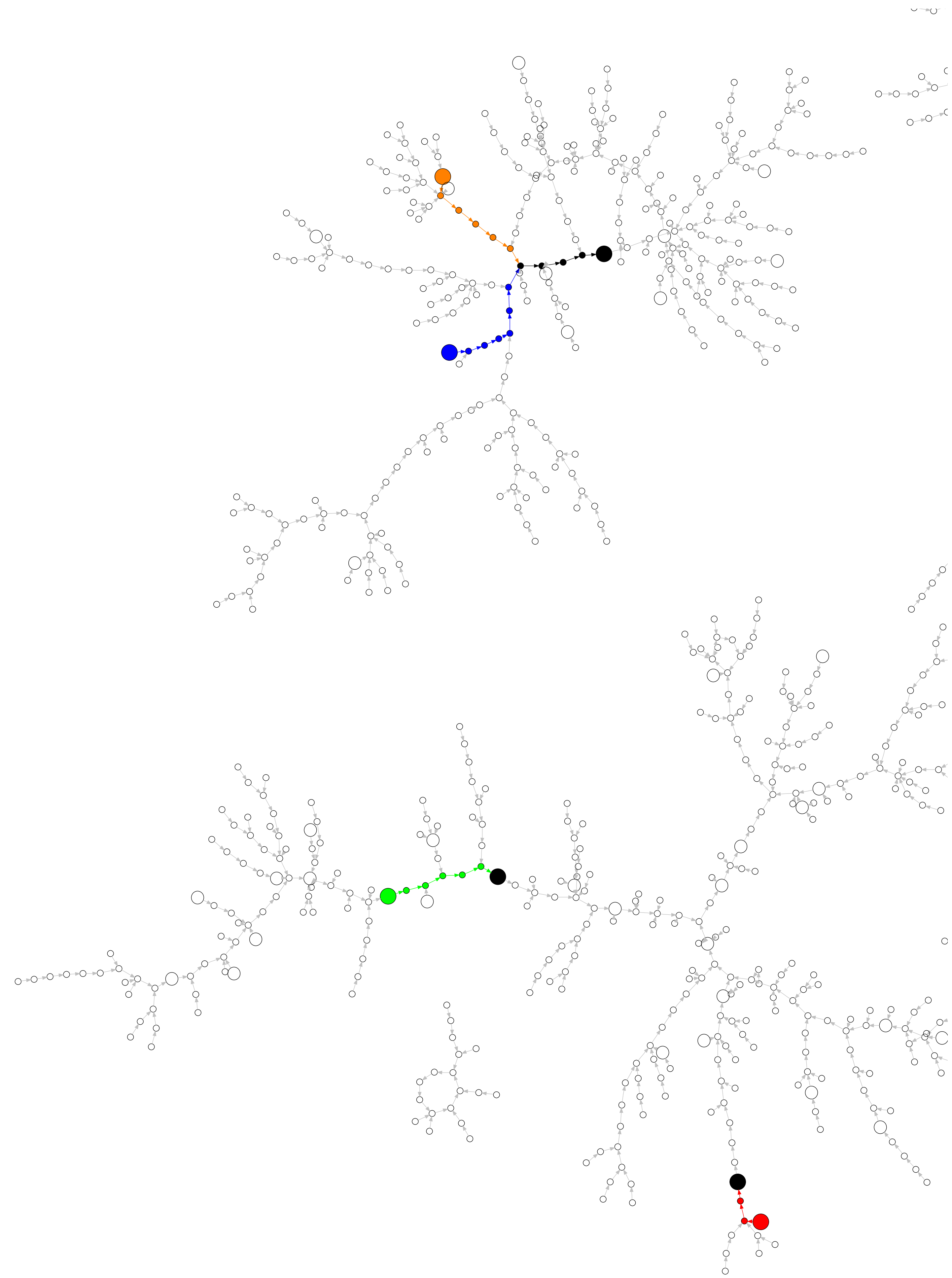
Parallel rho: Perform many walks with different starting points but same update function f.
If two different walks find the same point then their subsequent steps will match.

Terminate each walk once it hits a distinguished point and report the point along with a_i and b_i to server.
Server receives, stores, and sorts all distinguished points.
Two walks reaching same distinguished point give collision.
This collision solves the DLP.



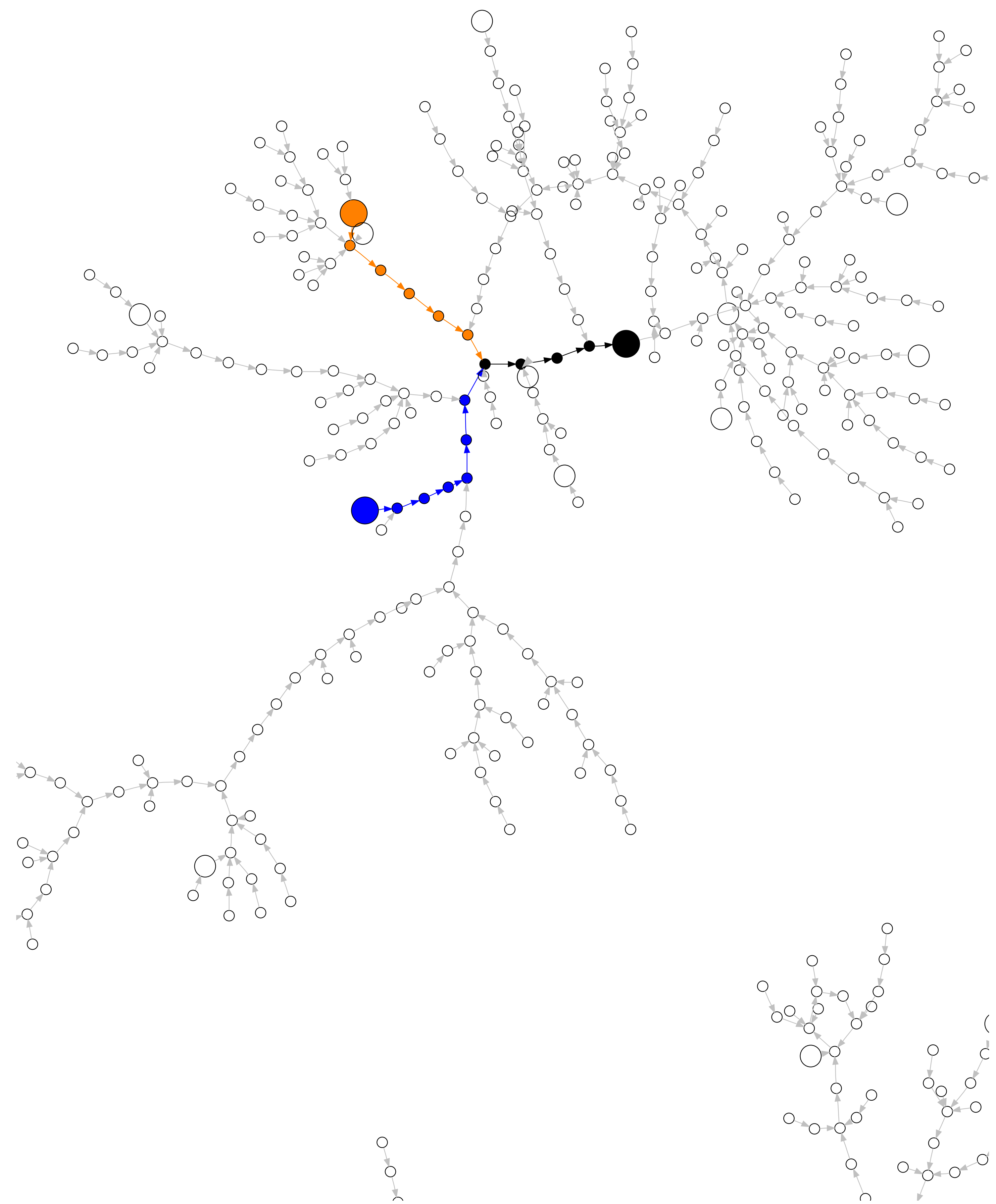
Parallel rho: Perform many walks with different starting points but same update function f.
If two different walks find the same point then their subsequent steps will match.

Terminate each walk once it hits a distinguished point and report the point along with a_i and b_i to server.
Server receives, stores, and sorts all distinguished points.
Two walks reaching same distinguished point give collision.
This collision solves the DLP.



Parallel rho: Perform many walks with different starting points but same update function f.
If two different walks find the same point then their subsequent steps will match.

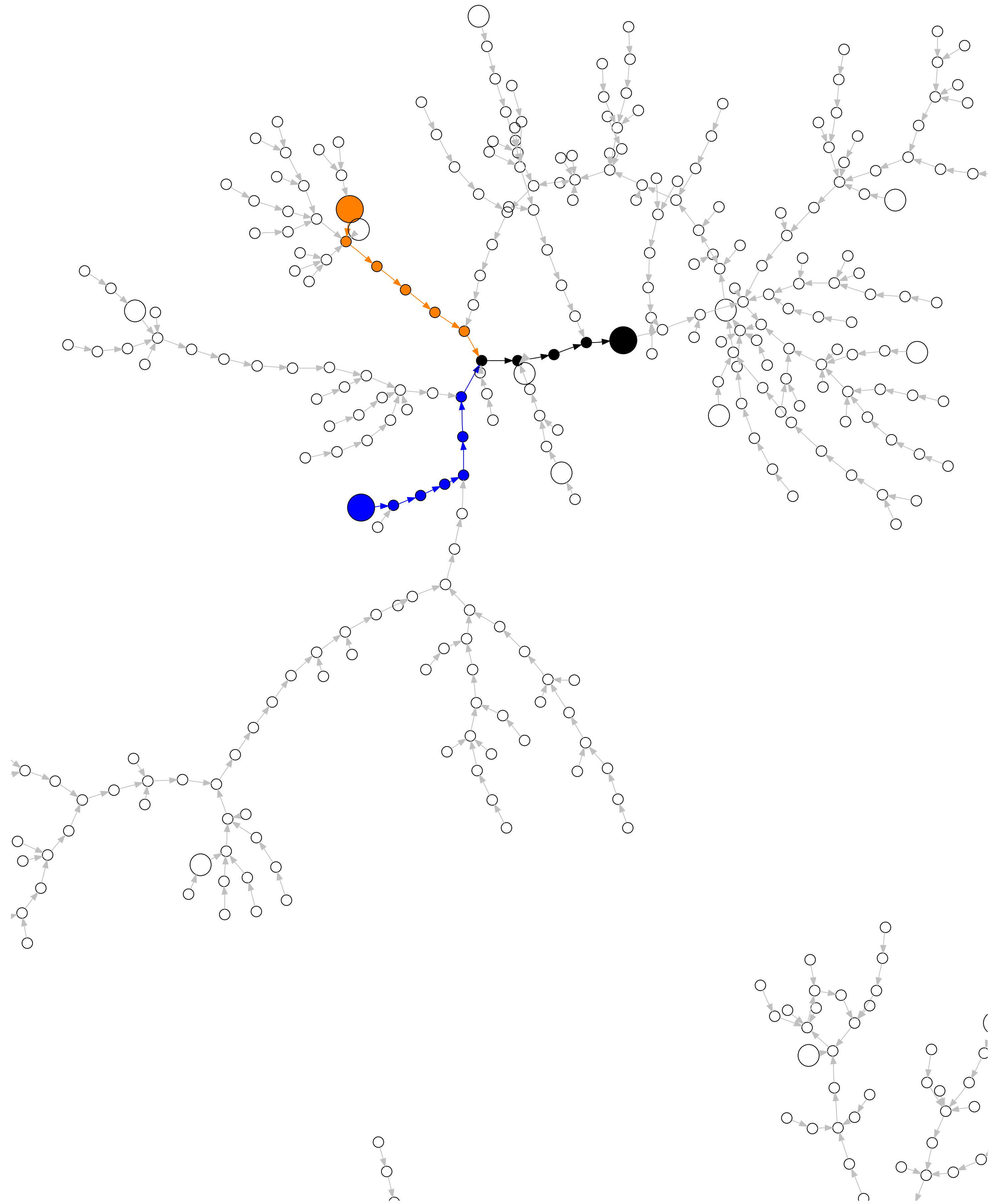
Terminate each walk once it hits a distinguished point and report the point along with a_i and b_i to server.
Server receives, stores, and sorts all distinguished points.
Two walks reaching same distinguished point give collision.
This collision solves the DLP.



rho: Perform many walks
ferent starting points
e update function f.
fferent walks
same point then
sequent steps will match.

te each walk once it hits
guished point and report
t along with a_i and b_i to

eceives, stores, and sorts
guished points.
lks reaching same
shed point give collision.
lision solves the DLP.

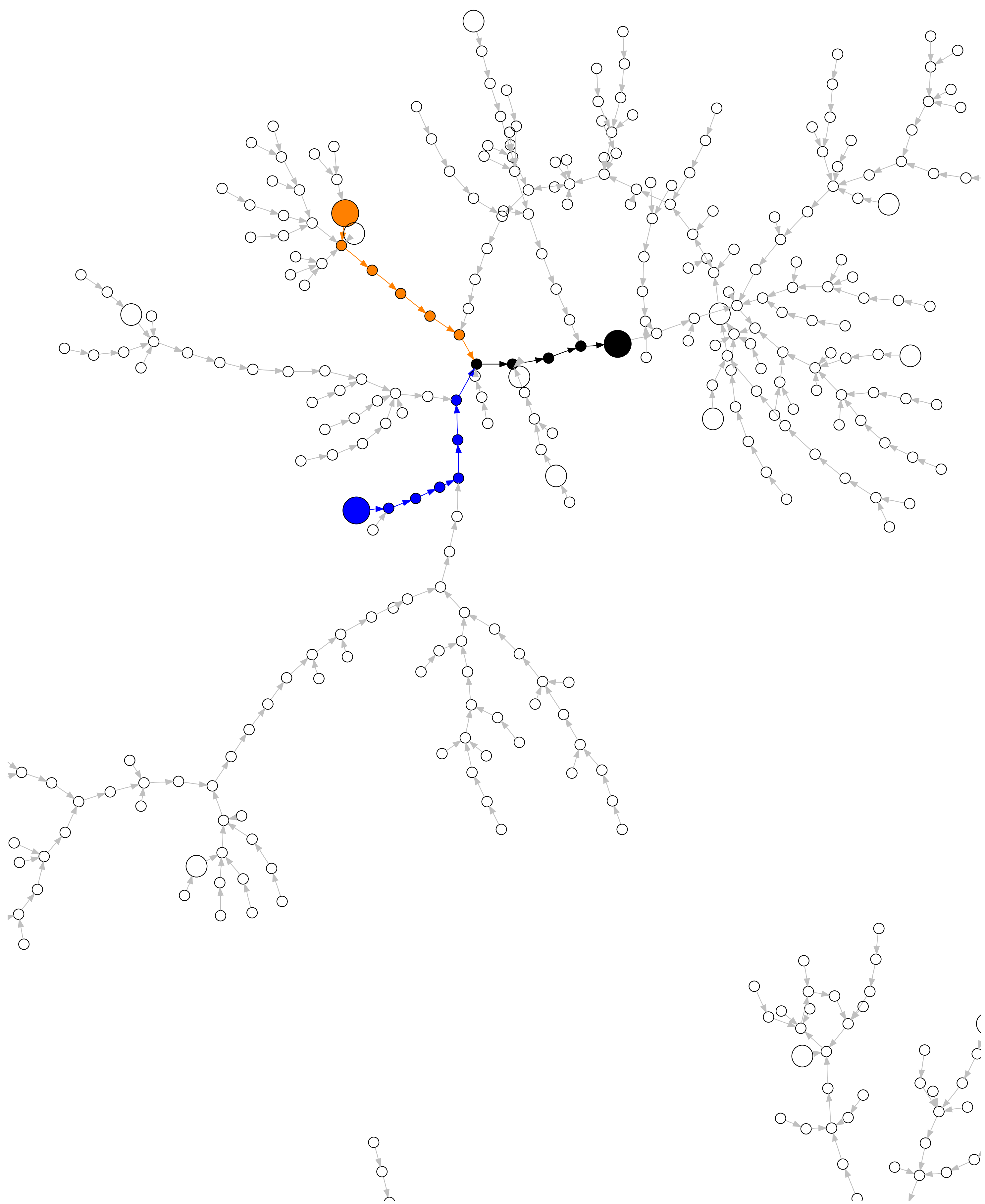


Attacker
definitio
Tradeoff
If disting
small nu
will be p
the num
points se
increases
collision
If disting
frequent
be perfo
In any c
Total #

orm many walks
ting points
function f.
lks
nt then
teps will match.

alk once it hits
int and report
th a_i and b_i to

ores, and sorts
oints.
ng same
t give collision.
es the DLP.



Attacker chooses f
definition of distin
Tradeoffs are poss
If distinguished po
small number of v
will be performed.
the number of dist
points sent to the
increases the delay
collision is recogni
If distinguished po
frequent, many sh
be performed.
In any case do not
Total # of iteratio

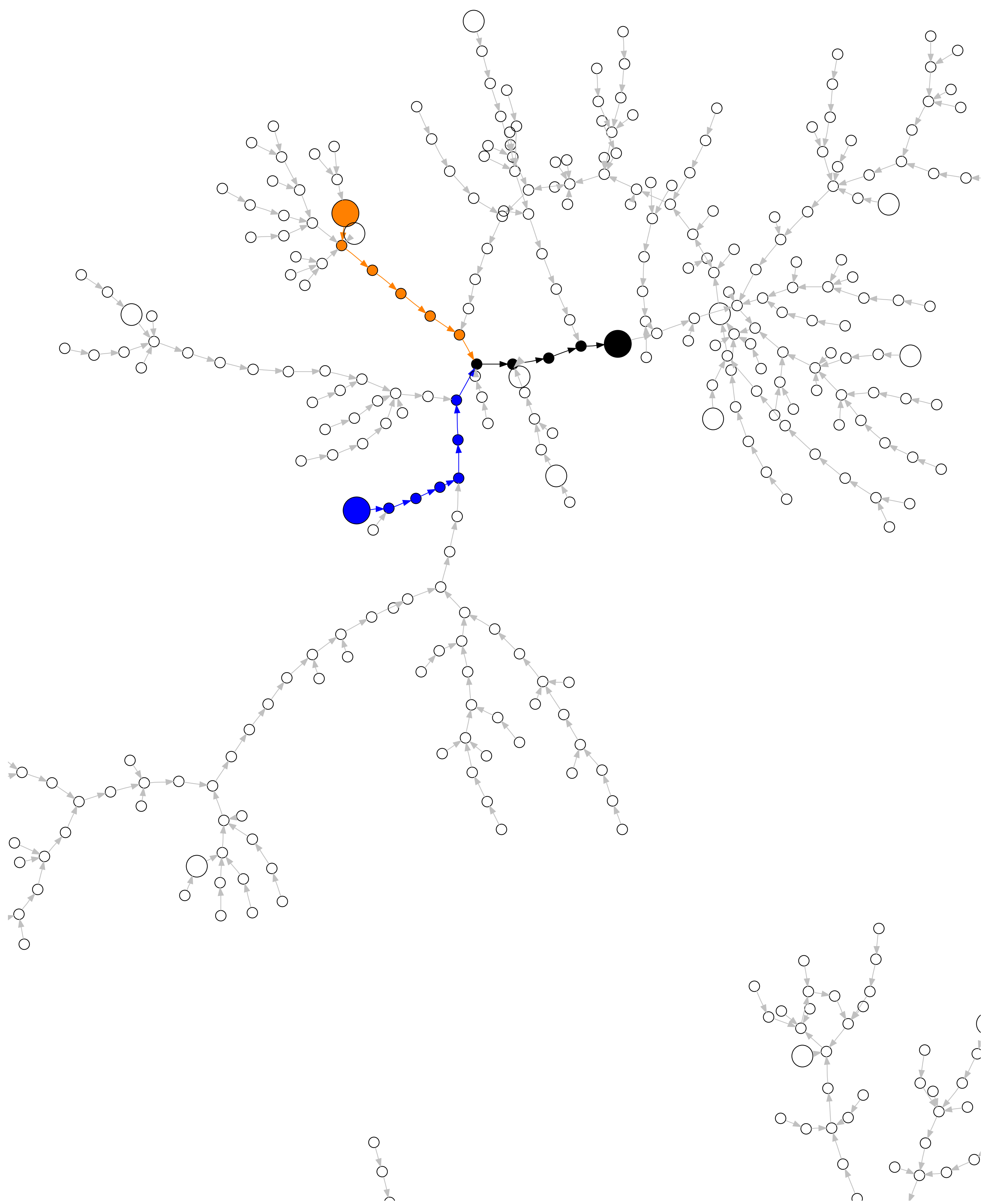
walks
s
.

match.

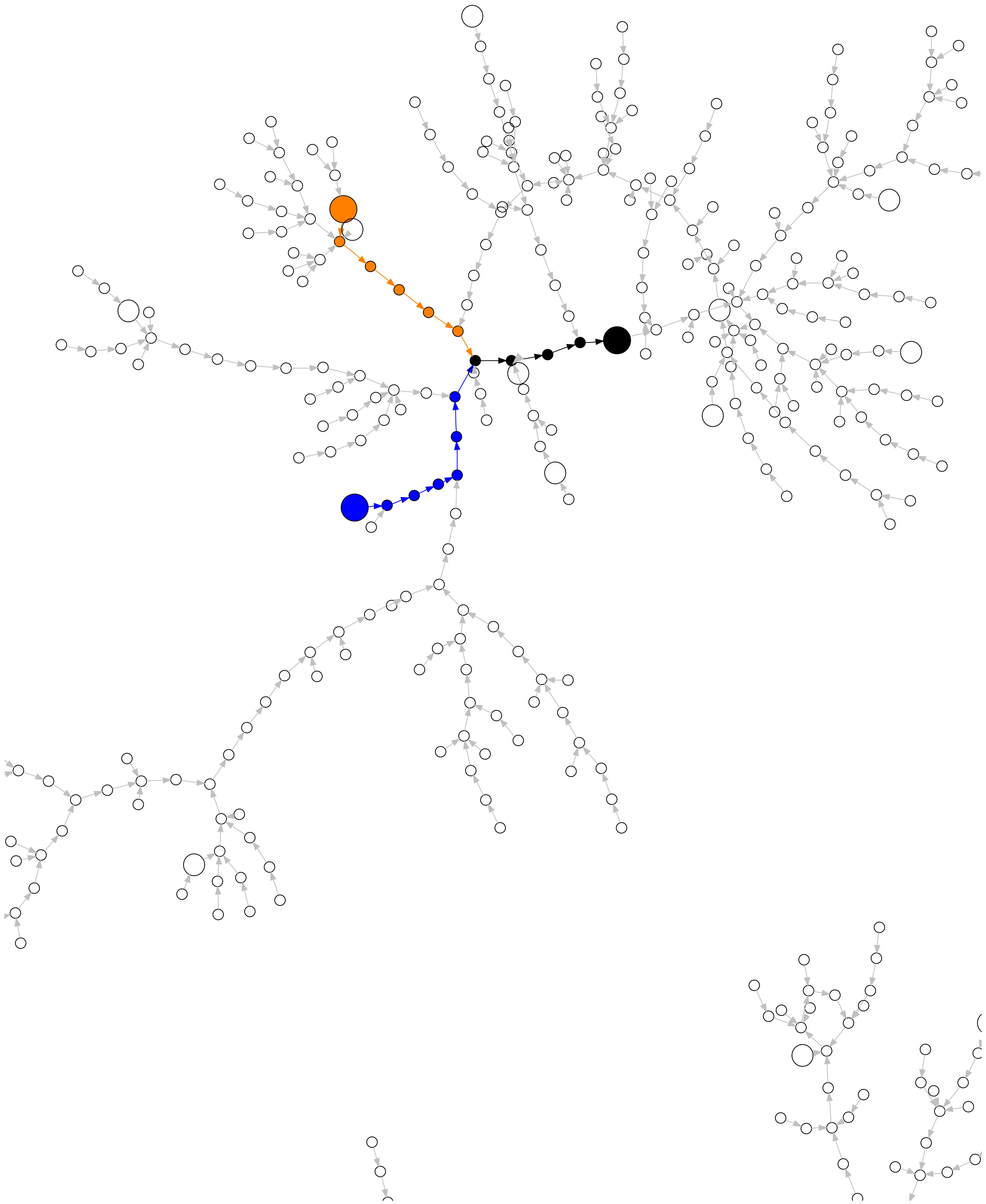
hits
port
b_i to

sorts

ision.
P.



Attacker chooses frequency
definition of distinguished po
Tradeoffs are possible:
If distinguished points are ra
small number of very long w
will be performed. This redu
the number of distinguished
points sent to the server but
increases the delay before a
collision is recognized.
If distinguished points are
frequent, many shorter walk
be performed.
In any case do not wait for c
Total # of iterations unchar



Attacker chooses frequency and definition of distinguished points.

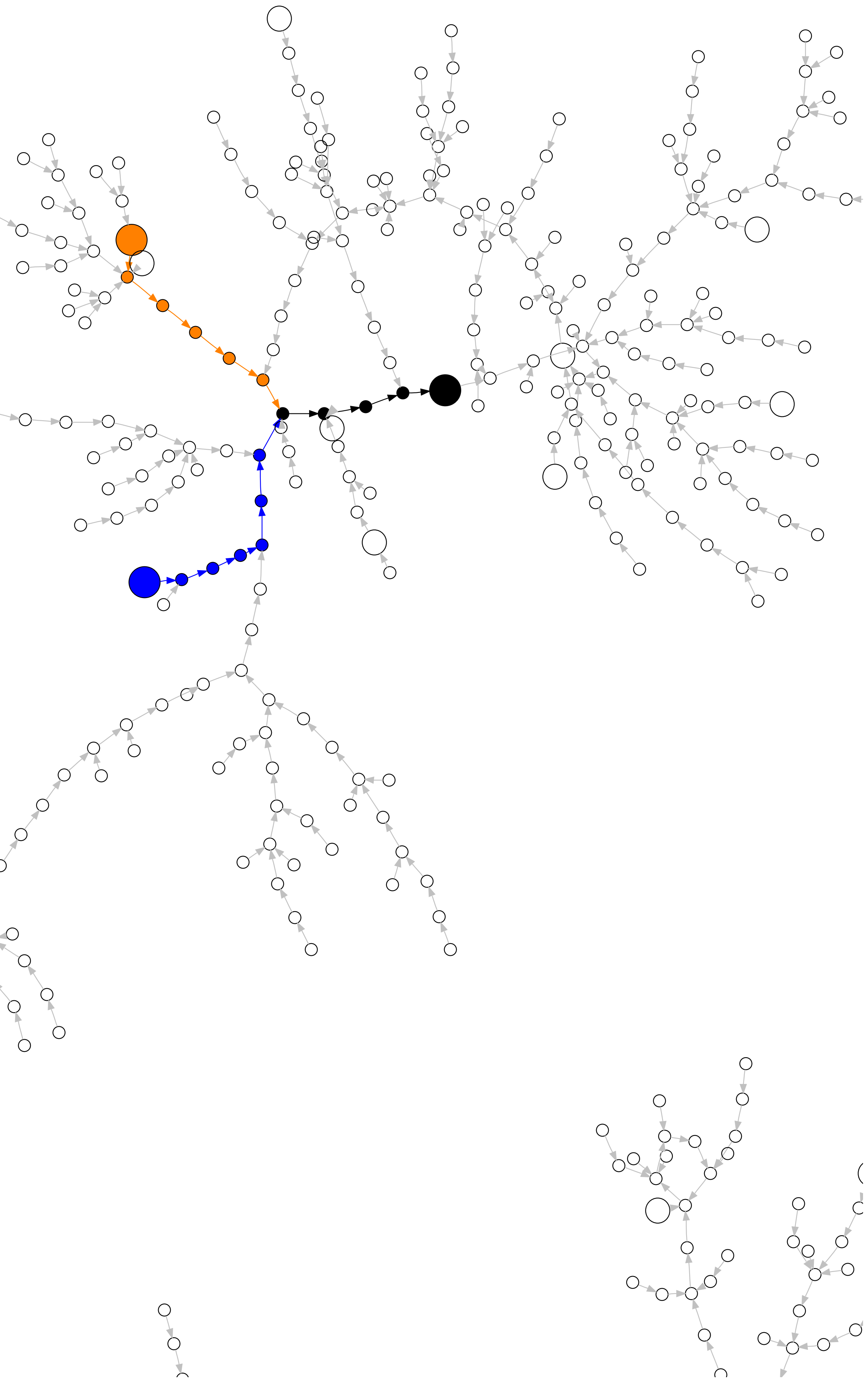
Tradeoffs are possible:

If distinguished points are rare, a small number of very long walks will be performed. This reduces the number of distinguished points sent to the server but increases the delay before a collision is recognized.

If distinguished points are frequent, many shorter walks will be performed.

In any case do not wait for cycle.

Total # of iterations unchanged.



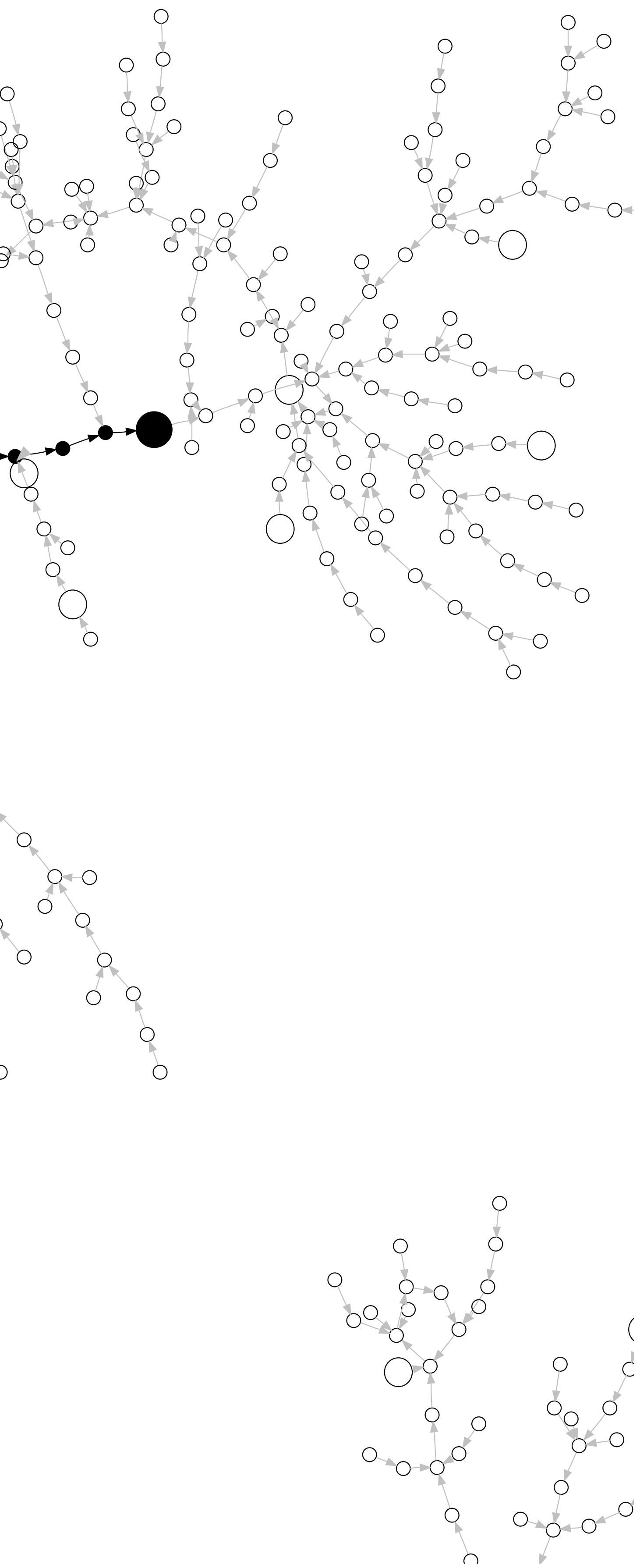
Attacker chooses frequency and definition of distinguished points. Tradeoffs are possible:

If distinguished points are rare, a small number of very long walks will be performed. This reduces the number of distinguished points sent to the server but increases the delay before a collision is recognized.

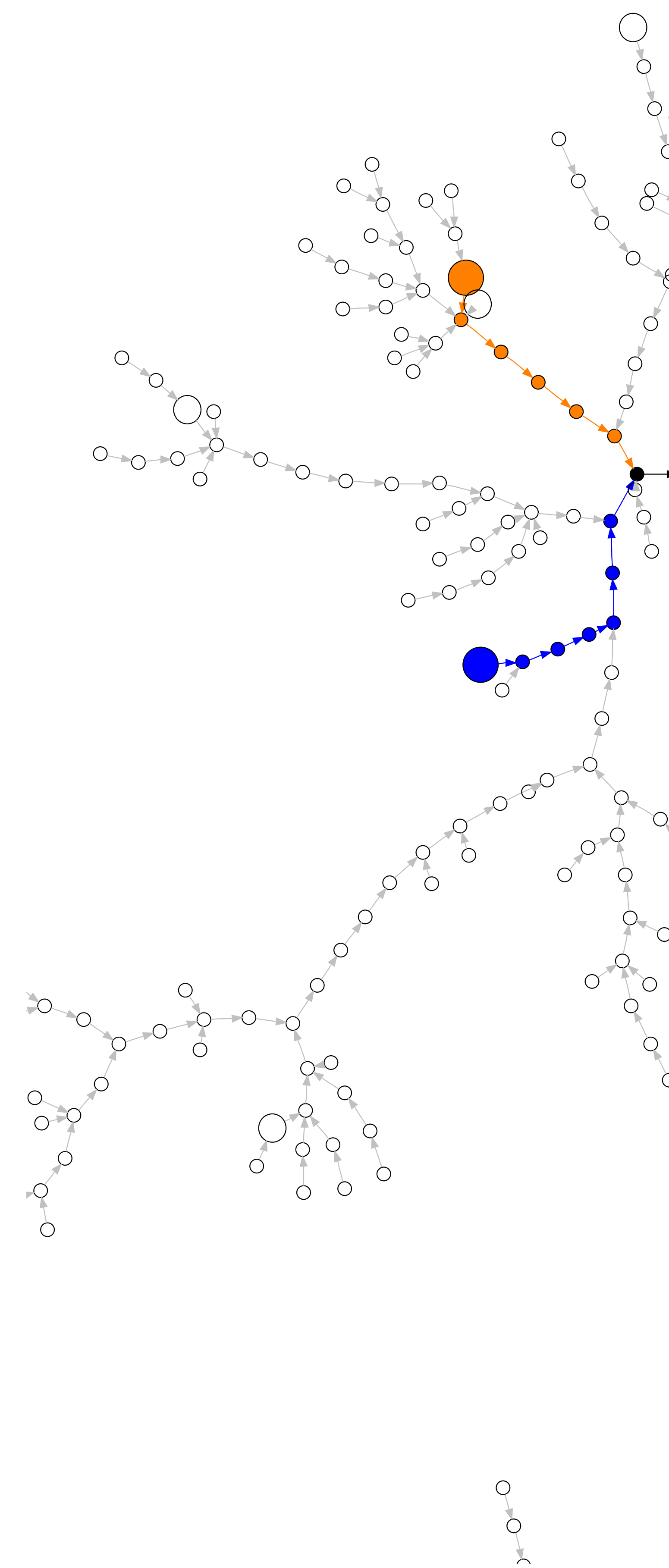
If distinguished points are frequent, many shorter walks will be performed.

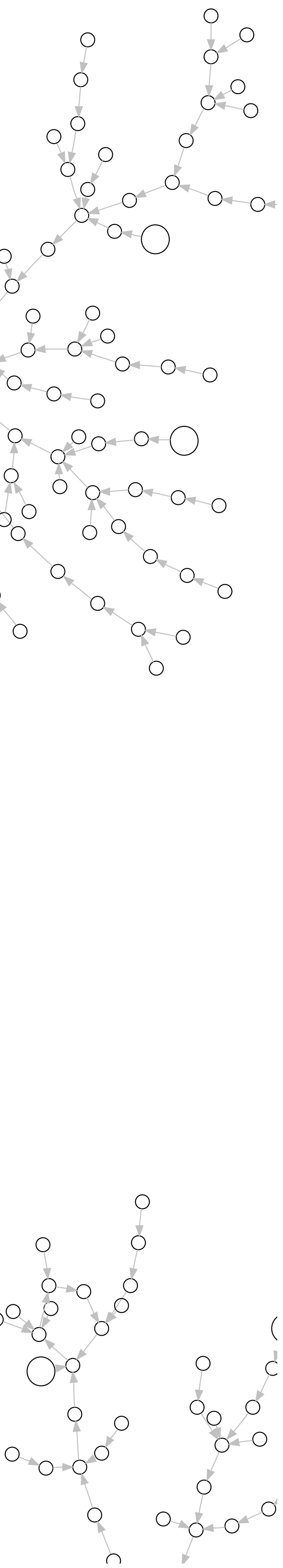
In any case do not wait for cycle.

Total # of iterations unchanged.

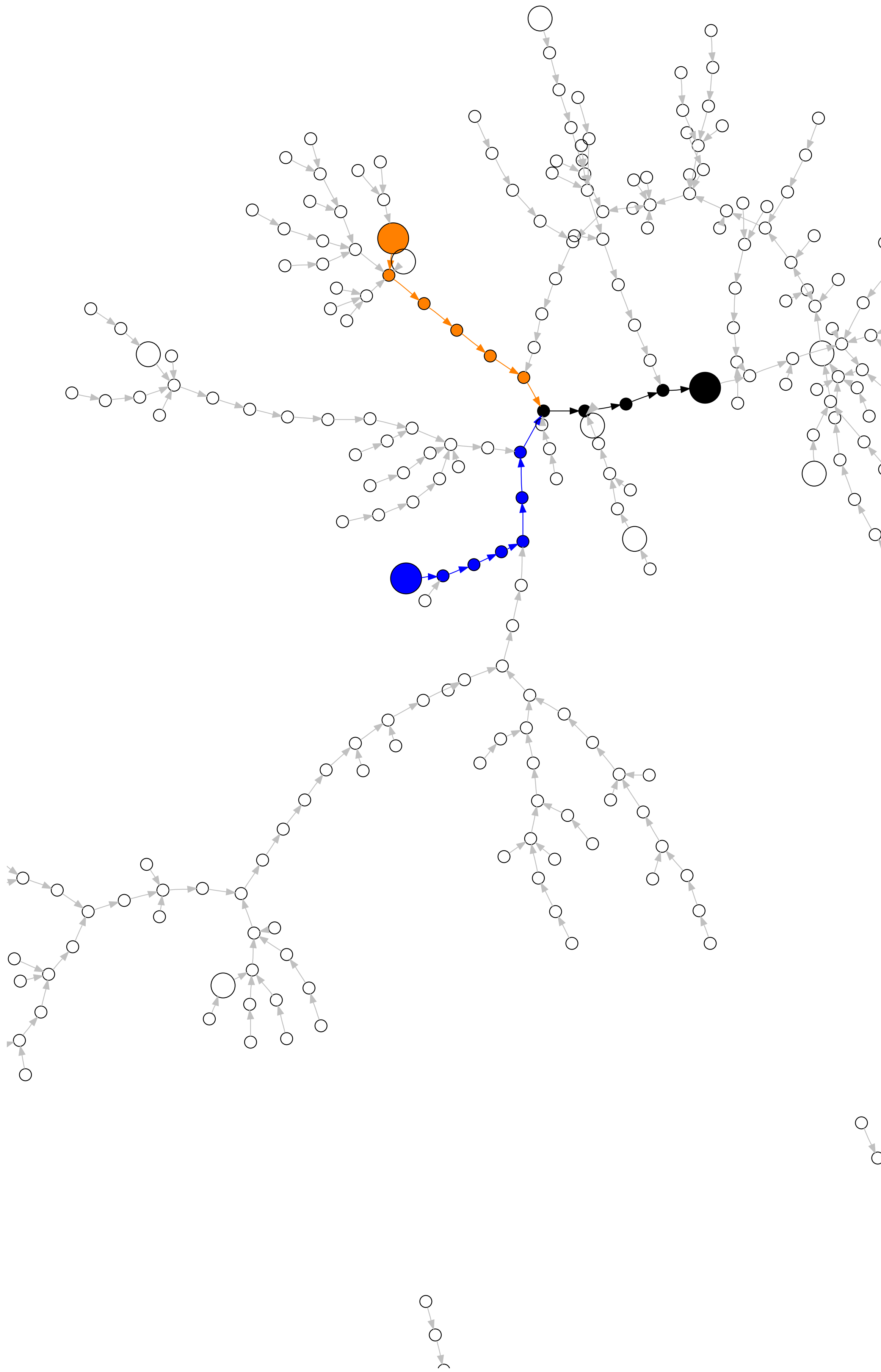


Attacker chooses frequency and definition of distinguished points.
Tradeoffs are possible:
If distinguished points are rare, a small number of very long walks will be performed. This reduces the number of distinguished points sent to the server but increases the delay before a collision is recognized.
If distinguished points are frequent, many shorter walks will be performed.
In any case do not wait for cycle.
Total # of iterations unchanged.

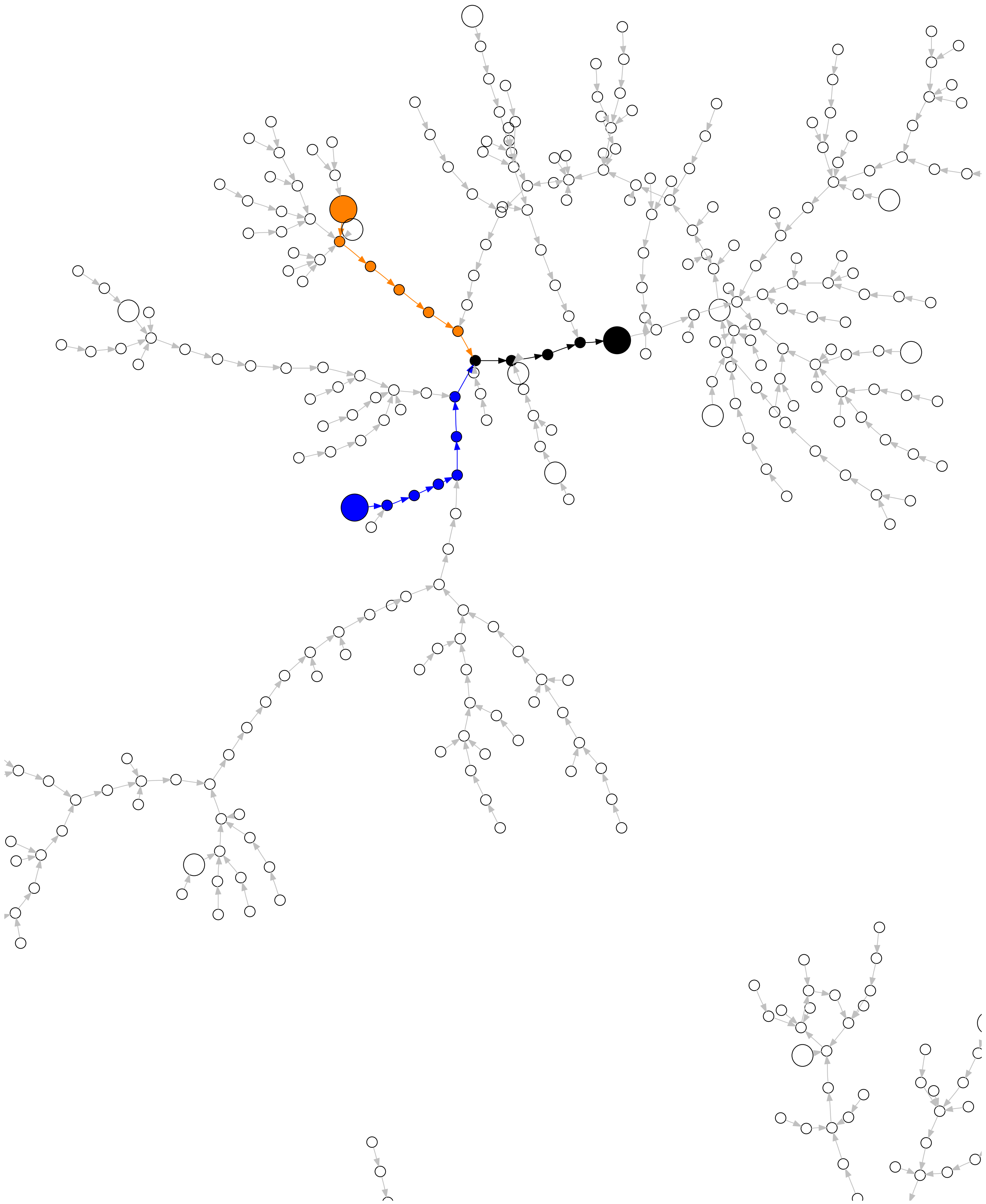




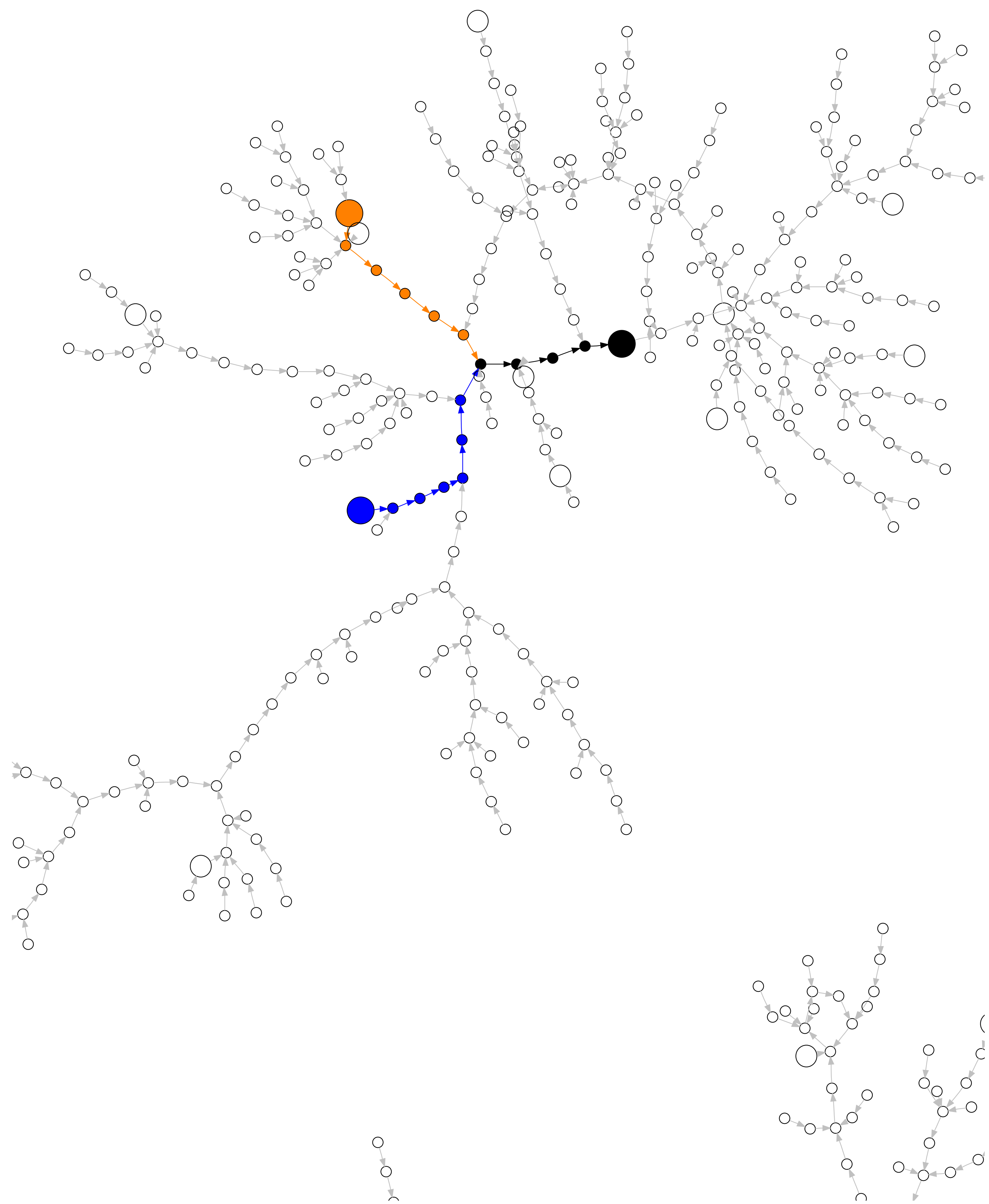
Attacker chooses frequency and definition of distinguished points.
Tradeoffs are possible:
If distinguished points are rare, a small number of very long walks will be performed. This reduces the number of distinguished points sent to the server but increases the delay before a collision is recognized.
If distinguished points are frequent, many shorter walks will be performed.
In any case do not wait for cycle.
Total # of iterations unchanged.



Attacker chooses frequency and definition of distinguished points.
Tradeoffs are possible:
If distinguished points are rare, a small number of very long walks will be performed. This reduces the number of distinguished points sent to the server but increases the delay before a collision is recognized.
If distinguished points are frequent, many shorter walks will be performed.
In any case do not wait for cycle.
Total # of iterations unchanged.



r chooses frequency and
n of distinguished points.
fs are possible:
guished points are rare, a
mber of very long walks
performed. This reduces
ber of distinguished
ent to the server but
s the delay before a
is recognized.
guished points are
, many shorter walks will
rmed.
ase do not wait for cycle.
of iterations unchanged.

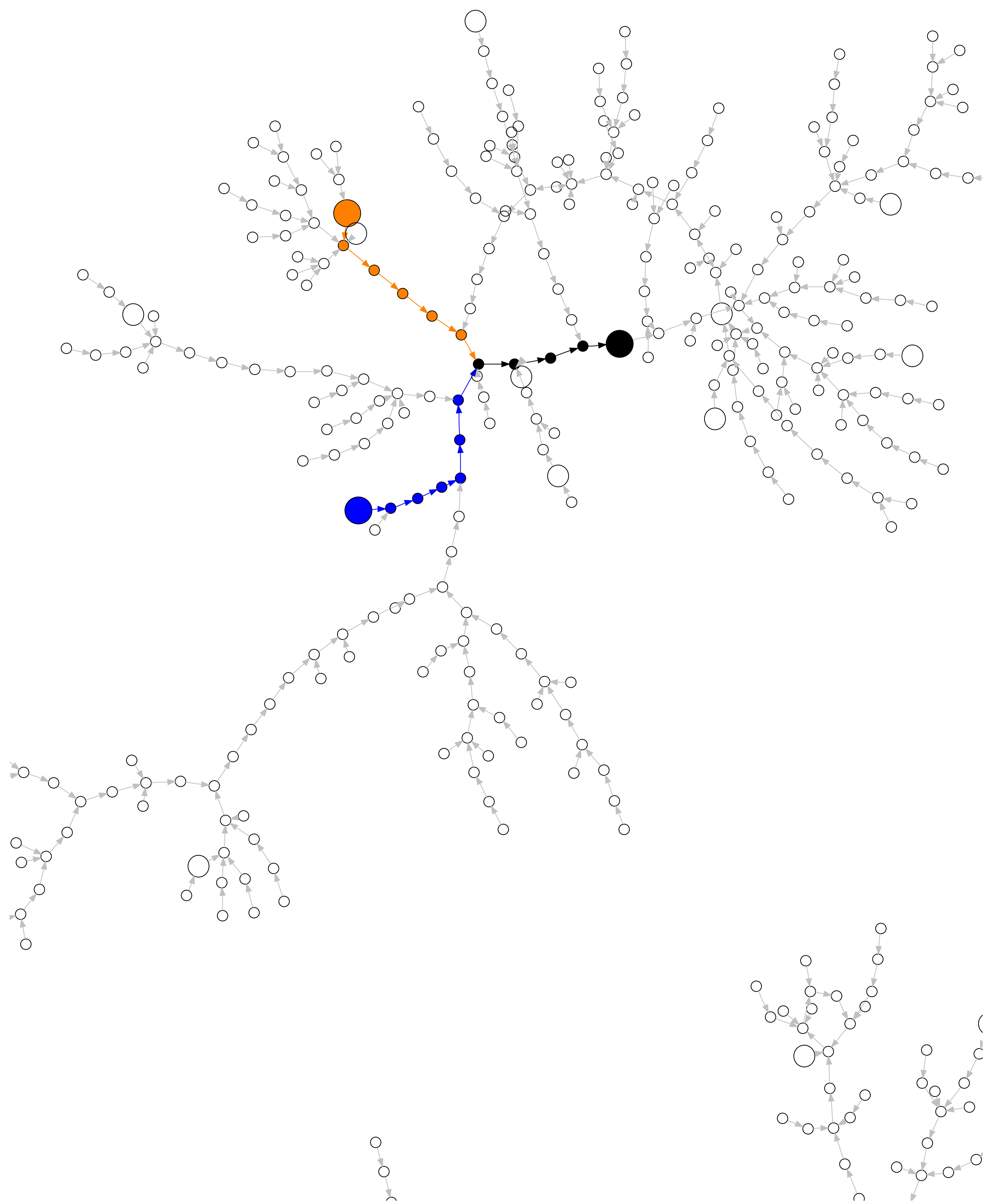


Additive

Generic
scalar m
iteration
Could re
multiplic
merge th
across se

frequency and
guished points.
ible:
oints are rare, a
ery long walks
. This reduces
tinguished
server but
y before a
zed.
oints are
orter walks will

t wait for cycle.
ons unchanged.



Additive walks

Generic rho metho
scalar multiplicatio
iteration.
Could replace by c
multiplication; cou
merge the 2-scalar
across several para

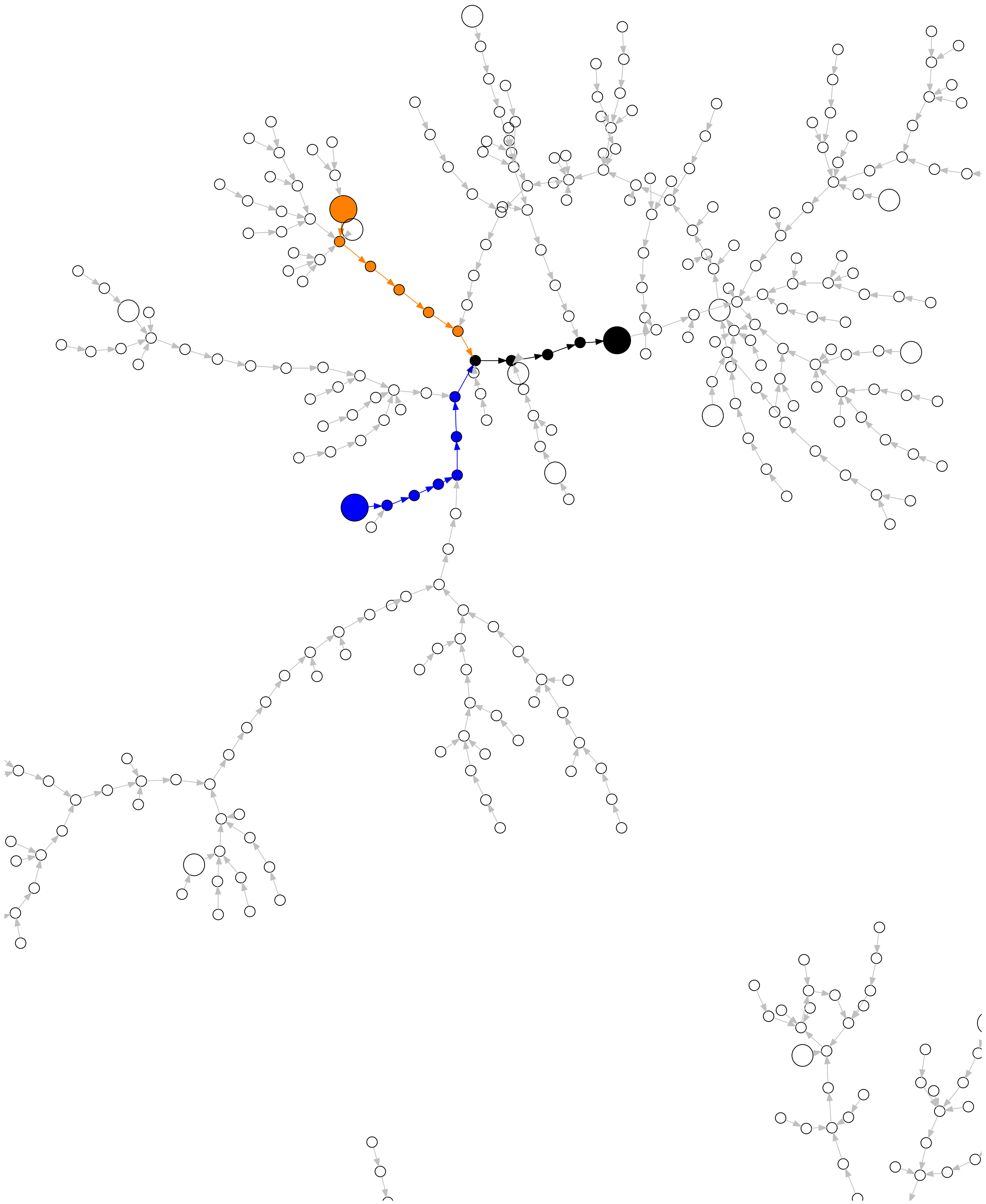
and
oints.

are, a
valks
uces

t

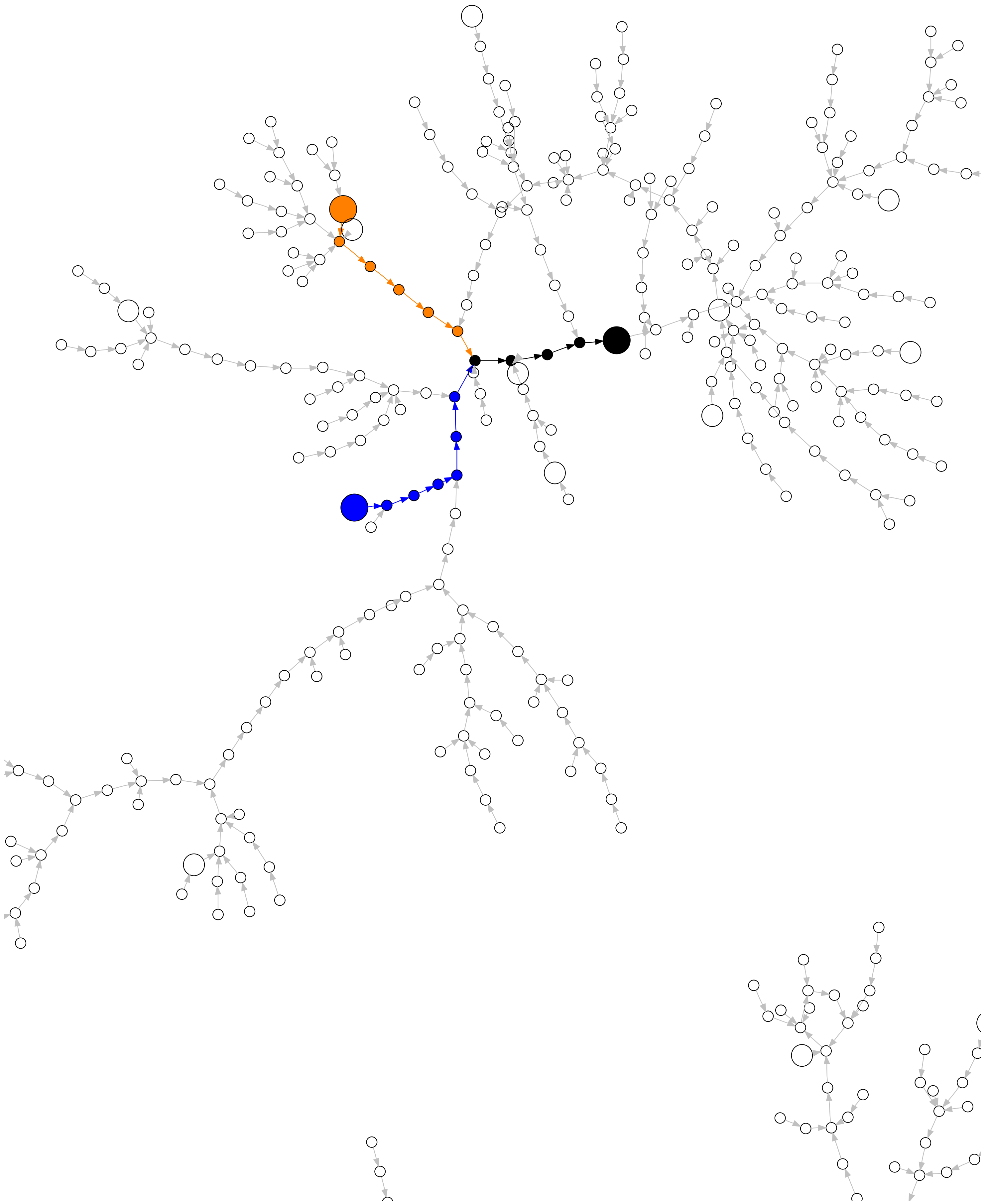
s will

cycle.
nged.



Additive walks

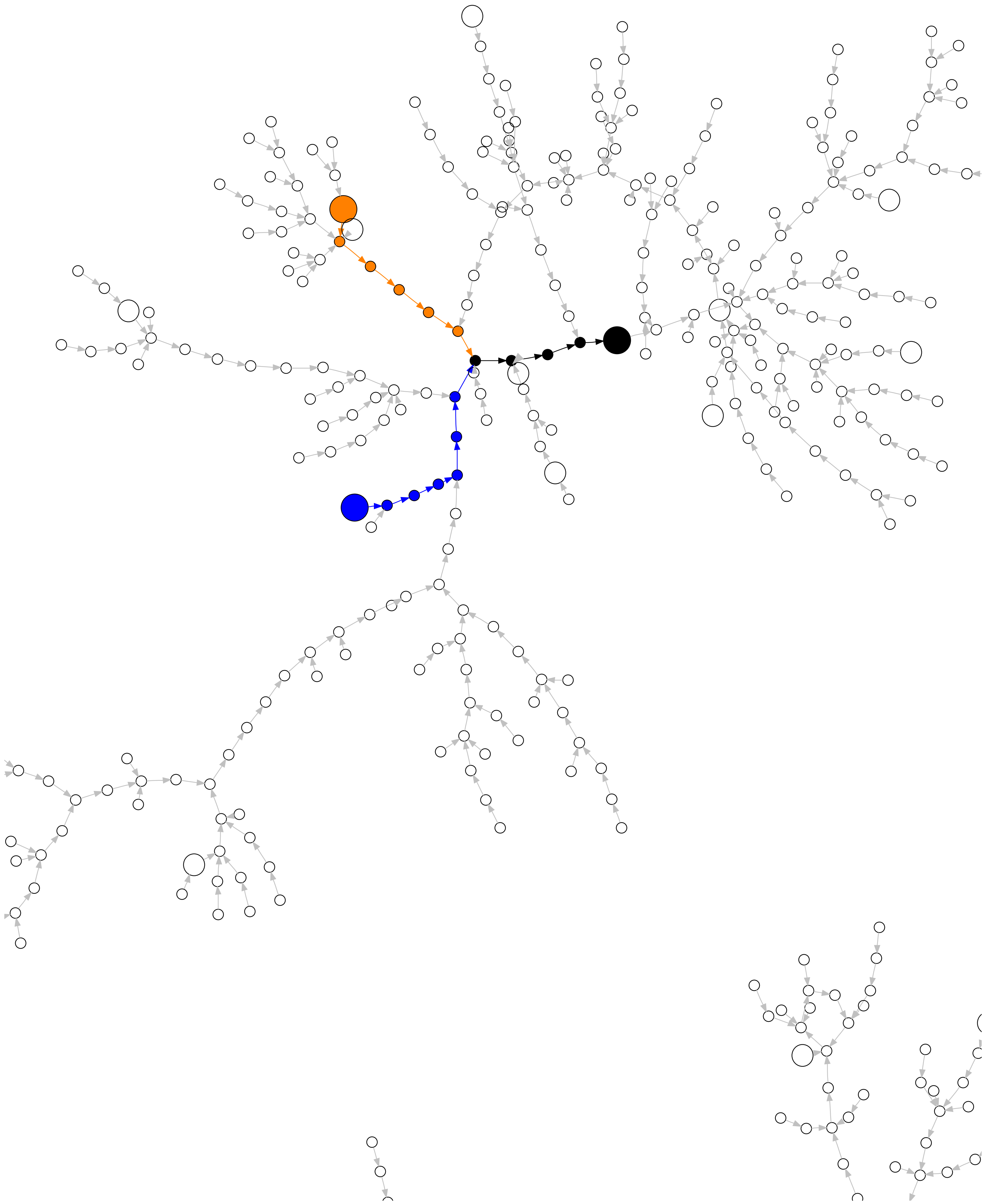
Generic rho method requires
scalar multiplications for eac
iteration.
Could replace by double-scal
multiplication; could further
merge the 2-scalar multiplica
across several parallel iterati



Additive walks

Generic rho method requires two scalar multiplications for each iteration.

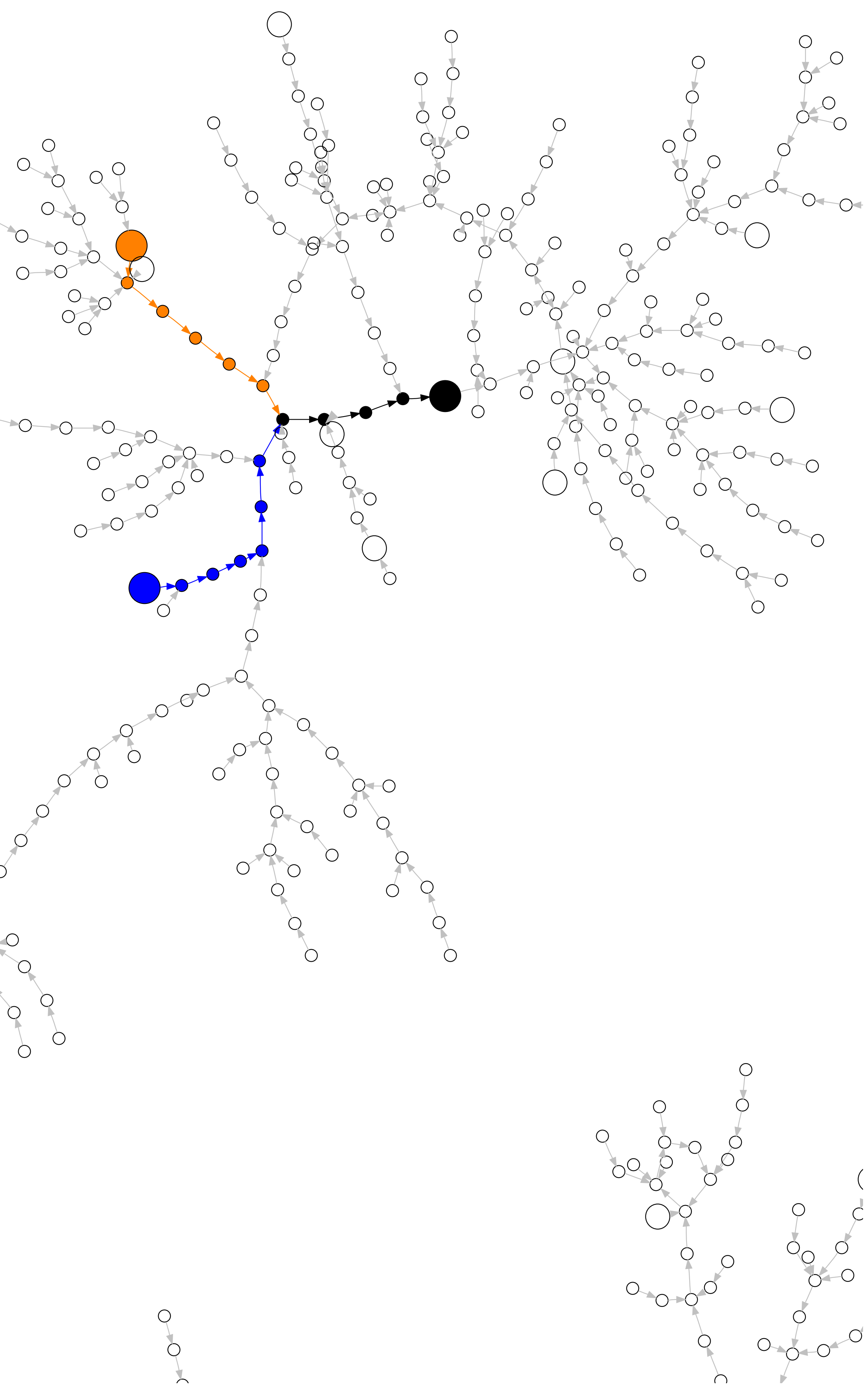
Could replace by double-scalar multiplication; could further merge the 2-scalar multiplications across several parallel iterations.



Additive walks

Generic rho method requires two scalar multiplications for each iteration.
Could replace by double-scalar multiplication; could further merge the 2-scalar multiplications across several parallel iterations.

More efficient: use *additive walk*:
Start with $W_0 = a_0 P$ and put
$f(W_i) = W_i + c_j P + d_j Q$
where $j = h(W_i)$.



Additive walks

Generic rho method requires two scalar multiplications for each iteration.
Could replace by double-scalar multiplication; could further merge the 2-scalar multiplications across several parallel iterations.

More efficient: use *additive walk*:
Start with $W_0 = a_0 P$ and put
$f(W_i) = W_i + c_j P + d_j Q$
where $j = h(W_i)$.

Pollard's

Use $x(W$

and upd

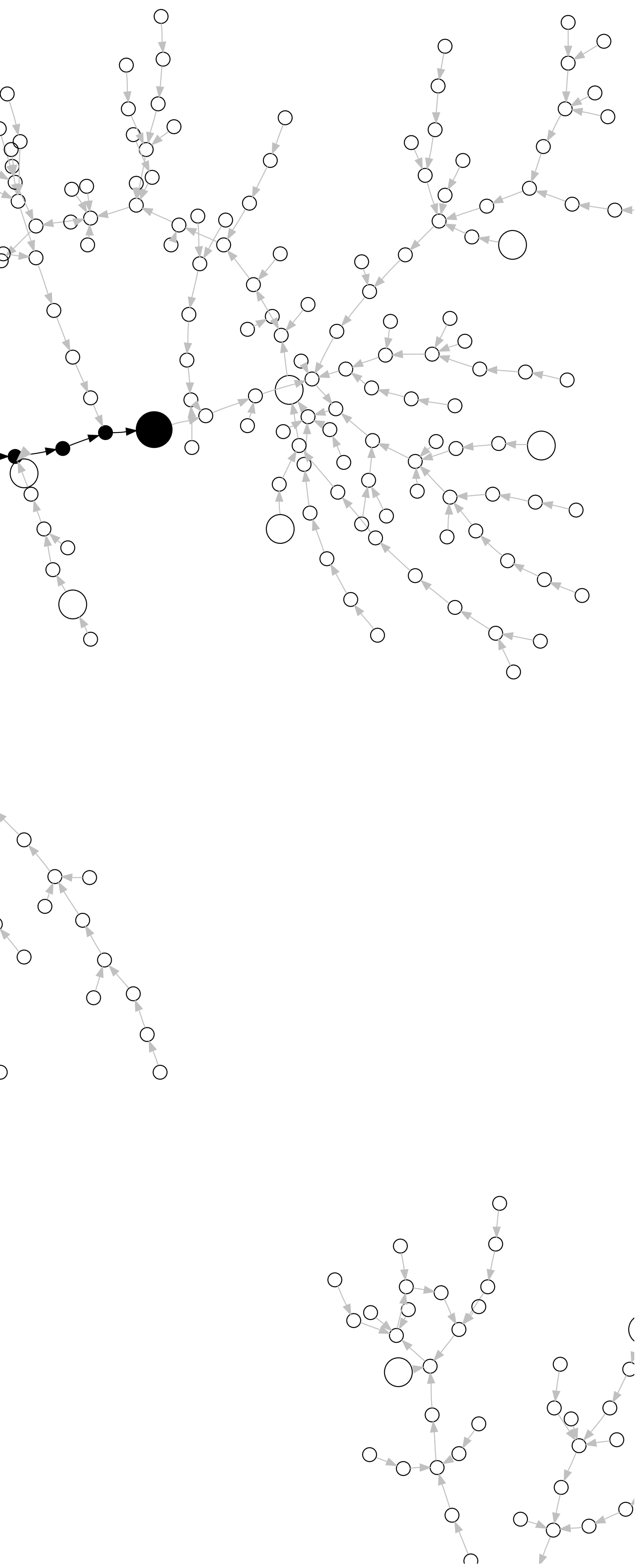
W_{i+1}

$\begin{cases} W_i + \\ 2W_i \\ W_i + \end{cases}$

Easy to

$(a_{i+1}, b_i$

$\begin{cases} (a_i + \\ (2a_i, \\ (a_i, b_i \end{cases}$



Additive walks

Generic rho method requires two scalar multiplications for each iteration.
Could replace by double-scalar multiplication; could further merge the 2-scalar multiplications across several parallel iterations.

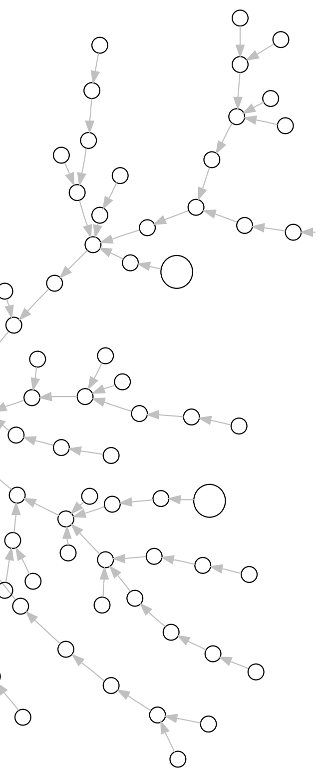
More efficient: use *additive walk*:
Start with $W_0 = a_0P$ and put
$f(W_i) = W_i + c_jP + d_jQ$
where $j = h(W_i)$.

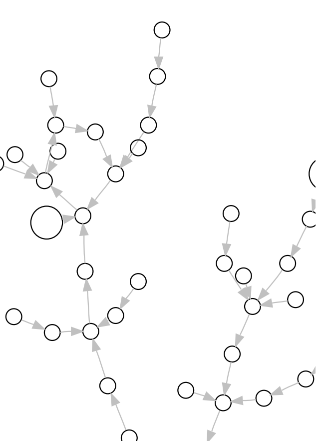
Pollard's initial pro
Use $x(W_i)$ mod 3
and update:

$W_{i+1} =$
$\begin{cases} W_i + P \text{ for } x(W \\ 2W_i \quad \text{for } x(W_i \\ W_i + Q \text{ for } x(W \end{cases}$

Easy to update a_i

$(a_{i+1}, b_{i+1}) =$
$\begin{cases} (a_i + 1, b_i) \text{ for } x \\ (2a_i, 2b_i) \quad \text{for} \\ (a_i, b_i + 1) \text{ for } x \end{cases}$





Additive walks

Generic rho method requires two scalar multiplications for each iteration.
Could replace by double-scalar multiplication; could further merge the 2-scalar multiplications across several parallel iterations.

More efficient: use *additive walk*:
Start with $W_0 = a_0 P$ and put
$f(W_i) = W_i + c_j P + d_j Q$
where $j = h(W_i)$.

Pollard's initial proposal:
Use $x(W_i)$ mod 3 as h
and update:

$$W_{i+1} = \begin{cases} W_i + P \text{ for } x(W_i) \bmod 3 \\ 2W_i \quad \text{ for } x(W_i) \bmod 3 = \\ W_i + Q \text{ for } x(W_i) \bmod 3 \end{cases}$$

Easy to update a_i and b_i.

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1, b_i) \text{ for } x(W_i) \text{ mod} \\ (2a_i, 2b_i) \quad \text{ for } x(W_i) \text{ mo} \\ (a_i, b_i + 1) \text{ for } x(W_i) \text{ mod} \end{cases}$$

Additive walks

Generic rho method requires two scalar multiplications for each iteration.
Could replace by double-scalar multiplication; could further merge the 2-scalar multiplications across several parallel iterations.

More efficient: use *additive walk*:
Start with $W_0 = a_0 P$ and put
$f(W_i) = W_i + c_j P + d_j Q$
where $j = h(W_i)$.

Pollard's initial proposal:
Use $x(W_i) \bmod 3$ as h
and update:

$$W_{i+1} = \begin{cases} W_i + P & \text{for } x(W_i) \bmod 3 = 0 \\ 2W_i & \text{for } x(W_i) \bmod 3 = 1 \\ W_i + Q & \text{for } x(W_i) \bmod 3 = 2 \end{cases}$$

Easy to update a_i and b_i.

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1, b_i) & \text{for } x(W_i) \bmod 3 = 0 \\ (2a_i, 2b_i) & \text{for } x(W_i) \bmod 3 = 1 \\ (a_i, b_i + 1) & \text{for } x(W_i) \bmod 3 = 2 \end{cases}$$

walks

rho method requires two
ultiplications for each
.
eplace by double-scalar
cation; could further
ne 2-scalar multiplications
everal parallel iterations.

ficient: use *additive walk*:
th $W_0 = a_0 P$ and put
$= W_i + c_j P + d_j Q$
$= h(W_i)$.

Pollard's initial proposal:

Use $x(W_i)$ mod 3 as h

and update:

$$W_{i+1} = \begin{cases} W_i + P & \text{for } x(W_i) \bmod 3 = 0 \\ 2W_i & \text{for } x(W_i) \bmod 3 = 1 \\ W_i + Q & \text{for } x(W_i) \bmod 3 = 2 \end{cases}$$

Easy to update a_i and b_i.

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1, b_i) & \text{for } x(W_i) \bmod 3 = 0 \\ (2a_i, 2b_i) & \text{for } x(W_i) \bmod 3 = 1 \\ (a_i, b_i + 1) & \text{for } x(W_i) \bmod 3 = 2 \end{cases}$$

Additive
addition
h maps
$\{0, 1, \ldots$
$R_j = c_j$
precomp
$j \in \{0, 1$

Easy coe
$W_i = a_i$
where a_i
recursive
$a_{i+1} =$
$b_{i+1} =$

od requires two
ons for each

double-scalar
uld further
r multiplications
allel iterations.

e *additive walk*:
$_0P$ and put
$P + d_j Q$

Pollard's initial proposal:

Use $x(W_i) \bmod 3$ as h

and update:

$$W_{i+1} = \begin{cases} W_i + P & \text{for } x(W_i) \bmod 3 = 0 \\ 2W_i & \text{for } x(W_i) \bmod 3 = 1 \\ W_i + Q & \text{for } x(W_i) \bmod 3 = 2 \end{cases}$$

Easy to update a_i and b_i.

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1, b_i) & \text{for } x(W_i) \bmod 3 = 0 \\ (2a_i, 2b_i) & \text{for } x(W_i) \bmod 3 = 1 \\ (a_i, b_i + 1) & \text{for } x(W_i) \bmod 3 = 2 \end{cases}$$

Additive walk requ
addition per iterat
h maps from $\langle P \rangle$
$\{0, 1, \ldots, r-1\}$,
$R_j = c_j P + d_j Q$ a
precomputed for e
$j \in \{0, 1, \ldots, r -$

Easy coefficient up
$W_i = a_i P + b_i Q$,
where a_i and b_i ar
recursively as follo
$a_{i+1} = a_i + c_{h(W_i}$
$b_{i+1} = b_i + d_{h(W_i)}$

s two
ch

lar

ations
ons.

walk:
ut

Pollard's initial proposal:
Use $x(W_i) \bmod 3$ as h
and update:

$$W_{i+1} = \begin{cases} W_i + P \text{ for } x(W_i) \bmod 3 = 0 \\ 2W_i \quad \text{ for } x(W_i) \bmod 3 = 1 \\ W_i + Q \text{ for } x(W_i) \bmod 3 = 2 \end{cases}$$

Easy to update a_i and b_i.

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1, b_i) \text{ for } x(W_i) \bmod 3 = 0 \\ (2a_i, 2b_i) \quad \text{ for } x(W_i) \bmod 3 = 1 \\ (a_i, b_i + 1) \text{ for } x(W_i) \bmod 3 = 2 \end{cases}$$

Additive walk requires only o
addition per iteration.
h maps from $\langle P \rangle$ to
$\{0, 1, \ldots, r-1\}$, and
$R_j = c_j P + d_j Q$ are
precomputed for each
$j \in \{0, 1, \ldots, r-1\}$.

Easy coefficient update:
$W_i = a_i P + b_i Q$,
where a_i and b_i are defined
recursively as follows:
$a_{i+1} = a_i + c_{h(W_i)}$ and
$b_{i+1} = b_i + d_{h(W_i)}$.

Pollard's initial proposal:

Use $x(W_i) \bmod 3$ as h and update:

$$W_{i+1} = \begin{cases} W_i + P & \text{for } x(W_i) \bmod 3 = 0 \\ 2W_i & \text{for } x(W_i) \bmod 3 = 1 \\ W_i + Q & \text{for } x(W_i) \bmod 3 = 2 \end{cases}$$

Easy to update a_i and b_i.

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1, b_i) & \text{for } x(W_i) \bmod 3 = 0 \\ (2a_i, 2b_i) & \text{for } x(W_i) \bmod 3 = 1 \\ (a_i, b_i + 1) & \text{for } x(W_i) \bmod 3 = 2 \end{cases}$$

Additive walk requires only one addition per iteration.

h maps from $\langle P \rangle$ to $\{0, 1, \ldots, r-1\}$, and $R_j = c_j P + d_j Q$ are precomputed for each $j \in \{0, 1, \ldots, r-1\}$.

Easy coefficient update:

$W_i = a_i P + b_i Q$,

where a_i and b_i are defined recursively as follows:

$a_{i+1} = a_i + c_{h(W_i)}$ and

$b_{i+1} = b_i + d_{h(W_i)}$.

s initial proposal:

$/_i)$ mod 3 as h

ate:

$=$

P for $x(W_i)$ mod $3 = 0$

for $x(W_i)$ mod $3 = 1$

Q for $x(W_i)$ mod $3 = 2$

update a_i and b_i.

$_{+1}) \quad =$

$1, b_i)$ for $x(W_i)$ mod $3 = 0$

$2b_i) \quad$ for $x(W_i)$ mod $3 = 1$

$+ 1)$ for $x(W_i)$ mod $3 = 2$

Additive walk requires only one addition per iteration.

h maps from $\langle P \rangle$ to $\{0, 1, \ldots, r - 1\}$, and $R_j = c_j P + d_j Q$ are precomputed for each $j \in \{0, 1, \ldots, r - 1\}$.

Easy coefficient update:

$W_i = a_i P + b_i Q$,

where a_i and b_i are defined recursively as follows:

$a_{i+1} = a_i + c_{h(W_i)}$ and

$b_{i+1} = b_i + d_{h(W_i)}$.

Additive

disadvar

The wal

nonrand

more ite

rho metl

This effe

but but

table R_0

into fast

on the p

GPUs.

More tro

later.

oposal:

as h

$V_i)$ mod $3=0$

$)$ mod $3=1$

$V_i)$ mod $3=2$

and b_i.

$x(W_i)$ mod $3=0$

$x(W_i)$ mod $3=1$

$x(W_i)$ mod $3=2$

Additive walk requires only one addition per iteration.

h maps from $\langle P\rangle$ to $\{0,1,\ldots,r-1\}$, and $R_j = c_j P + d_j Q$ are precomputed for each $j \in \{0,1,\ldots,r-1\}$.

Easy coefficient update:
$W_i = a_i P + b_i Q$,
where a_i and b_i are defined recursively as follows:
$a_{i+1} = a_i + c_{h(W_i)}$ and
$b_{i+1} = b_i + d_{h(W_i)}$.

Additive walks hav

disadvantages:

The walks are not

nonrandom; this m

more iterations th

rho method to fin

This effect disapp

but but then the p

table $R_0,\ldots,R_{r-}$

into fast memory.

on the platform, e

GPUs.

More trouble with

later.

$= 0$
$= 1$
$= 2$

d $3 = 0$
d $3 = 1$
d $3 = 2$

Additive walk requires only one addition per iteration.
h maps from $\langle P \rangle$ to $\{0, 1, \ldots, r-1\}$, and $R_j = c_j P + d_j Q$ are precomputed for each $j \in \{0, 1, \ldots, r-1\}$.

Easy coefficient update:
$W_i = a_i P + b_i Q$,
where a_i and b_i are defined recursively as follows:
$a_{i+1} = a_i + c_{h(W_i)}$ and
$b_{i+1} = b_i + d_{h(W_i)}$.

Additive walks have disadvantages:
The walks are noticeably nonrandom; this means they more iterations than the gen rho method to find a collisio

This effect disappears as r g but but then the precomput table $R_0, \ldots, R_{r-1}$ does not into fast memory. This depe on the platform, e.g. trouble GPUs.
More trouble with adding wa later.

Additive walk requires only one addition per iteration.
h maps from $\langle P \rangle$ to $\{0, 1, \ldots, r-1\}$, and $R_j = c_j P + d_j Q$ are precomputed for each $j \in \{0, 1, \ldots, r-1\}$.

Easy coefficient update:
$W_i = a_i P + b_i Q$,
where a_i and b_i are defined recursively as follows:
$a_{i+1} = a_i + c_{h(W_i)}$ and
$b_{i+1} = b_i + d_{h(W_i)}$.

Additive walks have disadvantages:
The walks are noticeably nonrandom; this means they need more iterations than the generic rho method to find a collision.

This effect disappears as r grows, but but then the precomputed table $R_0, \ldots, R_{r-1}$ does not fit into fast memory. This depends on the platform, e.g. trouble for GPUs.
More trouble with adding walks later.

walk requires only one per iteration.
from $\langle P \rangle$ to
$\ldots, r-1\}$, and
$P + d_j Q$ are
puted for each
$1, \ldots, r-1\}$.

efficient update:
$P + b_i Q$,
$_i$ and b_i are defined
ely as follows:
$a_i + c_{h(W_i)}$ and
$b_i + d_{h(W_i)}$.

Additive walks have disadvantages:
The walks are noticeably nonrandom; this means they need more iterations than the generic rho method to find a collision.

This effect disappears as r grows, but but then the precomputed table $R_0, \ldots, R_{r-1}$ does not fit into fast memory. This depends on the platform, e.g. trouble for GPUs.
More trouble with adding walks later.

Random

Let $h(W$

Fix a po
W' be tw
random

Let $W \neq$

This eve

uires only one
ion.
to
and
are
ach
1}.

pdate:

re defined
ws:
) and
).

Additive walks have disadvantages:
The walks are noticeably nonrandom; this means they need more iterations than the generic rho method to find a collision.

This effect disappears as r grows, but but then the precomputed table $R_0, \ldots, R_{r-1}$ does not fit into fast memory. This depends on the platform, e.g. trouble for GPUs.
More trouble with adding walks later.

Randomness of ad

Let $h(W) = i$ with

Fix a point T, and
W' be two indepe
random points.

Let $W \neq W'$ both

This event occurs

one

Additive walks have disadvantages:
The walks are noticeably nonrandom; this means they need more iterations than the generic rho method to find a collision.

This effect disappears as r grows, but but then the precomputed table $R_0, \ldots, R_{r-1}$ does not fit into fast memory. This depends on the platform, e.g. trouble for GPUs.
More trouble with adding walks later.

Randomness of adding walks

Let $h(W) = i$ with probabili

Fix a point T, and let W an W' be two independent unif random points.

Let $W \neq W'$ both map to T

This event occurs if

Additive walks have disadvantages:
The walks are noticeably nonrandom; this means they need more iterations than the generic rho method to find a collision.

This effect disappears as r grows, but but then the precomputed table $R_0, \ldots, R_{r-1}$ does not fit into fast memory. This depends on the platform, e.g. trouble for GPUs.
More trouble with adding walks later.

Randomness of adding walks

Let $h(W) = i$ with probability p_i.

Fix a point T, and let W and W' be two independent uniform random points.

Let $W \neq W'$ both map to T.

This event occurs if

Additive walks have disadvantages:
The walks are noticeably nonrandom; this means they need more iterations than the generic rho method to find a collision.

This effect disappears as r grows, but but then the precomputed table $R_0, \ldots, R_{r-1}$ does not fit into fast memory. This depends on the platform, e.g. trouble for GPUs.
More trouble with adding walks later.

Randomness of adding walks

Let $h(W) = i$ with probability p_i.

Fix a point T, and let W and W' be two independent uniform random points.

Let $W \neq W'$ both map to T.

This event occurs if simultaneously for $i \neq j$:
$T = W + R_i = W' + R_j$;
$h(W) = i$; $h(W') = j$.

These conditions have probability $1/\ell^2$, p_i, and p_j respectively.

walks have
ntages:
ks are noticeably
om; this means they need
rations than the generic
hod to find a collision.

ect disappears as r grows,
then the precomputed
$, \ldots, R_{r-1}$ does not fit
memory. This depends
latform, e.g. trouble for

uble with adding walks

Randomness of adding walks

Let $h(W) = i$ with probability p_i.

Fix a point T, and let W and W' be two independent uniform random points.

Let $W \neq W'$ both map to T.

This event occurs if simultaneously for $i \neq j$:
$T = W + R_i = W' + R_j$;
$h(W) = i$; $h(W') = j$.

These conditions have probability $1/\ell^2$, p_i, and p_j respectively.

Summin
gives the
$\Big(\sum_{i \neq j}$
$\Big(\sum_{i,j} p$
$(1 - \sum$

This me
of an im
and W'
we adde
In the si
are $1/r$,
optimal
factor of
$1/\sqrt{1 -}$

ve

iceably
neans they need
an the generic
d a collision.

ears as r grows,
recomputed
$_1$ does not fit
This depends
.g. trouble for

adding walks

Randomness of adding walks

Let $h(W) = i$ with probability p_i.

Fix a point T, and let W and W' be two independent uniform random points.

Let $W \neq W'$ both map to T.

This event occurs if simultaneously for $i \neq j$:
$T = W + R_i = W' + R_j$;
$h(W) = i$; $h(W') = j$.

These conditions have probability $1/\ell^2$, p_i, and p_j respectively.

Summing over all
gives the overall p
$\left(\sum_{i \neq j} p_i p_j\right)/\ell^2$
$\left(\sum_{i,j} p_i p_j - \sum_i\right.$
$\left(1 - \sum_i p_i^2\right)/\ell^2$.

This means that t
of an immediate c
and W' is $(1 - \sum$
we added over the
In the simple case
are $1/r$, the differe
optimal $\sqrt{\pi\ell/2}$ ite
factor of
$1/\sqrt{1 - 1/r} \approx 1$ -

need
neric
on.

grows,
ed
t fit
ends
e for

alks

Randomness of adding walks

Let $h(W) = i$ with probability p_i.

Fix a point T, and let W and W' be two independent uniform random points.

Let $W \neq W'$ both map to T.

This event occurs if simultaneously for $i \neq j$:
$T = W + R_i = W' + R_j$;
$h(W) = i$; $h(W') = j$.

These conditions have probability $1/\ell^2$, p_i, and p_j respectively.

Summing over all (i, j) gives the overall probability
$\left(\sum_{i \neq j} p_i p_j\right) / \ell^2 \quad =$
$\left(\sum_{i,j} p_i p_j - \sum_i p_i^2\right) / \ell^2 \;=$
$\left(1 - \sum_i p_i^2\right) / \ell^2$.

This means that the probab
of an immediate collision fro
and W' is $\left(1 - \sum_i p_i^2\right) / \ell$, w
we added over the ℓ choices
In the simple case that all th
are $1/r$, the difference from
optimal $\sqrt{\pi\ell/2}$ iterations is
factor of
$1/\sqrt{1 - 1/r} \approx 1 + 1/(2r)$.

Randomness of adding walks

Let $h(W) = i$ with probability p_i.

Fix a point T, and let W and W' be two independent uniform random points.

Let $W \neq W'$ both map to T.

This event occurs if simultaneously for $i \neq j$:
$T = W + R_i = W' + R_j$;
$h(W) = i$; $h(W') = j$.

These conditions have probability $1/\ell^2$, p_i, and p_j respectively.

Summing over all (i, j) gives the overall probability
$\left(\sum_{i \neq j} p_i p_j\right)/\ell^2 = \left(\sum_{i,j} p_i p_j - \sum_i p_i^2\right)/\ell^2 = \left(1 - \sum_i p_i^2\right)/\ell^2$.

This means that the probability of an immediate collision from W and W' is $\left(1 - \sum_i p_i^2\right)/\ell$, where we added over the ℓ choices of T.

In the simple case that all the p_i are $1/r$, the difference from the optimal $\sqrt{\pi\ell/2}$ iterations is a factor of
$1/\sqrt{1 - 1/r} \approx 1 + 1/(2r)$.

ness of adding walks

$) = i$ with probability p_i.

int T, and let W and

wo independent uniform

points.

$\neq W'$ both map to T.

ent occurs if

neously for $i \neq j$:

$+ R_i = W' + R_j$;

i; $h(W') = j$.

onditions have probability

, and p_j respectively.

Summing over all (i, j) gives the overall probability $\left(\sum_{i \neq j} p_i p_j\right) / \ell^2 = \left(\sum_{i,j} p_i p_j - \sum_i p_i^2\right) / \ell^2 = \left(1 - \sum_i p_i^2\right) / \ell^2$.

This means that the probability of an immediate collision from W and W' is $\left(1 - \sum_i p_i^2\right) / \ell$, where we added over the ℓ choices of T.

In the simple case that all the p_i are $1/r$, the difference from the optimal $\sqrt{\pi\ell/2}$ iterations is a factor of $1/\sqrt{1 - 1/r} \approx 1 + 1/(2r)$.

Various

standard

in differe

1981 Br

2001 Te

2009 EC

eprint 20

ding walks

n probability p_i.

l let W and

ndent uniform

map to T.

if

$i \neq j$:

$' + R_j$;

$= j$.

have probability

espectively.

Summing over all (i, j) gives the overall probability $\left(\sum_{i \neq j} p_i p_j\right) / \ell^2 = \left(\sum_{i,j} p_i p_j - \sum_i p_i^2\right) / \ell^2 = \left(1 - \sum_i p_i^2\right) / \ell^2$.

This means that the probability of an immediate collision from W and W' is $\left(1 - \sum_i p_i^2\right) / \ell$, where we added over the ℓ choices of T.

In the simple case that all the p_i are $1/r$, the difference from the optimal $\sqrt{\pi \ell / 2}$ iterations is a factor of $1/\sqrt{1 - 1/r} \approx 1 + 1/(2r)$.

Various heuristics

standard $\sqrt{1 - 1/}$

in different ways:

1981 Brent–Pollar

2001 Teske;

2009 ECC2K-130

eprint 2009/541.

s

ty p_i.

d

orm

T.

ability

y.

Summing over all (i, j) gives the overall probability $\left(\sum_{i \neq j} p_i p_j\right)/\ell^2 \;=\; \left(\sum_{i,j} p_i p_j - \sum_i p_i^2\right)/\ell^2 \;=\; \left(1 - \sum_i p_i^2\right)/\ell^2$.

This means that the probability of an immediate collision from W and W' is $\left(1 - \sum_i p_i^2\right)/\ell$, where we added over the ℓ choices of T.

In the simple case that all the p_i are $1/r$, the difference from the optimal $\sqrt{\pi\ell/2}$ iterations is a factor of $1/\sqrt{1 - 1/r} \approx 1 + 1/(2r)$.

Various heuristics leading to standard $\sqrt{1 - 1/r}$ formula in different ways:

1981 Brent–Pollard;

2001 Teske;

2009 ECC2K-130 paper, eprint 2009/541.

Summing over all (i, j) gives the overall probability $\left(\sum_{i \neq j} p_i p_j\right)/\ell^2 = \left(\sum_{i,j} p_i p_j - \sum_i p_i^2\right)/\ell^2 = \left(1 - \sum_i p_i^2\right)/\ell^2$.

This means that the probability of an immediate collision from W and W' is $\left(1 - \sum_i p_i^2\right)/\ell$, where we added over the ℓ choices of T.

In the simple case that all the p_i are $1/r$, the difference from the optimal $\sqrt{\pi\ell/2}$ iterations is a factor of $1/\sqrt{1 - 1/r} \approx 1 + 1/(2r)$.

Various heuristics leading to standard $\sqrt{1 - 1/r}$ formula in different ways:
1981 Brent–Pollard;
2001 Teske;
2009 ECC2K-130 paper, eprint 2009/541.

Summing over all (i, j) gives the overall probability $\left(\sum_{i \neq j} p_i p_j\right) / \ell^2 = \left(\sum_{i,j} p_i p_j - \sum_i p_i^2\right) / \ell^2 = \left(1 - \sum_i p_i^2\right) / \ell^2$.

This means that the probability of an immediate collision from W and W' is $\left(1 - \sum_i p_i^2\right) / \ell$, where we added over the ℓ choices of T. In the simple case that all the p_i are $1/r$, the difference from the optimal $\sqrt{\pi \ell / 2}$ iterations is a factor of

$1/\sqrt{1 - 1/r} \approx 1 + 1/(2r)$.

Various heuristics leading to standard $\sqrt{1 - 1/r}$ formula in different ways:
1981 Brent–Pollard;
2001 Teske;
2009 ECC2K-130 paper, eprint 2009/541.

2010 Bernstein–Lange: Standard formula is wrong! There is a further slowdown from higher-order anti-collisions: e.g. $W + R_i + R_k \neq W' + R_j + R_l$ if $R_i + R_k = R_j + R_l$.
$\approx 1\%$ slowdown for ECC2K-130.

g over all (i, j)
e overall probability
$p_i p_j \big) / \ell^2 \;\; =$
$_i p_j - \sum_i p_i^2 \big) / \ell^2 \; =$
$p_i^2) / \ell^2$.

ans that the probability
mediate collision from W
is $\left(1 - \sum_i p_i^2\right) / \ell$, where
d over the ℓ choices of T.
mple case that all the p_i
the difference from the
$\sqrt{\pi \ell / 2}$ iterations is a

$1/r \approx 1 + 1/(2r)$.

Various heuristics leading to standard $\sqrt{1 - 1/r}$ formula in different ways:
1981 Brent–Pollard;
2001 Teske;
2009 ECC2K-130 paper, eprint 2009/541.

2010 Bernstein–Lange:
Standard formula is wrong!
There is a further slowdown from higher-order anti-collisions:
e.g. $W + R_i + R_k \neq W' + R_j + R_l$ if $R_i + R_k = R_j + R_l$.
$\approx 1\%$ slowdown for ECC2K-130.

Eliminat

Usual de
keeps tra
with W_i

This req
impleme
or at lea
how ofte

For disti
these va
transmit
which st
e.g. $(W_i$

(i, j)
robability
$=$
$p_i^2\right) / \ell^2 \ =$

he probability
ollision from W
$_i\, p_i^2) / \ell$, where
ℓ choices of T.
that all the p_i
ence from the
erations is a

$+ 1/(2r)$.

Various heuristics leading to standard $\sqrt{1 - 1/r}$ formula in different ways:
1981 Brent–Pollard;
2001 Teske;
2009 ECC2K-130 paper, eprint 2009/541.

2010 Bernstein–Lange:
Standard formula is wrong!
There is a further slowdown from higher-order anti-collisions: e.g. $W + R_i + R_k \neq W' + R_j + R_l$ if $R_i + R_k = R_j + R_l$.
$\approx 1\%$ slowdown for ECC2K-130.

Eliminating storag

Usual description:
keeps track of a_i a
with $W_i = a_i P +$

This requires each
implement arithme
or at least keep tr
how often each R_j

For distinguished p
these values are
transmitted to serv
which stores them
e.g. (W_i, a_i, b_i) (s

=

ility

om W

vhere

of T.

ne p_i

the

a

Various heuristics leading to standard $\sqrt{1-1/r}$ formula in different ways:
1981 Brent–Pollard;
2001 Teske;
2009 ECC2K-130 paper, eprint 2009/541.

2010 Bernstein–Lange:
Standard formula is wrong!
There is a further slowdown from higher-order anti-collisions:
e.g. $W+R_i+R_k \neq W'+R_j+R_l$ if $R_i+R_k=R_j+R_l$.
$\approx 1\%$ slowdown for ECC2K-130.

Eliminating storage

Usual description: each walk keeps track of a_i and b_i with $W_i = a_iP + b_iQ$.

This requires each client to implement arithmetic modul or at least keep track of how often each R_j is used.

For distinguished points these values are transmitted to server (bandv which stores them as e.g. (W_i, a_i, b_i) (space).

Various heuristics leading to standard $\sqrt{1-1/r}$ formula in different ways:
1981 Brent–Pollard;
2001 Teske;
2009 ECC2K-130 paper, eprint 2009/541.

2010 Bernstein–Lange: Standard formula is wrong! There is a further slowdown from higher-order anti-collisions: e.g. $W + R_i + R_k \neq W' + R_j + R_l$ if $R_i + R_k = R_j + R_l$. $\approx$ 1% slowdown for ECC2K-130.

Eliminating storage

Usual description: each walk keeps track of a_i and b_i with $W_i = a_i P + b_i Q$.

This requires each client to implement arithmetic modulo ℓ or at least keep track of how often each R_j is used.

For distinguished points these values are transmitted to server (bandwidth) which stores them as e.g. (W_i, a_i, b_i) (space).

heuristics leading to
$\sqrt{1-1/r}$ formula
ent ways:
ent–Pollard;
ske;
CC2K-130 paper,
009/541.

rnstein–Lange:
d formula is wrong!
a further slowdown
gher-order anti-collisions:
$- R_i + R_k \neq W' + R_j + R_l$
$R_k = R_j + R_l$.
owdown for ECC2K-130.

Eliminating storage

Usual description: each walk keeps track of a_i and b_i with $W_i = a_i P + b_i Q$.

This requires each client to implement arithmetic modulo ℓ or at least keep track of how often each R_j is used.

For distinguished points these values are transmitted to server (bandwidth) which stores them as e.g. (W_i, a_i, b_i) (space).

2009 EC
Rememb
If $W_i =$
distingui
can reco
with a_i,
walk is c
Server st
points; c
coefficie

Our setu
seed; ser
point an
Saves tir

leading to
$\overline{r}$ formula

d;

paper,

ange:
is wrong!
slowdown
anti-collisions:
$\neq W' + R_j + R_l$
$+ R_l$.
or ECC2K-130.

<u>Eliminating storage</u>

Usual description: each walk keeps track of a_i and b_i with $W_i = a_i P + b_i Q$.

This requires each client to implement arithmetic modulo ℓ or at least keep track of how often each R_j is used.

For distinguished points these values are transmitted to server (bandwidth) which stores them as e.g. (W_i, a_i, b_i) (space).

2009 ECC2K-130
Remember where
If $W_i = W_j$ is the
distinguished poin
can recompute the
with a_i, b_i, a_j, anc
walk is determinist
Server stores 2^{45} c
points; only needs
coefficients for 2 c

Our setup: Each w
seed; server stores
point and seed.
Saves time, bandw

ions:

$R_j + R_l$

-130.

Eliminating storage

Usual description: each walk keeps track of a_i and b_i with $W_i = a_i P + b_i Q$.

This requires each client to implement arithmetic modulo ℓ or at least keep track of how often each R_j is used.

For distinguished points these values are transmitted to server (bandwidth) which stores them as e.g. (W_i, a_i, b_i) (space).

2009 ECC2K-130 paper:
Remember where you starte
If $W_i = W_j$ is the collision
distinguished points,
can recompute these walks
with a_i, b_i, a_j, and b_j;
walk is deterministic!
Server stores 2^{45} distinguish
points; only needs to know
coefficients for 2 of them.

Our setup: Each walk remer
seed; server stores distinguis
point and seed.
Saves time, bandwidth, spac

Eliminating storage

Usual description: each walk keeps track of a_i and b_i with $W_i = a_i P + b_i Q$.

This requires each client to implement arithmetic modulo ℓ or at least keep track of how often each R_j is used.

For distinguished points these values are transmitted to server (bandwidth) which stores them as e.g. (W_i, a_i, b_i) (space).

2009 ECC2K-130 paper:
Remember where you started.
If $W_i = W_j$ is the collision of distinguished points,
can recompute these walks with a_i, b_i, a_j, and b_j;
walk is deterministic!
Server stores 2^{45} distinguished points; only needs to know coefficients for 2 of them.

Our setup: Each walk remembers seed; server stores distinguished point and seed.
Saves time, bandwidth, space.

ing storage

escription: each walk
ack of a_i and b_i
$= a_i P + b_i Q$.

uires each client to
nt arithmetic modulo ℓ
st keep track of
en each R_j is used.

nguished points
lues are
ted to server (bandwidth)
ores them as
, a_i, b_i) (space).

2009 ECC2K-130 paper:
Remember where you started.
If $W_i = W_j$ is the collision of
distinguished points,
can recompute these walks
with a_i, b_i, a_j, and b_j;
walk is deterministic!
Server stores 2^{45} distinguished
points; only needs to know
coefficients for 2 of them.

Our setup: Each walk remembers
seed; server stores distinguished
point and seed.
Saves time, bandwidth, space.

Negation

$W = (x,$
have sar
Search f

Search s
only $\lceil \ell /$
speedup

To ensur
Define j
$f(W_i) =$
with, e.g
minimur
This neg
is textbo

e

each walk
and b_i
$b_i Q$.

client to
etic modulo ℓ
ack of
$_i$ is used.

points

ver (bandwidth)
as
pace).

2009 ECC2K-130 paper:
Remember where you started.
If $W_i = W_j$ is the collision of distinguished points,
can recompute these walks with a_i, b_i, a_j, and b_j;
walk is deterministic!
Server stores 2^{45} distinguished points; only needs to know coefficients for 2 of them.

Our setup: Each walk remembers seed; server stores distinguished point and seed.
Saves time, bandwidth, space.

Negation and rho

$W = (x, y)$ and $-$
have same x-coor
Search for x-coor

Search space for c
only $\lceil \ell/2 \rceil$; this gi
speedup . . . if $f(W$

To ensure $f(W_i) =$
Define $j = h(| W_i$
$f(W_i) =| W_i | + c_j$
with, e.g., $| W_i |$ t
minimum of $W_i, -$
This negation spe
is textbook materi

k

o ℓ

width)

2009 ECC2K-130 paper:
Remember where you started.
If $W_i = W_j$ is the collision of distinguished points,
can recompute these walks with a_i, b_i, a_j, and b_j;
walk is deterministic!
Server stores 2^{45} distinguished points; only needs to know coefficients for 2 of them.

Our setup: Each walk remembers seed; server stores distinguished point and seed.
Saves time, bandwidth, space.

<u>Negation and rho</u>

$W = (x, y)$ and $-W = (x, -$
have same x-coordinate.
Search for x-coordinate colli

Search space for collisions is
only $\lceil \ell/2 \rceil$; this gives factor
speedup . . . if $f(W_i) = f(-$

To ensure $f(W_i) = f(-W_i)$
Define $j = h(| W_i |)$ and
$f(W_i) = | W_i | + c_j P + d_j Q,$
with, e.g., $| W_i |$ the lexicog
minimum of $W_i, -W_i$.
This negation speedup
is textbook material.

2009 ECC2K-130 paper:
Remember where you started.
If $W_i = W_j$ is the collision of distinguished points,
can recompute these walks with a_i, b_i, a_j, and b_j;
walk is deterministic!
Server stores 2^{45} distinguished points; only needs to know coefficients for 2 of them.

Our setup: Each walk remembers seed; server stores distinguished point and seed.
Saves time, bandwidth, space.

Negation and rho

$W = (x, y)$ and $-W = (x, -y)$ have same x-coordinate.
Search for x-coordinate collision.

Search space for collisions is only $\lceil \ell/2 \rceil$; this gives factor $\sqrt{2}$ speedup ... if $f(W_i) = f(-W_i)$.

To ensure $f(W_i) = f(-W_i)$:
Define $j = h(|W_i|)$ and $f(W_i) = |W_i| + c_j P + d_j Q$,
with, e.g., $|W_i|$ the lexicographic minimum of $W_i, -W_i$.
This negation speedup is textbook material.

C2K-130 paper:
per where you started.
W_j is the collision of
shed points,
mpute these walks
b_i, a_j, and b_j;
deterministic!
tores 2^{45} distinguished
only needs to know
nts for 2 of them.

up: Each walk remembers
rver stores distinguished
d seed.
me, bandwidth, space.

Negation and rho

$W = (x, y)$ and $-W = (x, -y)$
have same x-coordinate.
Search for x-coordinate collision.

Search space for collisions is
only $\lceil \ell/2 \rceil$; this gives factor $\sqrt{2}$
speedup . . . if $f(W_i) = f(-W_i)$.

To ensure $f(W_i) = f(-W_i)$:
Define $j = h(|W_i|)$ and
$f(W_i) = |W_i| + c_j P + d_j Q$,
with, e.g., $|W_i|$ the lexicographic
minimum of $W_i, -W_i$.
This negation speedup
is textbook material.

Problem
run into
Example
and $h(|$
then W_i
$-W_{i+1}$
$-(| W_i$
$d_j Q = -$
so W_{i+3}
so W_{i+4}

If h map
then exp
with pro
at each
Known i

paper:
you started.
e collision of
ts,
ese walks
d b_j;
tic!
distinguished
to know
of them.

walk remembers
distinguished

width, space.

Negation and rho

$W = (x, y)$ and $-W = (x, -y)$
have same x-coordinate.
Search for x-coordinate collision.

Search space for collisions is
only $\lceil \ell/2 \rceil$; this gives factor $\sqrt{2}$
speedup . . . if $f(W_i) = f(-W_i)$.

To ensure $f(W_i) = f(-W_i)$:
Define $j = h(| W_i |)$ and
$f(W_i) = | W_i | + c_j P + d_j Q$,
with, e.g., $| W_i |$ the lexicographic
minimum of $W_i, -W_i$.
This negation speedup
is textbook material.

Problem: this wal
run into fruitless c
Example: If $| W_{i+}$
and $h(| W_{i+1} |) =$
then $W_{i+2} = f(W$
$-W_{i+1} + c_j P + d$
$-(| W_i | + c_j P +$
$d_j Q = - | W_i |$ so
so $W_{i+3} = W_{i+1}$
so $W_{i+4} = W_{i+2}$ e

If h maps to r diff
then expect this ex
with probability 1/
at each step.
Known issue, not

d.
of

ed

mbers
hed

ce.

Negation and rho

$W = (x, y)$ and $-W = (x, -y)$
have same x-coordinate.
Search for x-coordinate collision.

Search space for collisions is
only $\lceil \ell/2 \rceil$; this gives factor $\sqrt{2}$
speedup . . . if $f(W_i) = f(-W_i)$.

To ensure $f(W_i) = f(-W_i)$:
Define $j = h(\mid W_i \mid)$ and
$f(W_i) = \mid W_i \mid + c_j P + d_j Q$,
with, e.g., $\mid W_i \mid$ the lexicographic
minimum of $W_i, -W_i$.
This negation speedup
is textbook material.

Problem: this walk can
run into fruitless cycles!
Example: If $\mid W_{i+1} \mid = -W_{i}$
and $h(\mid W_{i+1} \mid) = j = h(\mid W$
then $W_{i+2} = f(W_{i+1}) =$
$-W_{i+1} + c_j P + d_j Q =$
$-(\mid W_i \mid + c_j P + d_j Q) + c_j$
$d_j Q = - \mid W_i \mid$ so $\mid W_{i+2} \mid =$
so $W_{i+3} = W_{i+1}$
so $W_{i+4} = W_{i+2}$ etc.

If h maps to r different valu
then expect this example to
with probability $1/(2r)$
at each step.
Known issue, not quite textb

Negation and rho

$W = (x, y)$ and $-W = (x, -y)$ have same x-coordinate.
Search for x-coordinate collision.

Search space for collisions is only $\lceil \ell/2 \rceil$; this gives factor $\sqrt{2}$ speedup . . . if $f(W_i) = f(-W_i)$.

To ensure $f(W_i) = f(-W_i)$:
Define $j = h(|W_i|)$ and $f(W_i) = |W_i| + c_j P + d_j Q$, with, e.g., $|W_i|$ the lexicographic minimum of $W_i, -W_i$.
This negation speedup is textbook material.

Problem: this walk can run into fruitless cycles!
Example: If $|W_{i+1}| = -W_{i+1}$ and $h(|W_{i+1}|) = j = h(|W_i|)$ then $W_{i+2} = f(W_{i+1}) = -W_{i+1} + c_j P + d_j Q = -(|W_i| + c_j P + d_j Q) + c_j P + d_j Q = -|W_i|$ so $|W_{i+2}| = |W_i|$
so $W_{i+3} = W_{i+1}$
so $W_{i+4} = W_{i+2}$ etc.

If h maps to r different values then expect this example to occur with probability $1/(2r)$ at each step.
Known issue, not quite textbook.

n and rho

$, y)$ and $-W = (x, -y)$

me x-coordinate.

or x-coordinate collision.

pace for collisions is

$2\rceil$; this gives factor $\sqrt{2}$

. . . if $f(W_i) = f(-W_i)$.

re $f(W_i) = f(-W_i)$:

$= h(| W_i |)$ and

$=| W_i | + c_j P + d_j Q$,

g., $| W_i |$ the lexicographic

n of $W_i, -W_i$.

gation speedup

ook material.

Problem: this walk can run into fruitless cycles!

Example: If $| W_{i+1} | = -W_{i+1}$ and $h(| W_{i+1} |) = j = h(| W_i |)$ then $W_{i+2} = f(W_{i+1}) =$
$-W_{i+1} + c_j P + d_j Q =$
$-(| W_i | + c_j P + d_j Q) + c_j P +$
$d_j Q = - | W_i |$ so $| W_{i+2} | = | W_i |$
so $W_{i+3} = W_{i+1}$
so $W_{i+4} = W_{i+2}$ etc.

If h maps to r different values then expect this example to occur with probability $1/(2r)$ at each step.
Known issue, not quite textbook.

Eliminat

Issue of

and seve

See appe

2011/00

and hist

Summar

got it wr

$W = (x, -y)$
dinate.
dinate collision.

ollisions is
ves factor $\sqrt{2}$
$V_i) = f(-W_i)$.

$= f(-W_i)$:
$|)$ and
$_j P + d_j Q$,
he lexicographic
$-W_i$.
edup
al.

Problem: this walk can
run into fruitless cycles!
Example: If $| W_{i+1} | = -W_{i+1}$
and $h(| W_{i+1} |) = j = h(| W_i |)$
then $W_{i+2} = f(W_{i+1}) =$
$-W_{i+1} + c_j P + d_j Q =$
$-(| W_i | + c_j P + d_j Q) + c_j P +$
$d_j Q = - | W_i |$ so $| W_{i+2} | = | W_i |$
so $W_{i+3} = W_{i+1}$
so $W_{i+4} = W_{i+2}$ etc.

If h maps to r different values
then expect this example to occur
with probability $1/(2r)$
at each step.
Known issue, not quite textbook.

Eliminating fruitles

Issue of fruitless cy
and several fixes a
See appendix of fu
2011/003 for even
and historical com

Summary: most o
got it wrong.

$-y)$

sion.

$\sqrt{2}$

$W_i)$.

:

raphic

Problem: this walk can run into fruitless cycles!

Example: If $|W_{i+1}| = -W_{i+1}$ and $h(|W_{i+1}|) = j = h(|W_i|)$ then $W_{i+2} = f(W_{i+1}) =$
$-W_{i+1} + c_j P + d_j Q =$
$-(|W_i| + c_j P + d_j Q) + c_j P +$
$d_j Q = -|W_i|$ so $|W_{i+2}| = |W_i|$
so $W_{i+3} = W_{i+1}$
so $W_{i+4} = W_{i+2}$ etc.

If h maps to r different values then expect this example to occur with probability $1/(2r)$ at each step.
Known issue, not quite textbook.

Eliminating fruitless cycles

Issue of fruitless cycles is kn
and several fixes are propose
See appendix of full version
2011/003 for even more det
and historical comments.

Summary: most of them got it wrong.

Problem: this walk can run into fruitless cycles!

Example: If $|W_{i+1}| = -W_{i+1}$ and $h(|W_{i+1}|) = j = h(|W_i|)$ then $W_{i+2} = f(W_{i+1}) = -W_{i+1} + c_j P + d_j Q = -(|W_i| + c_j P + d_j Q) + c_j P + d_j Q = -|W_i|$ so $|W_{i+2}| = |W_i|$

so $W_{i+3} = W_{i+1}$

so $W_{i+4} = W_{i+2}$ etc.

If h maps to r different values then expect this example to occur with probability $1/(2r)$ at each step.

Known issue, not quite textbook.

Eliminating fruitless cycles

Issue of fruitless cycles is known and several fixes are proposed. See appendix of full version ePrint 2011/003 for even more details and historical comments.

Summary: most of them got it wrong.

Problem: this walk can run into fruitless cycles!
Example: If $| W_{i+1} |= -W_{i+1}$ and $h(| W_{i+1} |) = j = h(| W_i |)$ then $W_{i+2} = f(W_{i+1}) = -W_{i+1} + c_j P + d_j Q = -(| W_i | + c_j P + d_j Q) + c_j P + d_j Q = - | W_i |$ so $| W_{i+2} |=| W_i |$
so $W_{i+3} = W_{i+1}$
so $W_{i+4} = W_{i+2}$ etc.

If h maps to r different values then expect this example to occur with probability $1/(2r)$ at each step.
Known issue, not quite textbook.

Eliminating fruitless cycles

Issue of fruitless cycles is known and several fixes are proposed.
See appendix of full version ePrint 2011/003 for even more details and historical comments.

Summary: most of them got it wrong.

So what to do?
Choose a big r, e.g. $r = 2048$.
$1/(2r) = 1/4096$ small;
cycles infrequent.

: this walk can
fruitless cycles!
e: If $|W_{i+1}| = -W_{i+1}$
$W_{i+1}|) = j = h(|W_i|)$
$_{+2} = f(W_{i+1}) =$
$+ c_j P + d_j Q =$
$| + c_j P + d_j Q) + c_j P +$
$- |W_i|$ so $|W_{i+2}| = |W_i|$
$= W_{i+1}$
$= W_{i+2}$ etc.

os to r different values
pect this example to occur
bability $1/(2r)$
step.
ssue, not quite textbook.

Eliminating fruitless cycles

Issue of fruitless cycles is known and several fixes are proposed. See appendix of full version ePrint 2011/003 for even more details and historical comments.

Summary: most of them got it wrong.

So what to do?
Choose a big r, e.g. $r = 2048$.
$1/(2r) = 1/4096$ small;
cycles infrequent.

Define |
(x, y) fo
or
$(x, -y)$

Precomp
$R_0, R_1, .$
random

k can
ycles!
$_1 | = -W_{i+1}$
$j = h(| W_i |)$
$_{i+1}) =$
$_j Q =$
$d_j Q) + c_j P +$
$| W_{i+2} | = | W_i |$

etc.

ferent values
xample to occur
$/(2r)$

quite textbook.

Eliminating fruitless cycles

Issue of fruitless cycles is known and several fixes are proposed. See appendix of full version ePrint 2011/003 for even more details and historical comments.

Summary: most of them got it wrong.

So what to do?
Choose a big r, e.g. $r = 2048$.
$1/(2r) = 1/4096$ small;
cycles infrequent.

Define $| (x, y) |$ to
(x, y) for $y \in \{0, 2$
or
$(x, -y)$ for $y \in \{1$

Precompute points
$R_0, R_1, \ldots, R_{r-1}$
random multiples

$+1$

$V_i \mid)$

$_i P +$

$= \mid W_i \mid$

ues

occur

book.

Eliminating fruitless cycles

Issue of fruitless cycles is known and several fixes are proposed. See appendix of full version ePrint 2011/003 for even more details and historical comments.

Summary: most of them got it wrong.

So what to do?
Choose a big r, e.g. $r = 2048$.
$1/(2r) = 1/4096$ small;
cycles infrequent.

Define $\mid (x, y) \mid$ to mean
(x, y) for $y \in \{0, 2, 4, \ldots, p$
or
$(x, -y)$ for $y \in \{1, 3, 5, \ldots,$

Precompute points
$R_0, R_1, \ldots, R_{r-1}$ as known
random multiples of P.

Eliminating fruitless cycles

Issue of fruitless cycles is known and several fixes are proposed. See appendix of full version ePrint 2011/003 for even more details and historical comments.

Summary: most of them got it wrong.

So what to do?
Choose a big r, e.g. $r = 2048$.
$1/(2r) = 1/4096$ small;
cycles infrequent.

Define $|\,(x, y)\,|$ to mean
(x, y) for $y \in \{0, 2, 4, \ldots, p-1\}$
or
$(x, -y)$ for $y \in \{1, 3, 5, \ldots, p-2\}$.

Precompute points
$R_0, R_1, \ldots, R_{r-1}$ as known
random multiples of P.

Eliminating fruitless cycles

Issue of fruitless cycles is known and several fixes are proposed. See appendix of full version ePrint 2011/003 for even more details and historical comments.

Summary: most of them got it wrong.

So what to do?
Choose a big r, e.g. $r = 2048$.
$1/(2r) = 1/4096$ small;
cycles infrequent.

Define $|\,(x, y)\,|$ to mean
(x, y) for $y \in \{0, 2, 4, \ldots, p-1\}$
or
$(x, -y)$ for $y \in \{1, 3, 5, \ldots, p-2\}$.

Precompute points
$R_0, R_1, \ldots, R_{r-1}$ as known random multiples of P.
Can do full scalar multiplication in inversion-free coordinates!
Start each walk at a point
$W_0 = |\, b_0 Q \,|$,
where b_0 is chosen randomly.
Compute $W_1, W_2, \ldots$ as
$W_{i+1} = |\, W_i + R_{h(W_i)} \,|$.

ing fruitless cycles

fruitless cycles is known
eral fixes are proposed.
endix of full version ePrint
03 for even more details
orical comments.

y: most of them
rong.

to do?
a big r, e.g. $r = 2048$.
$= 1/4096$ small;
frequent.

Define $|\,(x, y)\,|$ to mean
(x, y) for $y \in \{0, 2, 4, \ldots, p-1\}$
or
$(x, -y)$ for $y \in \{1, 3, 5, \ldots, p-2\}$.

Precompute points
$R_0, R_1, \ldots, R_{r-1}$ as known
random multiples of P.
Can do full scalar multiplication in
inversion-free coordinates!
Start each walk at a point
$W_0 = |\, b_0 Q \,|$,
where b_0 is chosen randomly.
Compute $W_1, W_2, \ldots$ as
$W_{i+1} = |\, W_i + R_{h(W_i)} \,|$.

Occasion
check fo
of length
For thos
definitio
Comput
whether
If W_{i-1}
If W_{i-1}
$W_i = |\, 2$
where m
lexicogra
Doublin
makes it

ss cycles

ycles is known
re proposed.
ull version ePrint
more details
ments.

f them

g. $r = 2048$.
small;

Define $|\,(x, y)\,|$ to mean
(x, y) for $y \in \{0, 2, 4, \ldots, p-1\}$
or
$(x, -y)$ for $y \in \{1, 3, 5, \ldots, p-2\}$.

Precompute points
$R_0, R_1, \ldots, R_{r-1}$ as known
random multiples of P.
Can do full scalar multiplication in
inversion-free coordinates!
Start each walk at a point
$W_0 = |\,b_0 Q\,|$,
where b_0 is chosen randomly.
Compute $W_1, W_2, \ldots$ as
$W_{i+1} = |\,W_i + R_{h(W_i)}\,|$.

Occasionally, ever
check for fruitless
of length 2.
For those cases ch
definition of W_i as
Compute W_{i-1} an
whether $W_{i-1} = W$
If $W_{i-1} \neq W_{i-3}$, p
If $W_{i-1} = W_{i-3}$, p
$W_i = |\,2\min\{W_{i-1}$
where min means
lexicographic mini
Doubling the poin
makes it escape th

own
ed.
ePrint
ails

48.

Define $|\,(x,y)\,|$ to mean
(x,y) for $y \in \{0,2,4,\ldots,p-1\}$
or
$(x,-y)$ for $y \in \{1,3,5,\ldots,p-2\}$.

Precompute points
$R_0, R_1, \ldots, R_{r-1}$ as known
random multiples of P.
Can do full scalar multiplication in
inversion-free coordinates!
Start each walk at a point
$W_0 = |\, b_0 Q \,|$,
where b_0 is chosen randomly.
Compute $W_1, W_2, \ldots$ as
$W_{i+1} = |\, W_i + R_{h(W_i)} \,|$.

Occasionally, every w iterat
check for fruitless cycles
of length 2.
For those cases change the
definition of W_i as follows:
Compute W_{i-1} and check
whether $W_{i-1} = W_{i-3}$.
If $W_{i-1} \neq W_{i-3}$, put $W_i =$
If $W_{i-1} = W_{i-3}$, put
$W_i = |\, 2\min\{W_{i-1}, W_{i-2}\} \,|$,
where min means
lexicographic minimum.
Doubling the point
makes it escape the cycle.

Define $| (x, y) |$ to mean
(x, y) for $y \in \{0, 2, 4, \ldots, p-1\}$
or
$(x, -y)$ for $y \in \{1, 3, 5, \ldots, p-2\}$.

Precompute points
$R_0, R_1, \ldots, R_{r-1}$ as known
random multiples of P.
Can do full scalar multiplication in
inversion-free coordinates!
Start each walk at a point
$W_0 = | b_0 Q |$,
where b_0 is chosen randomly.
Compute $W_1, W_2, \ldots$ as
$W_{i+1} = | W_i + R_{h(W_i)} |$.

Occasionally, every w iterations,
check for fruitless cycles
of length 2.
For those cases change the
definition of W_i as follows:
Compute W_{i-1} and check
whether $W_{i-1} = W_{i-3}$.
If $W_{i-1} \neq W_{i-3}$, put $W_i = W_{i-1}$.
If $W_{i-1} = W_{i-3}$, put
$W_i = | 2 \min\{W_{i-1}, W_{i-2}\} |$,
where min means
lexicographic minimum.
Doubling the point
makes it escape the cycle.

$(x, y) \mid$ to mean
r $y \in \{0, 2, 4, \ldots, p-1\}$

for $y \in \{1, 3, 5, \ldots, p-2\}$.

oute points
$\ldots, R_{r-1}$ as known
multiples of P.
full scalar multiplication in
n-free coordinates!
ch walk at a point
$_0Q \mid$,
$_0$ is chosen randomly.
e $W_1, W_2, \ldots$ as
$\mid W_i + R_{h(W_i)} \mid$.

Occasionally, every w iterations,
check for fruitless cycles
of length 2.
For those cases change the
definition of W_i as follows:
Compute W_{i-1} and check
whether $W_{i-1} = W_{i-3}$.
If $W_{i-1} \neq W_{i-3}$, put $W_i = W_{i-1}$.
If $W_{i-1} = W_{i-3}$, put
$W_i = \mid 2\min\{W_{i-1}, W_{i-2}\} \mid$,
where min means
lexicographic minimum.
Doubling the point
makes it escape the cycle.

Cycles o
occur fa
Cycles o
are dete
for cycle
so skip i

Same wa
define W
| 2min{W
W
W
if trappe
and W_i

mean
$2, 4, \ldots, p-1\}$

$., 3, 5, \ldots, p-2\}$.

s
as known
of P.
multiplication in
rdinates!
a point

randomly.
... as
(W_i) |.

Occasionally, every w iterations, check for fruitless cycles of length 2.
For those cases change the definition of W_i as follows:
Compute W_{i-1} and check whether $W_{i-1} = W_{i-3}$.
If $W_{i-1} \neq W_{i-3}$, put $W_i = W_{i-1}$.
If $W_{i-1} = W_{i-3}$, put
$W_i = | 2 \min\{W_{i-1}, W_{i-2}\} |$,
where min means lexicographic minimum.
Doubling the point makes it escape the cycle.

Cycles of length 4
occur far less frequ
Cycles of length 4
are detected when
for cycles of length
so skip individual

Same way of escap
define $W_i =$
$| 2\min\{W_{i-1}, W_{i-}$
W_{i-5}, W_{i-6}
W_{i-9}, W_{i-10}
if trapped
and $W_i = W_{i-1}$ ot

$-1\}$

$p-2\}$.

tion in

/.

Occasionally, every w iterations, check for fruitless cycles of length 2.
For those cases change the definition of W_i as follows:
Compute W_{i-1} and check whether $W_{i-1} = W_{i-3}$.
If $W_{i-1} \neq W_{i-3}$, put $W_i = W_{i-1}$.
If $W_{i-1} = W_{i-3}$, put
$W_i = |\, 2 \min\{W_{i-1}, W_{i-2}\} \,|$,
where min means lexicographic minimum.
Doubling the point makes it escape the cycle.

Cycles of length 4, 6, or 12 occur far less frequently.
Cycles of length 4, or 6 are detected when checking for cycles of length 12;
so skip individual ones.

Same way of escape:
define $W_i =$
$|\, 2\min\{W_{i-1}, W_{i-2}, W_{i-3}, W$
$W_{i-5}, W_{i-6}, W_{i-7}, W_i$
$W_{i-9}, W_{i-10}, W_{i-11}, W$
if trapped
and $W_i = W_{i-1}$ otherwise.

Occasionally, every w iterations,
check for fruitless cycles
of length 2.
For those cases change the
definition of W_i as follows:
Compute W_{i-1} and check
whether $W_{i-1} = W_{i-3}$.
If $W_{i-1} \neq W_{i-3}$, put $W_i = W_{i-1}$.
If $W_{i-1} = W_{i-3}$, put
$W_i = \mid 2\min\{W_{i-1}, W_{i-2}\} \mid$,
where min means
lexicographic minimum.
Doubling the point
makes it escape the cycle.

Cycles of length 4, 6, or 12
occur far less frequently.
Cycles of length 4, or 6
are detected when checking
for cycles of length 12;
so skip individual ones.

Same way of escape:
define $W_i =$
$$\mid 2\min\{W_{i-1}, W_{i-2}, W_{i-3}, W_{i-4}, W_{i-5}, W_{i-6}, W_{i-7}, W_{i-8}, W_{i-9}, W_{i-10}, W_{i-11}, W_{i-12}\} \mid$$
if trapped
and $W_i = W_{i-1}$ otherwise.

nally, every w iterations,
r fruitless cycles
n 2.
e cases change the
n of W_i as follows:
e W_{i-1} and check
$W_{i-1} = W_{i-3}$.
$\neq W_{i-3}$, put $W_i = W_{i-1}$.
$= W_{i-3}$, put
min$\{W_{i-1}, W_{i-2}\}$ |,
in means
aphic minimum.
g the point
t escape the cycle.

Cycles of length 4, 6, or 12 occur far less frequently.
Cycles of length 4, or 6 are detected when checking for cycles of length 12; so skip individual ones.

Same way of escape: define $W_i =$

$$| \, 2\min\{W_{i-1}, W_{i-2}, W_{i-3}, W_{i-4}, W_{i-5}, W_{i-6}, W_{i-7}, W_{i-8}, W_{i-9}, W_{i-10}, W_{i-11}, W_{i-12}\} \, |$$

if trapped and $W_i = W_{i-1}$ otherwise.

Do not s
When ch
store on
W_{i-13} (
comparis
smallest
(to escap
For large
look for
in genera
fruitless
up to $\approx$

y w iterations,
cycles

ange the
s follows:
d check
V_{i-3}.
out $W_i = W_{i-1}$.
out
$_1, W_{i-2}\} |$,

mum.
t
ne cycle.

Cycles of length 4, 6, or 12
occur far less frequently.
Cycles of length 4, or 6
are detected when checking
for cycles of length 12;
so skip individual ones.

Same way of escape:
define $W_i =$
$| \, 2\min\{W_{i-1}, W_{i-2}, W_{i-3}, W_{i-4}, W_{i-5}, W_{i-6}, W_{i-7}, W_{i-8}, W_{i-9}, W_{i-10}, W_{i-11}, W_{i-12}\} \, |$
if trapped
and $W_i = W_{i-1}$ otherwise.

Do not store all th
When checking for
store only potentia
W_{i-13} (one coordi
comparison) and t
smallest point enc
(to escape).

For large DLP
look for larger cyc
in general, look fo
fruitless cycles of e
up to $\approx (\log \ell)/(\mathrm{lo}$

ions,

W_{i-1}.

Cycles of length 4, 6, or 12 occur far less frequently. Cycles of length 4, or 6 are detected when checking for cycles of length 12; so skip individual ones.

Same way of escape: define $W_i =$

$$| 2\min\{W_{i-1}, W_{i-2}, W_{i-3}, W_{i-4}, W_{i-5}, W_{i-6}, W_{i-7}, W_{i-8}, W_{i-9}, W_{i-10}, W_{i-11}, W_{i-12}\} |$$

if trapped and $W_i = W_{i-1}$ otherwise.

Do not store all these points

When checking for cycle, store only potential entry po W_{i-13} (one coordinate, for comparison) and the smallest point encountered s (to escape).

For large DLP look for larger cycles; in general, look for fruitless cycles of even lengt up to $\approx (\log \ell)/(\log r)$.

Cycles of length 4, 6, or 12
occur far less frequently.
Cycles of length 4, or 6
are detected when checking
for cycles of length 12;
so skip individual ones.

Same way of escape:
define $W_i =$
$$| 2\min\{W_{i-1}, W_{i-2}, W_{i-3}, W_{i-4}, W_{i-5}, W_{i-6}, W_{i-7}, W_{i-8}, W_{i-9}, W_{i-10}, W_{i-11}, W_{i-12}\} |$$
if trapped
and $W_i = W_{i-1}$ otherwise.

Do not store all these points!

When checking for cycle,
store only potential entry point
W_{i-13} (one coordinate, for
comparison) and the
smallest point encountered since
(to escape).

For large DLP
look for larger cycles;
in general, look for
fruitless cycles of even lengths
up to $\approx (\log \ell)/(\log r)$.

f length 4, 6, or 12
r less frequently.
f length 4, or 6
cted when checking
es of length 12;
ndividual ones.

ay of escape:
$/_i =$
$W_{i-1}, W_{i-2}, W_{i-3}, W_{i-4},$
$_{i-5}, W_{i-6}, W_{i-7}, W_{i-8},$
$_{i-9}, W_{i-10}, W_{i-11}, W_{i-12}\} \mid$
ed
$= W_{i-1}$ otherwise.

Do not store all these points!

When checking for cycle, store only potential entry point W_{i-13} (one coordinate, for comparison) and the smallest point encountered since (to escape).

For large DLP look for larger cycles; in general, look for fruitless cycles of even lengths up to $\approx (\log \ell)/(\log r)$.

How to

Fruitless
with pro
These cy
until det
After w
probabili
wastes ≈
(on aver

Do not
as small
If a cycl
the chec

, 6, or 12
uently.
, or 6
checking
n 12;
ones.

pe:

$_2, W_{i-3}, W_{i-4},$
$, W_{i-7}, W_{i-8},$
$_0, W_{i-11}, W_{i-12}\} \mid$

therwise.

Do not store all these points!

When checking for cycle, store only potential entry point W_{i-13} (one coordinate, for comparison) and the smallest point encountered since (to escape).

For large DLP look for larger cycles; in general, look for fruitless cycles of even lengths up to $\approx (\log \ell)/(\log r)$.

How to choose w?

Fruitless cycles of
with probability $\approx$
These cycles persi
until detected.
After w iterations,
probability of cycl
wastes $\approx w/2$ iter
(on average) if it

Do not choose w
as small as possibl
If a cycle has *not*
the check wastes

$/_{i-4},$

$_{-8},$

$W_{i-12}\} \mid$

Do not store all these points!

When checking for cycle, store only potential entry point W_{i-13} (one coordinate, for comparison) and the smallest point encountered since (to escape).

For large DLP look for larger cycles; in general, look for fruitless cycles of even lengths up to $\approx (\log \ell)/(\log r)$.

How to choose w?

Fruitless cycles of length 2 a
with probability $\approx 1/(2r)$.
These cycles persist
until detected.
After w iterations,
probability of cycle $\approx w/(2r$
wastes $\approx w/2$ iterations
(on average) if it does appea

Do not choose w
as small as possible!
If a cycle has *not* appeared
the check wastes an iteratio

Do not store all these points!

When checking for cycle, store only potential entry point W_{i-13} (one coordinate, for comparison) and the smallest point encountered since (to escape).

For large DLP look for larger cycles; in general, look for fruitless cycles of even lengths up to $\approx (\log \ell)/(\log r)$.

How to choose w?

Fruitless cycles of length 2 appear with probability $\approx 1/(2r)$. These cycles persist until detected. After w iterations, probability of cycle $\approx w/(2r)$, wastes $\approx w/2$ iterations (on average) if it does appear.

Do not choose w as small as possible! If a cycle has *not* appeared then the check wastes an iteration.

store all these points!

hecking for cycle,
ly potential entry point
one coordinate, for
son) and the
point encountered since
pe).

e DLP
larger cycles;
al, look for
cycles of even lengths
$(\log \ell)/(\log r)$.

How to choose w?

Fruitless cycles of length 2 appear
with probability $\approx 1/(2r)$.
These cycles persist
until detected.
After w iterations,
probability of cycle $\approx w/(2r)$,
wastes $\approx w/2$ iterations
(on average) if it does appear.

Do not choose w
as small as possible!
If a cycle has *not* appeared then
the check wastes an iteration.

The ove
$1 + w^2/$
To minir
$1/w + w$

Cycles o
probabili
optimal
$\approx 1/r^{c/2}$
Loss rap
as c incr
Can use
to check

hese points!

r cycle,

al entry point

inate, for

he

ountered since

les;

r

even lengths

og r).

How to choose w?

Fruitless cycles of length 2 appear with probability $\approx 1/(2r)$.

These cycles persist until detected.

After w iterations, probability of cycle $\approx w/(2r)$, wastes $\approx w/2$ iterations (on average) if it does appear.

Do not choose w as small as possible!

If a cycle has *not* appeared then the check wastes an iteration.

The overall loss is

$1 + w^2/(4r)$ iterat

To minimize the q

$1/w + w/(4r)$ we

Cycles of length 2

probability $\approx 1/r^{c}$

optimal checking f

$\approx 1/r^{c/2}$.

Loss rapidly disapp

as c increases.

Can use lcm of cyc

to check.

s!

oint

since

hs

How to choose w?

Fruitless cycles of length 2 appear with probability $\approx 1/(2r)$.
These cycles persist until detected.
After w iterations, probability of cycle $\approx w/(2r)$, wastes $\approx w/2$ iterations (on average) if it does appear.

Do not choose w as small as possible!
If a cycle has *not* appeared then the check wastes an iteration.

The overall loss is approxima
$1 + w^2/(4r)$ iterations out o
To minimize the quotient
$1/w + w/(4r)$ we take $w \approx$

Cycles of length $2c$ appear w
probability $\approx 1/r^c$,
optimal checking frequency
$\approx 1/r^{c/2}$.
Loss rapidly disappears as c increases.
Can use lcm of cycle lengths
to check.

How to choose w?

Fruitless cycles of length 2 appear with probability $\approx 1/(2r)$.
These cycles persist until detected.
After w iterations, probability of cycle $\approx w/(2r)$, wastes $\approx w/2$ iterations (on average) if it does appear.

Do not choose w as small as possible!
If a cycle has *not* appeared then the check wastes an iteration.

The overall loss is approximately $1 + w^2/(4r)$ iterations out of w.
To minimize the quotient $1/w + w/(4r)$ we take $w \approx 2\sqrt{r}$.

Cycles of length $2c$ appear with probability $\approx 1/r^c$, optimal checking frequency is $\approx 1/r^{c/2}$.
Loss rapidly disappears as c increases.
Can use lcm of cycle lengths to check.

choose w?

cycles of length 2 appear
bability $\approx 1/(2r)$.
ycles persist
tected.
iterations,
ity of cycle $\approx w/(2r)$,
$\approx w/2$ iterations
rage) if it does appear.

choose w
as possible!
e has *not* appeared then
ck wastes an iteration.

The overall loss is approximately $1 + w^2/(4r)$ iterations out of w. To minimize the quotient $1/w + w/(4r)$ we take $w \approx 2\sqrt{r}$.

Cycles of length $2c$ appear with probability $\approx 1/r^c$, optimal checking frequency is $\approx 1/r^{c/2}$. Loss rapidly disappears as c increases. Can use lcm of cycle lengths to check.

Concrete

Use $r =$
every 48
Check fo
frequent
Unify th
6-cycles
every 49

Choice o
$r = 512$
for 2-cyc
In genera
$\approx$ doubl
is reduce

?

length 2 appear
$1/(2r)$.
st

,
e $\approx w/(2r)$,
ations
does appear.

le!
appeared then
an iteration.

The overall loss is approximately $1 + w^2/(4r)$ iterations out of w.
To minimize the quotient $1/w + w/(4r)$ we take $w \approx 2\sqrt{r}$.

Cycles of length $2c$ appear with probability $\approx 1/r^c$,
optimal checking frequency is $\approx 1/r^{c/2}$.
Loss rapidly disappears as c increases.
Can use lcm of cycle lengths to check.

Concrete example:

Use $r = 2048$. Ch
every 48 iterations
Check for larger cy
frequently.
Unify the checks f
6-cycles into a che
every 49152 iterat

Choice of r has bi
$r = 512$ calls for c
for 2-cycles every
In general, negatio
$\approx$ doubles when ta
is reduced by facto

appear

r),

ar.

then

n.

The overall loss is approximately $1 + w^2/(4r)$ iterations out of w. To minimize the quotient $1/w + w/(4r)$ we take $w \approx 2\sqrt{r}$.

Cycles of length $2c$ appear with probability $\approx 1/r^c$, optimal checking frequency is $\approx 1/r^{c/2}$. Loss rapidly disappears as c increases. Can use lcm of cycle lengths to check.

Concrete example: 112-bit D

Use $r = 2048$. Check for 2- every 48 iterations. Check for larger cycles much frequently. Unify the checks for 4-cycles 6-cycles into a check for 12- every 49152 iterations.

Choice of r has big impact! $r = 512$ calls for checking for 2-cycles every 24 iteratio In general, negation overhea $\approx$ doubles when table size is reduced by factor of 4.

The overall loss is approximately
$1 + w^2/(4r)$ iterations out of w.
To minimize the quotient
$1/w + w/(4r)$ we take $w \approx 2\sqrt{r}$.

Cycles of length $2c$ appear with
probability $\approx 1/r^c$,
optimal checking frequency is
$\approx 1/r^{c/2}$.
Loss rapidly disappears
as c increases.
Can use lcm of cycle lengths
to check.

<u>Concrete example: 112-bit DLP</u>

Use $r = 2048$. Check for 2-cycles
every 48 iterations.
Check for larger cycles much less
frequently.
Unify the checks for 4-cycles and
6-cycles into a check for 12-cycles
every 49152 iterations.

Choice of r has big impact!
$r = 512$ calls for checking
for 2-cycles every 24 iterations.
In general, negation overhead
$\approx$ doubles when table size
is reduced by factor of 4.

rall loss is approximately
$(4r)$ iterations out of w.
mize the quotient
$v/(4r)$ we take $w \approx 2\sqrt{r}$.

f length $2c$ appear with
ity $\approx 1/r^c$,
checking frequency is
^{2}.
idly disappears
reases.
lcm of cycle lengths
.

Concrete example: 112-bit DLP

Use $r = 2048$. Check for 2-cycles every 48 iterations.
Check for larger cycles much less frequently.
Unify the checks for 4-cycles and 6-cycles into a check for 12-cycles every 49152 iterations.

Choice of r has big impact!
$r = 512$ calls for checking for 2-cycles every 24 iterations.
In general, negation overhead $\approx$ doubles when table size is reduced by factor of 4.

Bernstei
(PKC 20

Our soft
random
(with no
in 35.6 F

For com
Bos–Kai
Montgo
uses 65

approximately
tions out of w.
uotient
take $w \approx 2\sqrt{r}$.

c appear with
f,
frequency is

pears

cle lengths

Concrete example: 112-bit DLP

Use $r = 2048$. Check for 2-cycles every 48 iterations.
Check for larger cycles much less frequently.
Unify the checks for 4-cycles and 6-cycles into a check for 12-cycles every 49152 iterations.

Choice of r has big impact!
$r = 512$ calls for checking for 2-cycles every 24 iterations.
In general, negation overhead $\approx$ doubles when table size is reduced by factor of 4.

Bernstein, Lange,
(PKC 2011):

Our software solve
random ECDL on
(with no precompu
in 35.6 PS3 years

For comparison:
Bos–Kaihara–Klei
Montgomery softw
uses 65 PS3 years

ately
of w.

$2\sqrt{r}$.

with

is

s

Concrete example: 112-bit DLP

Use $r = 2048$. Check for 2-cycles every 48 iterations.
Check for larger cycles much less frequently.
Unify the checks for 4-cycles and 6-cycles into a check for 12-cycles every 49152 iterations.

Choice of r has big impact!
$r = 512$ calls for checking for 2-cycles every 24 iterations.
In general, negation overhead $\approx$ doubles when table size is reduced by factor of 4.

Bernstein, Lange, Schwabe
(PKC 2011):

Our software solves random ECDL on the same
(with no precomputation)
in 35.6 PS3 years on averag

For comparison:
Bos–Kaihara–Kleinjung–Len
Montgomery software
uses 65 PS3 years on averag

Concrete example: 112-bit DLP

Use $r = 2048$. Check for 2-cycles every 48 iterations.
Check for larger cycles much less frequently.
Unify the checks for 4-cycles and 6-cycles into a check for 12-cycles every 49152 iterations.

Choice of r has big impact!
$r = 512$ calls for checking for 2-cycles every 24 iterations.
In general, negation overhead $\approx$ doubles when table size is reduced by factor of 4.

Bernstein, Lange, Schwabe (PKC 2011):

Our software solves random ECDL on the same curve (with no precomputation) in 35.6 PS3 years on average.

For comparison:
Bos–Kaihara–Kleinjung–Lenstra–Montgomery software uses 65 PS3 years on average.

Concrete example: 112-bit DLP

Use $r = 2048$. Check for 2-cycles every 48 iterations.
Check for larger cycles much less frequently.
Unify the checks for 4-cycles and 6-cycles into a check for 12-cycles every 49152 iterations.

Choice of r has big impact!
$r = 512$ calls for checking for 2-cycles every 24 iterations.
In general, negation overhead $\approx$ doubles when table size is reduced by factor of 4.

Bernstein, Lange, Schwabe (PKC 2011):

Our software solves random ECDL on the same curve (with no precomputation) in 35.6 PS3 years on average.

For comparison:
Bos–Kaihara–Kleinjung–Lenstra–Montgomery software uses 65 PS3 years on average.

First big speedup:
We use the negation map.
Second speedup: Fast arithmetic.

e example: 112-bit DLP

2048. Check for 2-cycles
iterations.
or larger cycles much less
ly.
e checks for 4-cycles and
into a check for 12-cycles
152 iterations.

of r has big impact!
calls for checking
cles every 24 iterations.
al, negation overhead
es when table size
ed by factor of 4.

Bernstein, Lange, Schwabe
(PKC 2011):

Our software solves
random ECDL on the same curve
(with no precomputation)
in 35.6 PS3 years on average.

For comparison:
Bos–Kaihara–Kleinjung–Lenstra–
Montgomery software
uses 65 PS3 years on average.

First big speedup:
We use the negation map.
Second speedup: Fast arithmetic.

Why are

We only
not 200
Don't w
to show
the right

112-bit DLP

heck for 2-cycles
s.
ycles much less

or 4-cycles and
eck for 12-cycles
ions.

g impact!
checking
24 iterations.
on overhead
able size
or of 4.

Bernstein, Lange, Schwabe (PKC 2011):

Our software solves random ECDL on the same curve (with no precomputation) in 35.6 PS3 years on average.

For comparison: Bos–Kaihara–Kleinjung–Lenstra–Montgomery software uses 65 PS3 years on average.

First big speedup:
We use the negation map.
Second speedup: Fast arithmetic.

Why are we confid

We only have 1 P
not 200 used in th
Don't want to wai
to show that we a
the right thing.

DLP

cycles

n less

s and
-cycles

ons.
d

Bernstein, Lange, Schwabe (PKC 2011):

Our software solves random ECDL on the same curve (with no precomputation) in 35.6 PS3 years on average.

For comparison: Bos–Kaihara–Kleinjung–Lenstra–Montgomery software uses 65 PS3 years on average.

First big speedup: We use the negation map.
Second speedup: Fast arithmetic.

Why are we confident this w

We only have 1 PlayStation-
not 200 used in their record.
Don't want to wait for 36 ye
to show that we actually co
the right thing.

Bernstein, Lange, Schwabe (PKC 2011):

Our software solves random ECDL on the same curve (with no precomputation) in 35.6 PS3 years on average.

For comparison: Bos–Kaihara–Kleinjung–Lenstra–Montgomery software uses 65 PS3 years on average.

First big speedup: We use the negation map.
Second speedup: Fast arithmetic.

Why are we confident this works?

We only have 1 PlayStation-3, not 200 used in their record. Don't want to wait for 36 years to show that we actually compute the right thing.

Bernstein, Lange, Schwabe (PKC 2011):

Our software solves random ECDL on the same curve (with no precomputation) in 35.6 PS3 years on average.

For comparison: Bos–Kaihara–Kleinjung–Lenstra–Montgomery software uses 65 PS3 years on average.

First big speedup: We use the negation map. Second speedup: Fast arithmetic.

Why are we confident this works?

We only have 1 PlayStation-3, not 200 used in their record. Don't want to wait for 36 years to show that we actually compute the right thing.

Can produced scaled versions: Use *same* prime field (so that we can compare the field arithmetic) and same curve shape $y^2 = x^3 - 3x + b$ but vary b to get curves with small subgroups.

n, Lange, Schwabe
011):

ware solves
ECDL on the same curve
precomputation)
PS3 years on average.

parison:
hara–Kleinjung–Lenstra–
mery software
PS3 years on average.

speedup:
the negation map.
speedup: Fast arithmetic.

Why are we confident this works?

We only have 1 PlayStation-3,
not 200 used in their record.
Don't want to wait for 36 years
to show that we actually compute
the right thing.

Can produced scaled versions:
Use *same* prime field
(so that we can compare the field
arithmetic) and same curve shape
$y^2 = x^3 - 3x + b$
but vary b to get curves with
small subgroups.

This pro
many of
subgrou
Specify
2^{50}, or 2
the actu
the expe
And tha

We used
point to
big attac
Need to
do not r
(artefact
We abor

Schwabe

es
the same curve
utation)
on average.

njung–Lenstra–
vare
on average.

on map.
Fast arithmetic.

Why are we confident this works?

We only have 1 PlayStation-3,
not 200 used in their record.
Don't want to wait for 36 years
to show that we actually compute
the right thing.

Can produced scaled versions:
Use *same* prime field
(so that we can compare the field
arithmetic) and same curve shape
$y^2 = x^3 - 3x + b$
but vary b to get curves with
small subgroups.

This produces oth
many of those hav
subgroups.
Specify DLP in su
2^{50}, or 2^{55}, or 2^{60}
the actual running
the expectation.
And that DLP is c

We used same pro
point to be disting
big attack; probab
Need to watch out
do not run into rh
(artefact of small
We aborted overlo

curve

e.

stra–

ge.

metic.

Why are we confident this works?

We only have 1 PlayStation-3, not 200 used in their record. Don't want to wait for 36 years to show that we actually compute the right thing.

Can produced scaled versions: Use *same* prime field (so that we can compare the field arithmetic) and same curve shape $y^2 = x^3 - 3x + b$ but vary b to get curves with small subgroups.

This produces other curves, many of those have smaller subgroups.
Specify DLP in subgroup of 2^{50}, or 2^{55}, or 2^{60} and show the actual running time mat the expectation.
And that DLP is correct.

We used same property for a point to be distinguished as big attack; probability is 2^{-2}
Need to watch out that wall do not run into rho-type cyc (artefact of small group orde
We aborted overlong walks.

Why are we confident this works?

We only have 1 PlayStation-3, not 200 used in their record. Don't want to wait for 36 years to show that we actually compute the right thing.

Can produced scaled versions: Use *same* prime field (so that we can compare the field arithmetic) and same curve shape $y^2 = x^3 - 3x + b$ but vary b to get curves with small subgroups.

This produces other curves, and many of those have smaller order subgroups.
Specify DLP in subgroup of size 2^{50}, or 2^{55}, or 2^{60} and show that the actual running time matches the expectation.
And that DLP is correct.

We used same property for a point to be distinguished as in big attack; probability is 2^{-20}. Need to watch out that walks do not run into rho-type cycles (artefact of small group order). We aborted overlong walks.

e we confident this works?

have 1 PlayStation-3,
used in their record.
ant to wait for 36 years
that we actually compute
t thing.

duced scaled versions:
e prime field
we can compare the field
tic) and same curve shape
$- 3x + b$
b to get curves with
bgroups.

This produces other curves, and many of those have smaller order subgroups.
Specify DLP in subgroup of size 2^{50}, or 2^{55}, or 2^{60} and show that the actual running time matches the expectation.
And that DLP is correct.

We used same property for a point to be distinguished as in big attack; probability is 2^{-20}.
Need to watch out that walks do not run into rho-type cycles (artefact of small group order).
We aborted overlong walks.

New rec

Announc
most wo
(`@crypt`

Elliptic c
DLP in s
Used par
DP crite

Expected
$\sqrt{\pi 2^{117.}}$
DPs, but
968 531 4
Computa
FPGAs i

dent this works?

layStation-3,
heir record.
t for 36 years
ctually compute

ed versions:
eld
ompare the field
me curve shape

curves with

This produces other curves, and many of those have smaller order subgroups.
Specify DLP in subgroup of size 2^{50}, or 2^{55}, or 2^{60} and show that the actual running time matches the expectation.
And that DLP is correct.

We used same property for a point to be distinguished as in big attack; probability is 2^{-20}.
Need to watch out that walks do not run into rho-type cycles (artefact of small group order).
We aborted overlong walks.

New record

Announced 29 No
most work by Rub
(@cryptocephaly

Elliptic curve over
DLP in subgroup o
Used parallel Polla
DP criterion: 30 t

Expected
$\sqrt{\pi 2^{117.35}/4}/2^{30}$
DPs, but ended up
968 531 433.
Computations ran
FPGAs in parallel.

works?

-3,

.

ears

mpute

s:

e field

shape

h

This produces other curves, and many of those have smaller order subgroups.
Specify DLP in subgroup of size 2^{50}, or 2^{55}, or 2^{60} and show that the actual running time matches the expectation.
And that DLP is correct.

We used same property for a point to be distinguished as in big attack; probability is 2^{-20}.
Need to watch out that walks do not run into rho-type cycles (artefact of small group order).
We aborted overlong walks.

New record

Announced 29 Nov 2016, most work by Ruben Niederl (@cryptocephaly on twitte

Elliptic curve over $\mathbf{F}_{2^{127}}$,
DLP in subgroup of order 2^1
Used parallel Pollard rho,
DP criterion: 30 top bits eq

Expected
$\sqrt{\pi 2^{117.35}/4}/2^{30} \sim 379\,821$
DPs, but ended up needing
968 531 433.
Computations ran on 64 to
FPGAs in parallel.

This produces other curves, and many of those have smaller order subgroups.
Specify DLP in subgroup of size 2^{50}, or 2^{55}, or 2^{60} and show that the actual running time matches the expectation.
And that DLP is correct.

We used same property for a point to be distinguished as in big attack; probability is 2^{-20}.
Need to watch out that walks do not run into rho-type cycles (artefact of small group order).
We aborted overlong walks.

New record

Announced 29 Nov 2016,
most work by Ruben Niederhagen
(`@cryptocephaly` on twitter).

Elliptic curve over $\mathbf{F}_{2^{127}}$,
DLP in subgroup of order $2^{117.35}$.
Used parallel Pollard rho,
DP criterion: 30 top bits equal 0.

Expected
$\sqrt{\pi 2^{117.35}/4}/2^{30} \sim 379\,821\,956$
DPs, but ended up needing
968 531 433.
Computations ran on 64 to 576
FPGAs in parallel.

oduces other curves, and
f those have smaller order
ps.
DLP in subgroup of size
2^{55}, or 2^{60} and show that
al running time matches
ectation.
t DLP is correct.

l same property for a
be distinguished as in
ck; probability is 2^{-20}.
watch out that walks
un into rho-type cycles
t of small group order).
rted overlong walks.

New record

Announced 29 Nov 2016, most work by Ruben Niederhagen (`@cryptocephaly` on twitter).

Elliptic curve over $\mathbf{F}_{2^{127}}$, DLP in subgroup of order $2^{117.35}$. Used parallel Pollard rho, DP criterion: 30 top bits equal 0.

Expected $\sqrt{\pi 2^{117.35}/4}/2^{30} \sim 379\,821\,956$ DPs, but ended up needing $968\,531\,433$. Computations ran on 64 to 576 FPGAs in parallel.

DLs in i

Want to
that DL
small int
much sm

We can
giant-ste

How to
memory-

er curves, and
ve smaller order

bgroup of size
and show that
; time matches

correct.

operty for a
guished as in
bility is 2^{-20}.
t that walks
o-type cycles
group order).
ong walks.

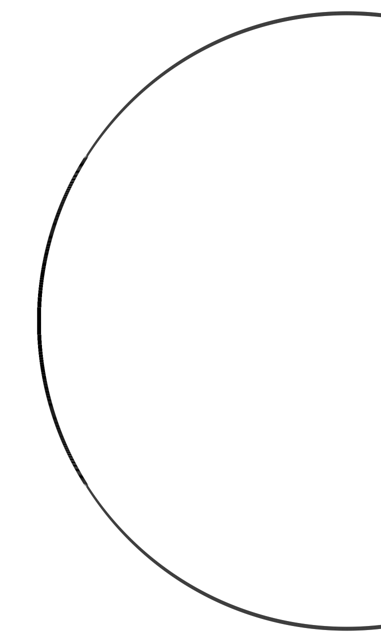
New record

Announced 29 Nov 2016,
most work by Ruben Niederhagen
(`@cryptocephaly` on twitter).

Elliptic curve over $\mathbf{F}_{2^{127}}$,
DLP in subgroup of order $2^{117.35}$.
Used parallel Pollard rho,
DP criterion: 30 top bits equal 0.

Expected
$\sqrt{\pi 2^{117.35}/4}/2^{30} \sim 379\,821\,956$
DPs, but ended up needing
968 531 433.
Computations ran on 64 to 576
FPGAs in parallel.

DLs in intervals



Want to use know
that DL is in a
small interval $[a, b$
much smaller than

We can use this in
giant-step algorith

How to use this in
memory-less algori

and
order

size
that
ches

a
in
20.
ks
cles
er).

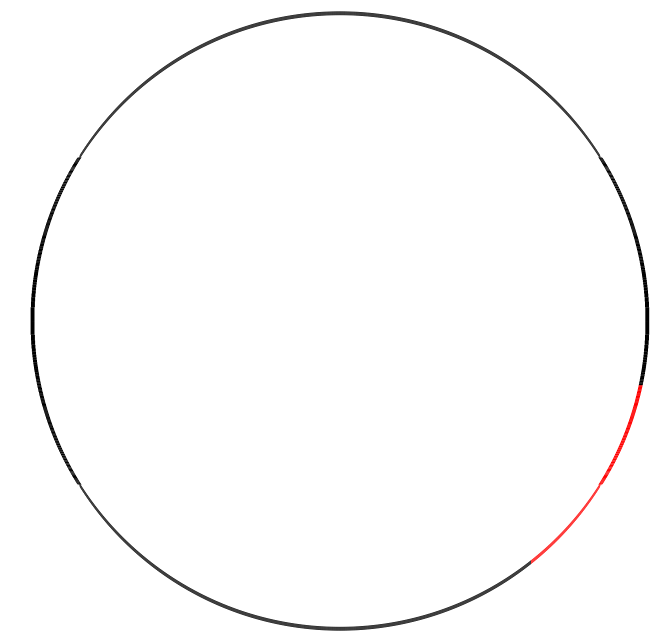
New record

Announced 29 Nov 2016,
most work by Ruben Niederhagen
(`@cryptocephaly` on twitter).

Elliptic curve over $\mathbf{F}_{2^{127}}$,
DLP in subgroup of order $2^{117.35}$.
Used parallel Pollard rho,
DP criterion: 30 top bits equal 0.

Expected
$\sqrt{\pi 2^{117.35}/4}/2^{30} \sim 379\,821\,956$
DPs, but ended up needing
968 531 433.
Computations ran on 64 to 576
FPGAs in parallel.

DLs in intervals



Want to use knowledge
that DL is in a
small interval $[a, b]$,
much smaller than ℓ.

We can use this in baby-step
giant-step algorithm.

How to use this in a
memory-less algorithm?

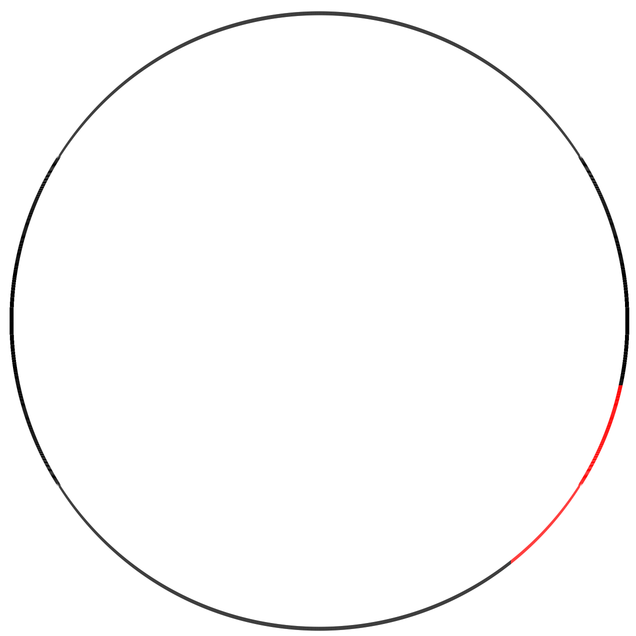
New record

Announced 29 Nov 2016,
most work by Ruben Niederhagen
(`@cryptocephaly` on twitter).

Elliptic curve over $\mathbf{F}_{2^{127}}$,
DLP in subgroup of order $2^{117.35}$.
Used parallel Pollard rho,
DP criterion: 30 top bits equal 0.

Expected
$\sqrt{\pi 2^{117.35}/4}/2^{30} \sim 379\,821\,956$
DPs, but ended up needing
968 531 433.
Computations ran on 64 to 576
FPGAs in parallel.

DLs in intervals



Want to use knowledge
that DL is in a
small interval $[a, b]$,
much smaller than ℓ.

We can use this in baby-step
giant-step algorithm.

How to use this in a
memory-less algorithm?

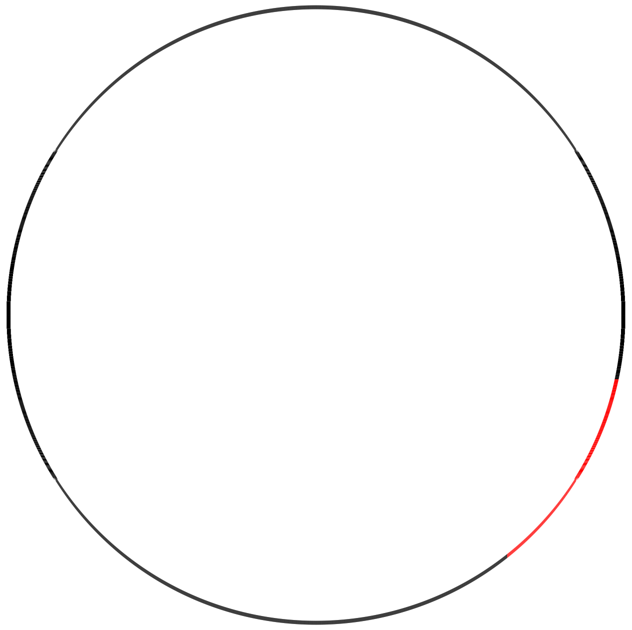
ord

ced 29 Nov 2016,
rk by Ruben Niederhagen
ocephaly on twitter).

curve over $\mathbf{F}_{2^{127}}$,
subgroup of order $2^{117.35}$.
rallel Pollard rho,
erion: 30 top bits equal 0.

d
$.35/4/2^{30} \sim 379\,821\,956$
t ended up needing
433.
ations ran on 64 to 576
n parallel.

DLs in intervals



Want to use knowledge
that DL is in a
small interval $[a, b]$,
much smaller than ℓ.

We can use this in baby-step
giant-step algorithm.

How to use this in a
memory-less algorithm?

Standard
Pollard's

Pollard's
jumps ar

v 2016,

en Niederhagen

on twitter).

$\mathbf{F}_{2^{127}}$,

of order $2^{117.35}$.

rd rho,

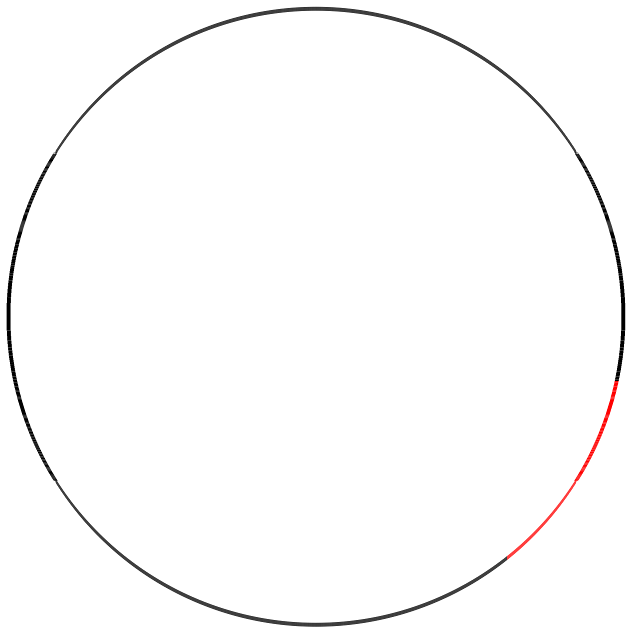
op bits equal 0.

$\sim 379\,821\,956$

o needing

on 64 to 576

DLs in intervals



Want to use knowledge that DL is in a small interval $[a, b]$, much smaller than ℓ.

We can use this in baby-step giant-step algorithm.

How to use this in a memory-less algorithm?

Standard interval

Pollard's kangaroo

Pollard's kangaroo

jumps around the

hagen
er).

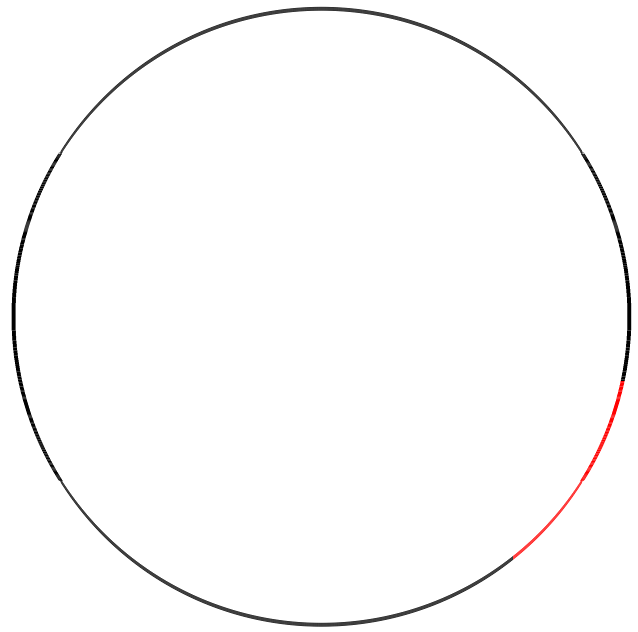
117.35 .

ual 0.

956

576

DLs in intervals



Want to use knowledge that DL is in a small interval $[a, b]$, much smaller than ℓ.

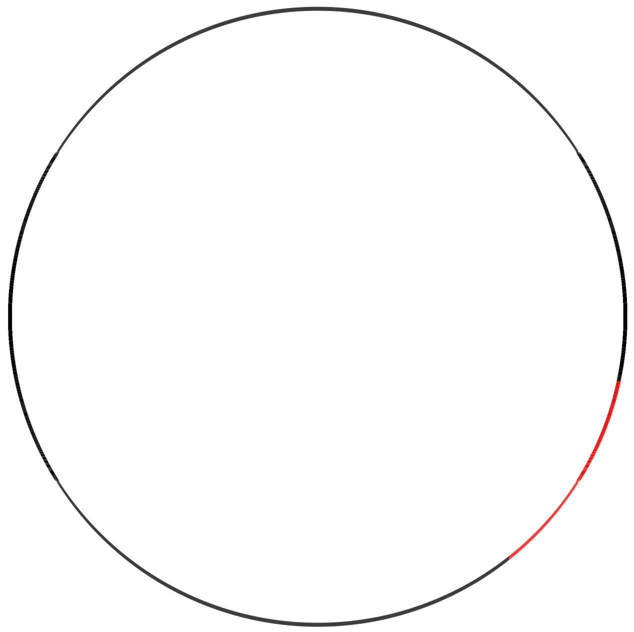
We can use this in baby-step giant-step algorithm.

How to use this in a memory-less algorithm?

Standard interval method: Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

DLs in intervals



Want to use knowledge that DL is in a small interval $[a, b]$, much smaller than ℓ.

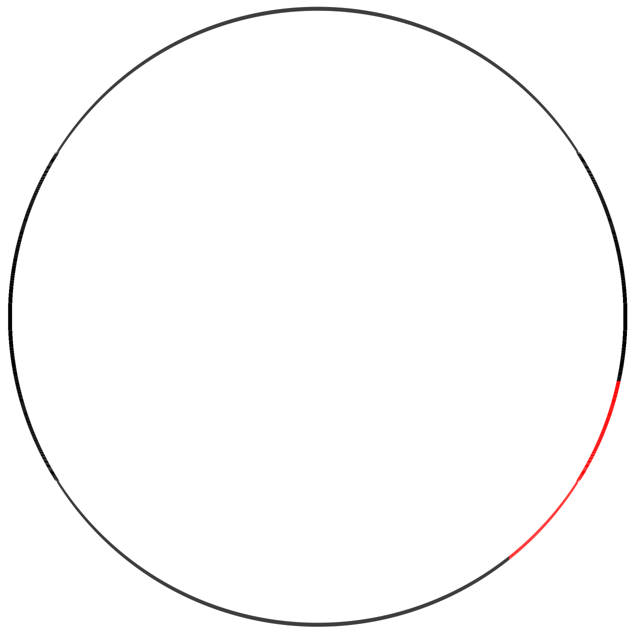
We can use this in baby-step giant-step algorithm.

How to use this in a memory-less algorithm?

Standard interval method: Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

DLs in intervals



Want to use knowledge that DL is in a small interval $[a, b]$, much smaller than ℓ.

We can use this in baby-step giant-step algorithm.

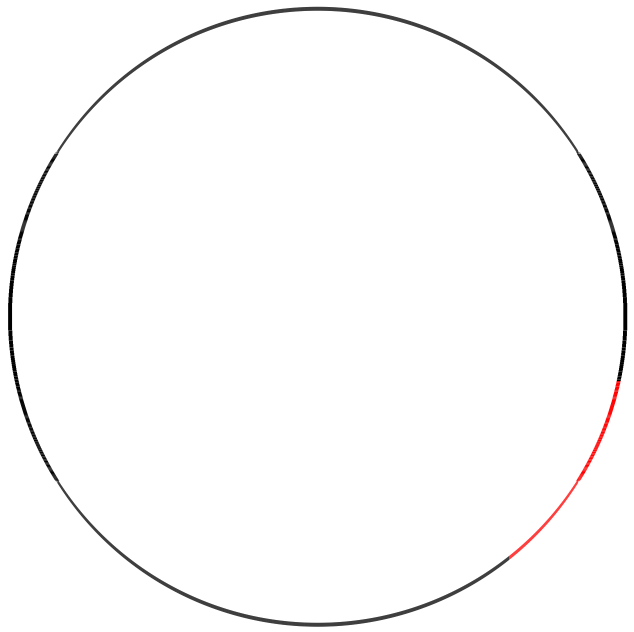
How to use this in a memory-less algorithm?

Standard interval method: Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Real kangaroos sleep

DLs in intervals



Want to use knowledge that DL is in a small interval $[a, b]$, much smaller than ℓ.

We can use this in baby-step giant-step algorithm.

How to use this in a memory-less algorithm?

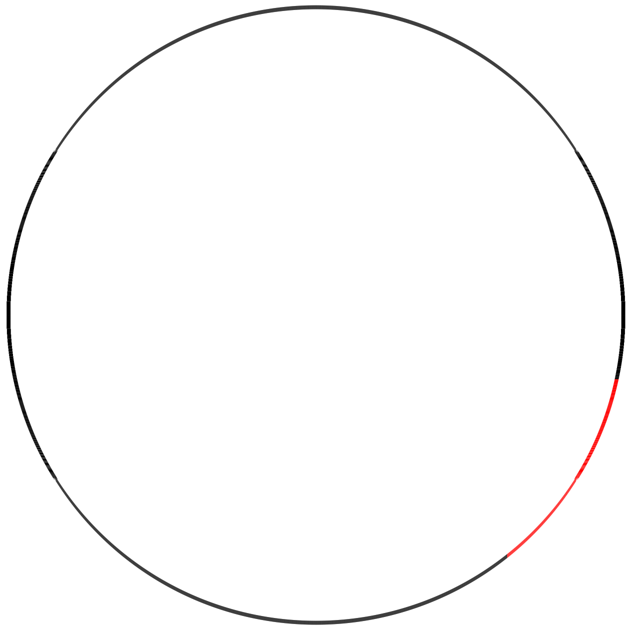
Standard interval method: Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Real kangaroos sleep

(at least outside Australia).

ntervals



use knowledge
is in a
terval $[a, b]$,
naller than ℓ.

use this in baby-step
ep algorithm.

use this in a
-less algorithm?

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small
jumps around the interval.

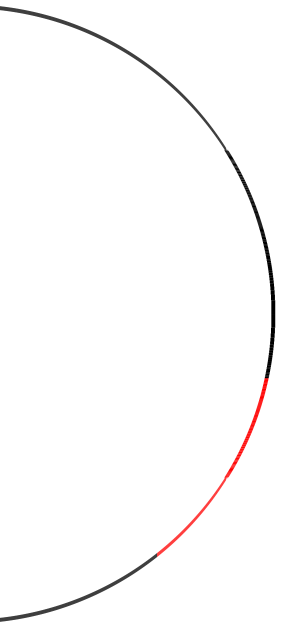
Real kangaroos sleep

(at least outside Australia).

Kangaro

in Austr

Main ac



ledge

],

ℓ.

baby-step

m.

a

ithm?

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small
jumps around the interval.

Real kangaroos sleep

(at least outside Australia).

Kangaroo method

in Australia

Main actor:

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Real kangaroos sleep

(at least outside Australia).

Kangaroo method

in Australia

Main actor:

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Real kangaroos sleep

(at least outside Australia).

Kangaroo method

in Australia

Main actor:

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Real kangaroos sleep

(at least outside Australia).

The tame kangaroo

starts at a known multiple of P, e.g. bP.

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Real kangaroos sleep

(at least outside Australia).

The tame kangaroo jumps.

Jumps are determined by current position.

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Real kangaroos sleep

(at least outside Australia).

The tame kangaroo jumps.

Jumps are determined by current position.
Average jump distance is $\sqrt{b-a}$.

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Real kangaroos sleep

(at least outside Australia).

The tame kangaroo jumps.

Jumps are determined by current position.
Average jump distance is $\sqrt{b-a}$.

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Real kangaroos sleep

(at least outside Australia).

The tame kangaroo jumps.

Jumps are determined by current position.
Average jump distance is $\sqrt{b-a}$.

d interval method:
s kangaroo method.

s kangaroos do small
round the interval.

ngaroos sleep

outside Australia).

The tame kangaroo jumps.

Jumps are determined
by current position.
Average jump distance
is $\sqrt{b-a}$.

The tam

after a f
(about √

The tam
and wait

method:

method.

s do small

interval.

eep

ustralia).

The tame kangaroo jumps.

Jumps are determined
by current position.
Average jump distance
is $\sqrt{b-a}$.

The tame kangaro

after a fixed numb
(about $\sqrt{b-a}$ ma

The tame kangaro
and waits.

The tame kangaroo jumps.

Jumps are determined by current position.

Average jump distance is $\sqrt{b-a}$.

The tame kangaroo stops

after a fixed number of jum (about $\sqrt{b-a}$ many).

The tame kangaroo installs and waits.

The tame kangaroo jumps.

Jumps are determined by current position. Average jump distance is $\sqrt{b-a}$.

The tame kangaroo stops

after a fixed number of jumps (about $\sqrt{b-a}$ many).

The tame kangaroo installs a trap and waits.

he kangaroo jumps.

re determined
nt position.
jump distance
a.

The tame kangaroo stops

after a fixed number of jumps (about $\sqrt{b-a}$ many).

The tame kangaroo installs a trap and waits.

The wil

starts at
Follows
jumps.

o jumps.

ined

n.

ance

The tame kangaroo stops

after a fixed number of jumps (about $\sqrt{b-a}$ many).

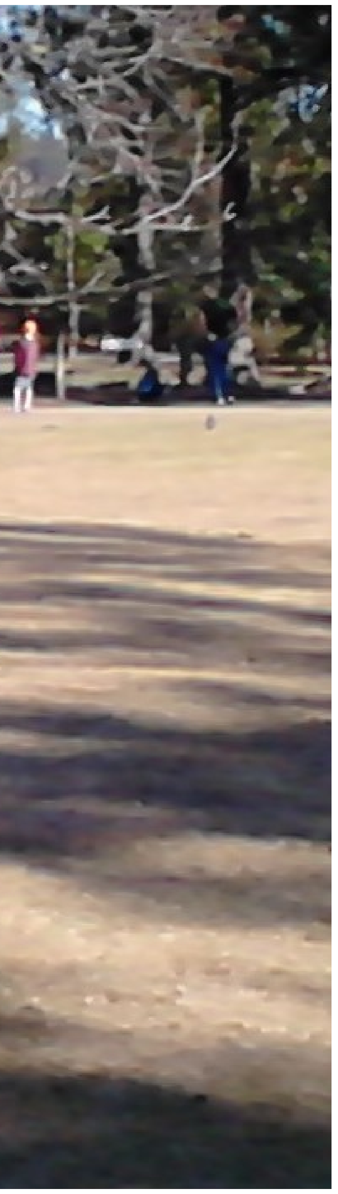
The tame kangaroo installs a trap and waits.

The wild kangaroo

starts at point Q.

Follows the same i

jumps.



The tame kangaroo stops

after a fixed number of jumps (about $\sqrt{b-a}$ many).

The tame kangaroo installs a trap and waits.

The wild kangaroo

starts at point Q.

Follows the same instructio
jumps.

The tame kangaroo stops

after a fixed number of jumps (about $\sqrt{b-a}$ many).

The tame kangaroo installs a trap and waits.

The wild kangaroo

starts at point Q.

Follows the same instructions for jumps.

he kangaroo stops

ixed number of jumps
$\sqrt{b-a}$ many).

he kangaroo installs a trap
ts.

The wild kangaroo

starts at point Q.
Follows the same instructions for jumps.

But we c
the start
Know Q

Hope th
and wild

Similar t
kangaroo
path fro

Eventua
into the
(Or disa
paths ha
Start a f
from Q -

o stops

er of jumps
any).

o installs a trap

The wild kangaroo

starts at point Q.
Follows the same instructions for jumps.

But we don't know
the starting point
Know $Q = nP$ wit

Hope that the pat
and wild kangaroo

Similar to the rho
kangaroos will hop
path from that po

Eventually the wil
into the trap.
(Or disappears in
paths have not int
Start a fresh one
from $Q + P, Q + 2$

ps

a trap

The wild kangaroo

starts at point Q.
Follows the same instructions for jumps.

But we don't know where the starting point Q is.
Know $Q = nP$ with $n \in [a,$

Hope that the paths of the t
and wild kangaroo intersect.

Similar to the rho method th
kangaroos will hop on the sa
path from that point onwarc

Eventually the wild kangaroo
into the trap.
(Or disappears in the distanc
paths have not intersected.
Start a fresh one
from $Q + P, Q + 2P, \ldots$.)

The wild kangaroo

starts at point Q.
Follows the same instructions for jumps.

But we don't know where the starting point Q is.
Know $Q = nP$ with $n \in [a, b]$.

Hope that the paths of the tame and wild kangaroo intersect.

Similar to the rho method the kangaroos will hop on the same path from that point onwards.

Eventually the wild kangaroo falls into the trap.
(Or disappears in the distance if paths have not intersected.
Start a fresh one from $Q + P, Q + 2P, \ldots$.)

d kangaroo

point Q.
the same instructions for

But we don't know where the starting point Q is.
Know $Q = nP$ with $n \in [a, b]$.

Hope that the paths of the tame and wild kangaroo intersect.

Similar to the rho method the kangaroos will hop on the same path from that point onwards.

Eventually the wild kangaroo falls into the trap.
(Or disappears in the distance if paths have not intersected.
Start a fresh one from $Q + P, Q + 2P, \dots$.)

Same st

Kangaro
Starting
Distance
Step set
with s_i
$s = \beta\sqrt{}$
Hash fu
$H : \langle P \rangle$
Update
$d_{i+1} =$
$X_{i+1} =$

o

instructions for

But we don't know where the starting point Q is.
Know $Q = nP$ with $n \in [a, b]$.

Hope that the paths of the tame and wild kangaroo intersect.

Similar to the rho method the kangaroos will hop on the same path from that point onwards.

Eventually the wild kangaroo falls into the trap.
(Or disappears in the distance if paths have not intersected.
Start a fresh one
from $Q + P, Q + 2P, \ldots.$)

Same story in mat

Kangaroo = seque
Starting point X_0
Distance $d_0 = 0$.
Step set: $S = \{s_1$
with s_i on average
$s = \beta\sqrt{b-a}$.
Hash function
$H : \langle P \rangle \to \{1, 2, \ldots$
Update function
$d_{i+1} = d_i + s_{H(X_i}$
$X_{i+1} = X_i + s_{H(X}$

ns for

But we don't know where the starting point Q is.
Know $Q = nP$ with $n \in [a, b]$.

Hope that the paths of the tame and wild kangaroo intersect.

Similar to the rho method the kangaroos will hop on the same path from that point onwards.

Eventually the wild kangaroo falls into the trap.
(Or disappears in the distance if paths have not intersected.
Start a fresh one
from $Q + P, Q + 2P, \ldots$.)

Same story in math

Kangaroo = sequence $X_i \in$
Starting point $X_0 = s_0 P$.
Distance $d_0 = 0$.
Step set: $S = \{s_1 P, \ldots, s_L$
with s_i on average
$s = \beta\sqrt{b - a}$.
Hash function
$H : \langle P \rangle \to \{1, 2, \ldots, L\}$.
Update function
$d_{i+1} = d_i + s_{H(X_i)}, \quad i = 0$
$X_{i+1} = X_i + s_{H(X_i)} P, \quad i =$

But we don't know where the starting point Q is.
Know $Q = nP$ with $n \in [a, b]$.

Hope that the paths of the tame and wild kangaroo intersect.

Similar to the rho method the kangaroos will hop on the same path from that point onwards.

Eventually the wild kangaroo falls into the trap.
(Or disappears in the distance if paths have not intersected.
Start a fresh one from $Q+P, Q+2P, \ldots$.)

<u>Same story in math</u>

Kangaroo = sequence $X_i \in \langle P \rangle$.
Starting point $X_0 = s_0 P$.
Distance $d_0 = 0$.
Step set: $S = \{s_1 P, \ldots, s_L P\}$,
with s_i on average
$s = \beta\sqrt{b-a}$.
Hash function
$H : \langle P \rangle \to \{1, 2, \ldots, L\}$.
Update function
$d_{i+1} = d_i + s_{H(X_i)}, \quad i = 0, 1, 2, \ldots,$
$X_{i+1} = X_i + s_{H(X_i)} P, \quad i = 0, 1, 2, \ldots$

don't know where
ting point Q is.
$= nP$ with $n \in [a, b]$.

at the paths of the tame
l kangaroo intersect.

to the rho method the
os will hop on the same
m that point onwards.

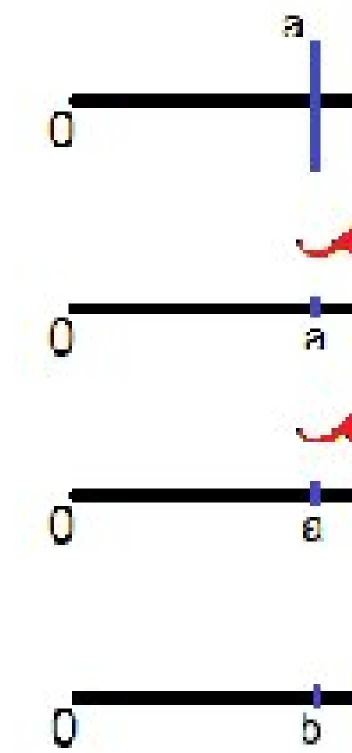
lly the wild kangaroo falls
trap.
ppears in the distance if
ave not intersected.
fresh one
$+ P, Q + 2P, \ldots.)$

Same story in math

Kangaroo = sequence $X_i \in \langle P \rangle$.
Starting point $X_0 = s_0 P$.
Distance $d_0 = 0$.
Step set: $S = \{s_1 P, \ldots, s_L P\}$,
with s_i on average
$s = \beta\sqrt{b-a}$.
Hash function
$H : \langle P \rangle \rightarrow \{1, 2, \ldots, L\}$.
Update function

$$d_{i+1} = d_i + s_{H(X_i)}, \quad i = 0, 1, 2, \ldots,$$
$$X_{i+1} = X_i + s_{H(X_i)} P, \quad i = 0, 1, 2, \ldots$$

Tame ka
$X_0 = bF$
wild kan
$X'_0 = Q$
Trap: di
endpoint



Picture
Christine

w where
Q is.
th $n \in [a, b]$.

hs of the tame
intersect.

method the
on the same
int onwards.

d kangaroo falls

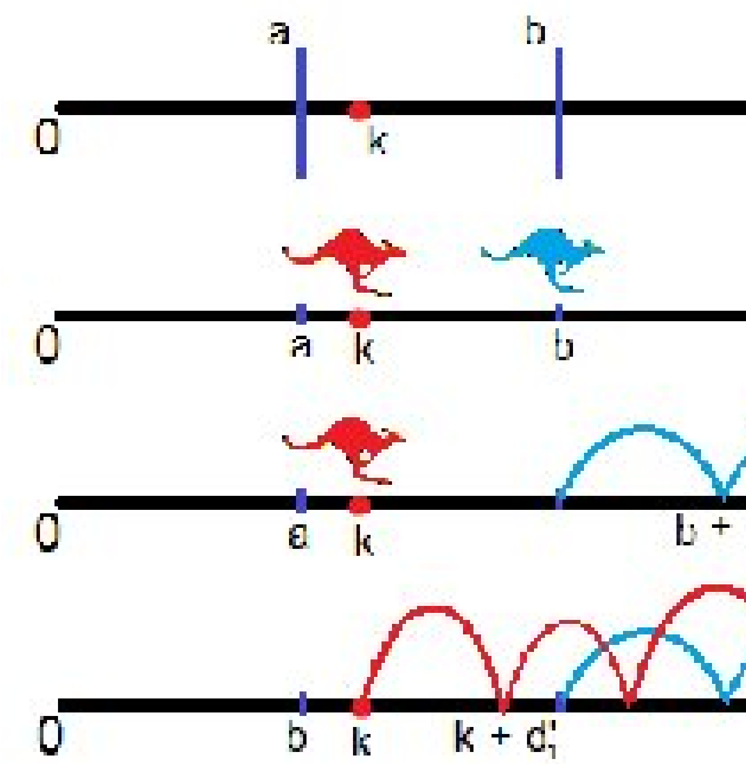
the distance if
ersected.

$2P, \ldots .)$

Same story in math

Kangaroo = sequence $X_i \in \langle P \rangle$.
Starting point $X_0 = s_0 P$.
Distance $d_0 = 0$.
Step set: $S = \{s_1 P, \ldots, s_L P\}$,
with s_i on average
$s = \beta\sqrt{b-a}$.
Hash function
$H : \langle P \rangle \rightarrow \{1, 2, \ldots, L\}$.
Update function
$d_{i+1} = d_i + s_{H(X_i)}, \quad i = 0, 1, 2, \ldots,$
$X_{i+1} = X_i + s_{H(X_i)} P, \quad i = 0, 1, 2, \ldots$

Tame kangaroo st
$X_0 = bP$,
wild kangaroo star
$X'_0 = Q = nP$.
Trap: distance d_N
endpoint $X_N = (b$



Picture credit:
Christine van Vred

$b]$.

tame

he

ame

ds.

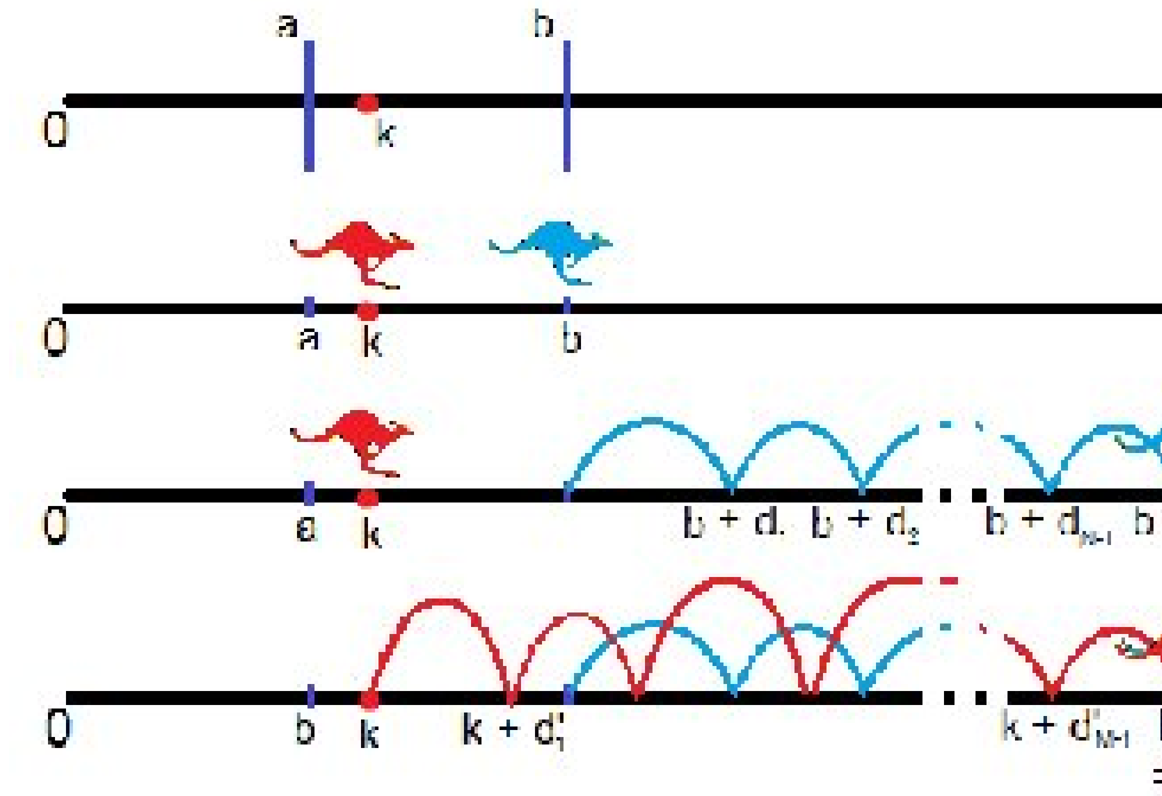
o falls

ce if

Same story in math

Kangaroo = sequence $X_i \in \langle P \rangle$.
Starting point $X_0 = s_0 P$.
Distance $d_0 = 0$.
Step set: $S = \{s_1 P, \ldots, s_L P\}$,
with s_i on average
$s = \beta\sqrt{b - a}$.
Hash function
$H : \langle P \rangle \to \{1, 2, \ldots, L\}$.
Update function
$d_{i+1} = d_i + s_{H(X_i)}, \quad i = 0, 1, 2, \ldots,$
$X_{i+1} = X_i + s_{H(X_i)} P, \quad i = 0, 1, 2, \ldots$

Tame kangaroo starts at
$X_0 = bP$,
wild kangaroo starts at
$X_0' = Q = nP$.
Trap: distance d_N,
endpoint $X_N = (b + d_N)P$.



Picture credit:
Christine van Vredendaal.

Same story in math

Kangaroo = sequence $X_i \in \langle P \rangle$.

Starting point $X_0 = s_0 P$.

Distance $d_0 = 0$.

Step set: $S = \{s_1 P, \ldots, s_L P\}$,

with s_i on average

$s = \beta\sqrt{b - a}$.

Hash function

$H : \langle P \rangle \to \{1, 2, \ldots, L\}$.

Update function

$d_{i+1} = d_i + s_{H(X_i)}, \quad i = 0, 1, 2, \ldots,$

$X_{i+1} = X_i + s_{H(X_i)} P, \quad i = 0, 1, 2, \ldots$

Tame kangaroo starts at

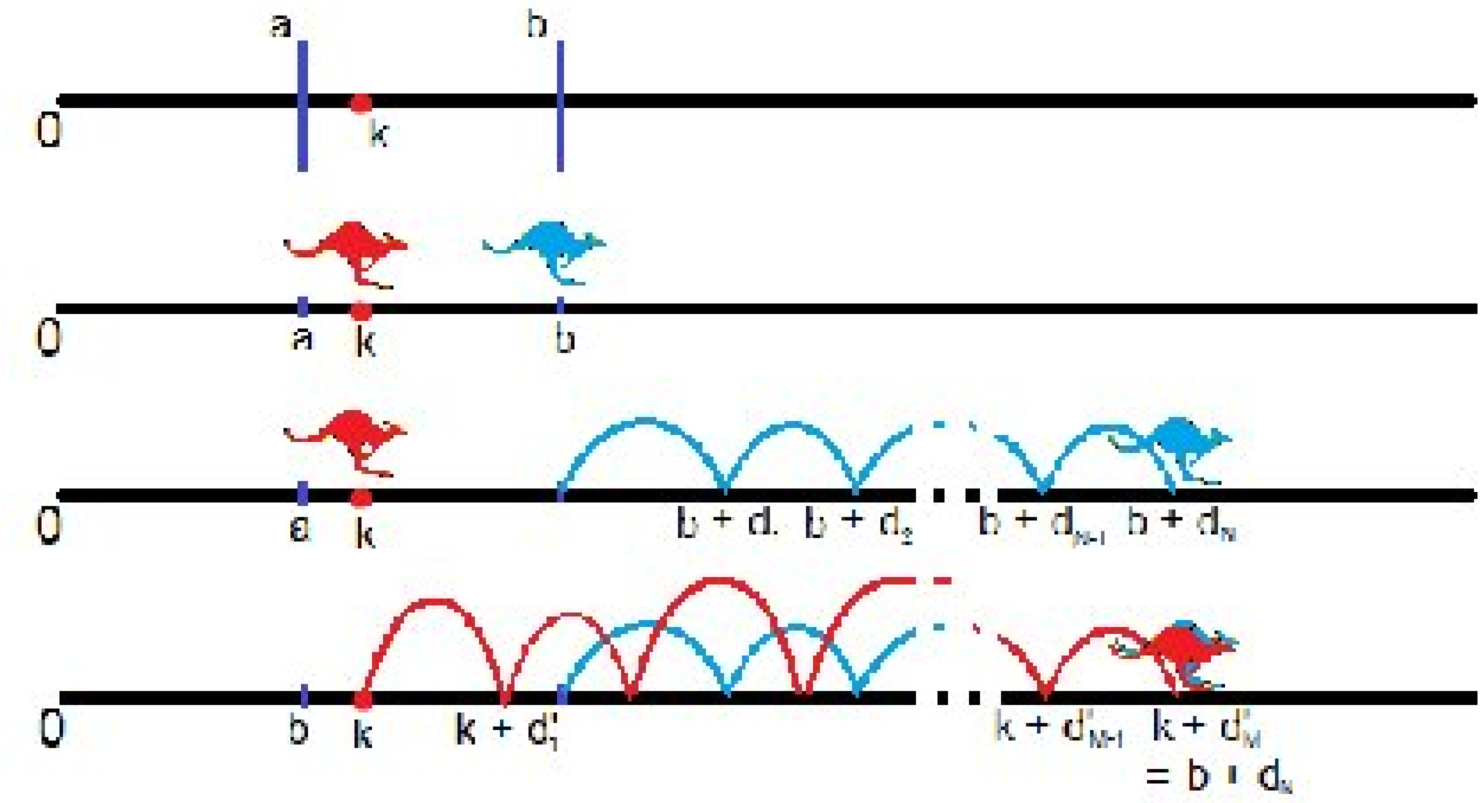
$X_0 = bP$,

wild kangaroo starts at

$X'_0 = Q = nP$.

Trap: distance d_N,

endpoint $X_N = (b + d_N)P$.



Picture credit:

Christine van Vredendaal.

ory in math

oo = sequence $X_i \in \langle P \rangle$.

point $X_0 = s_0 P$.

e $d_0 = 0$.

: $S = \{s_1 P, \ldots, s_L P\}$,

on average

$\overline{b - a}$.

nction

$\to \{1, 2, \ldots, L\}$.

function

$d_i + s_{H(X_i)}, \quad i = 0, 1, 2, \ldots,$

$X_i + s_{H(X_i)} P, \quad i = 0, 1, 2, \ldots$

Tame kangaroo starts at

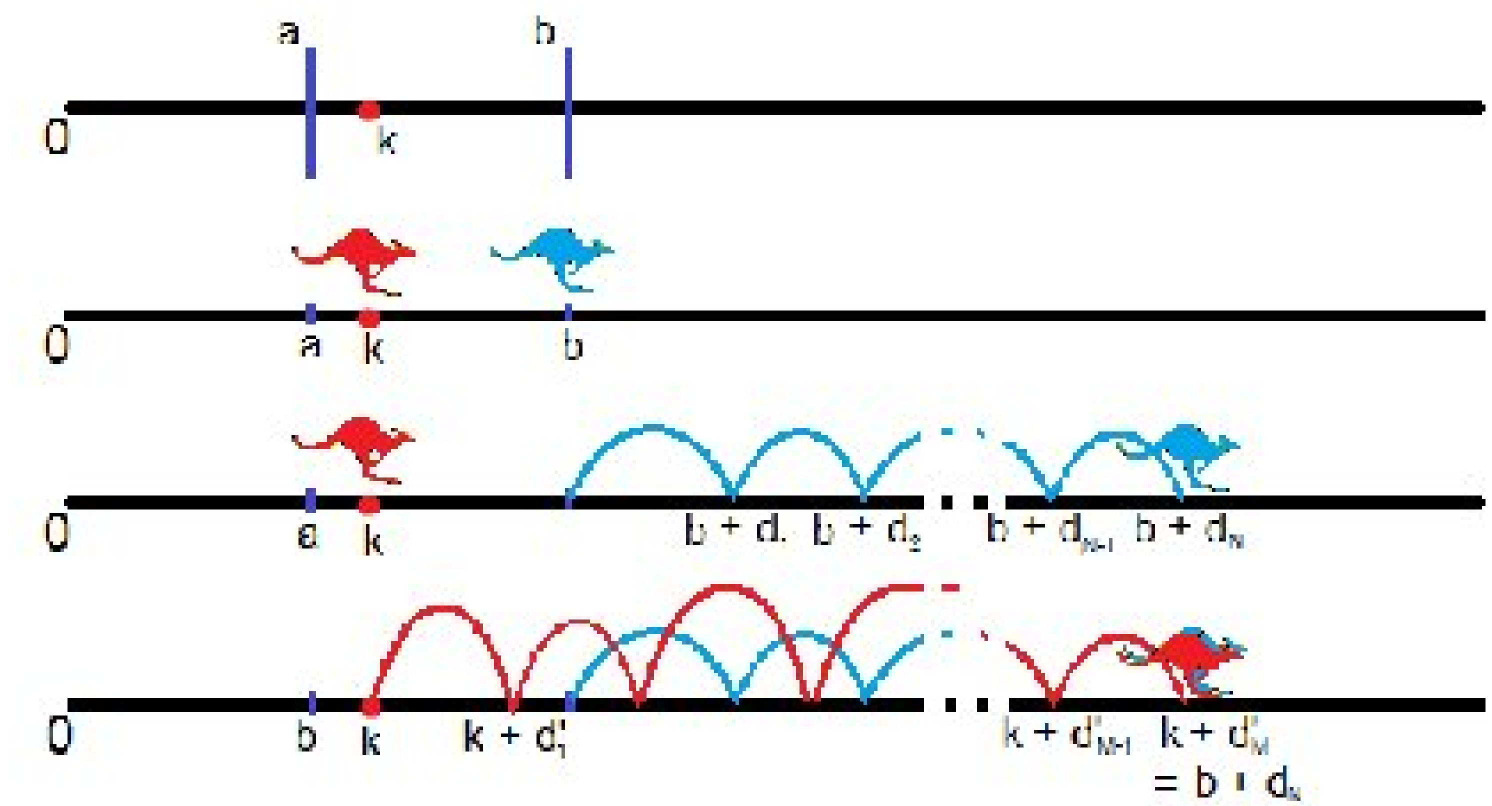
$X_0 = bP$,

wild kangaroo starts at

$X'_0 = Q = nP$.

Trap: distance d_N,

endpoint $X_N = (b + d_N)P$.

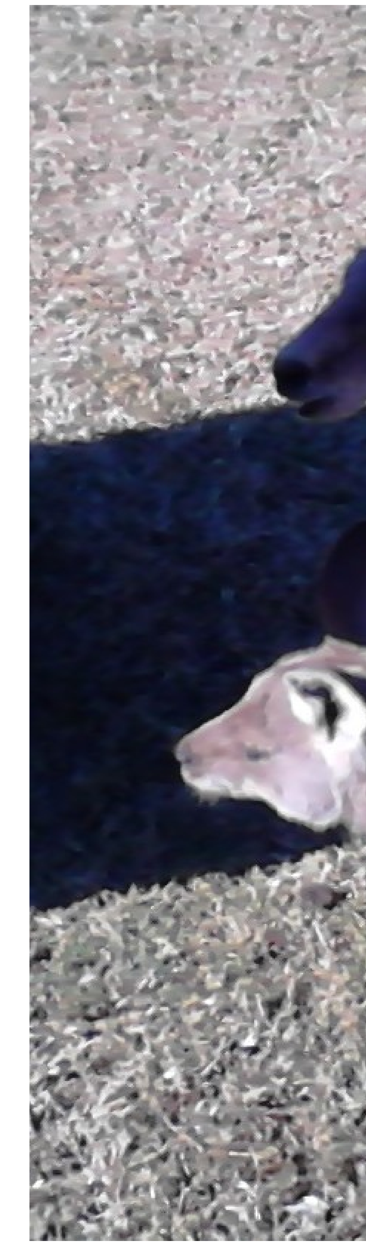


Picture credit:

Christine van Vredendaal.

Parallel

Use an e



of tame

all starti

around (

th

ence $X_i \in \langle P \rangle$.

$= s_0 P$.

$P, \dots, s_L P\}$,

e

$\dots, L\}$.

$)$, $\quad i = 0, 1, 2, \dots,$

$_i)P$, $\quad i = 0, 1, 2, \dots$

Tame kangaroo starts at

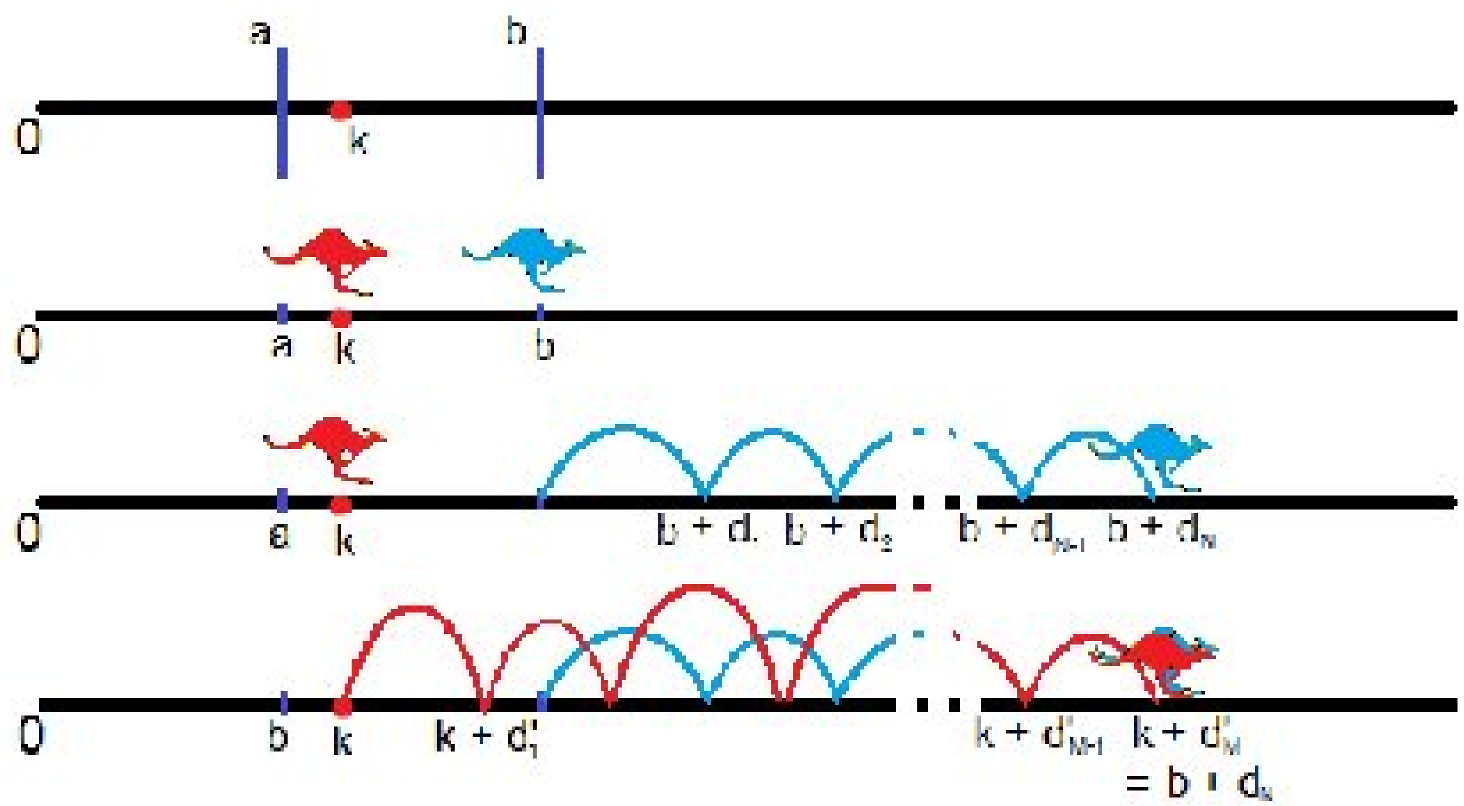
$X_0 = bP$,

wild kangaroo starts at

$X'_0 = Q = nP$.

Trap: distance d_N,

endpoint $X_N = (b + d_N)P$.



Picture credit:

Christine van Vredendaal.

Parallel kangaroo

Use an entire herd

of tame kangaroos

all starting

around $((b - a)/2$

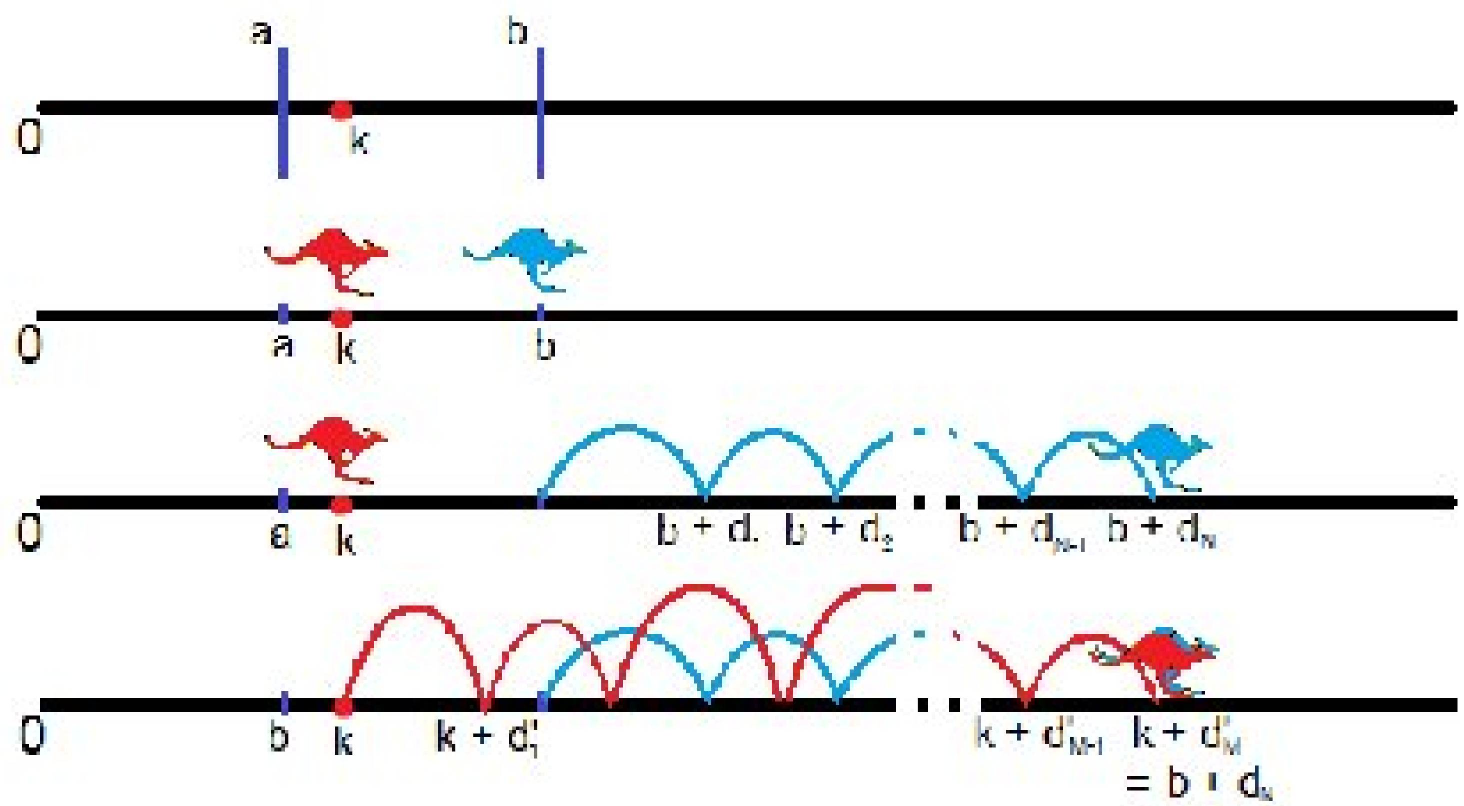
$\langle P \rangle$.

$P\}$,

$, 1, 2, \ldots,$

$0, 1, 2, \ldots$

Tame kangaroo starts at
$X_0 = bP$,
wild kangaroo starts at
$X'_0 = Q = nP$.
Trap: distance d_N,
endpoint $X_N = (b + d_N)P$.



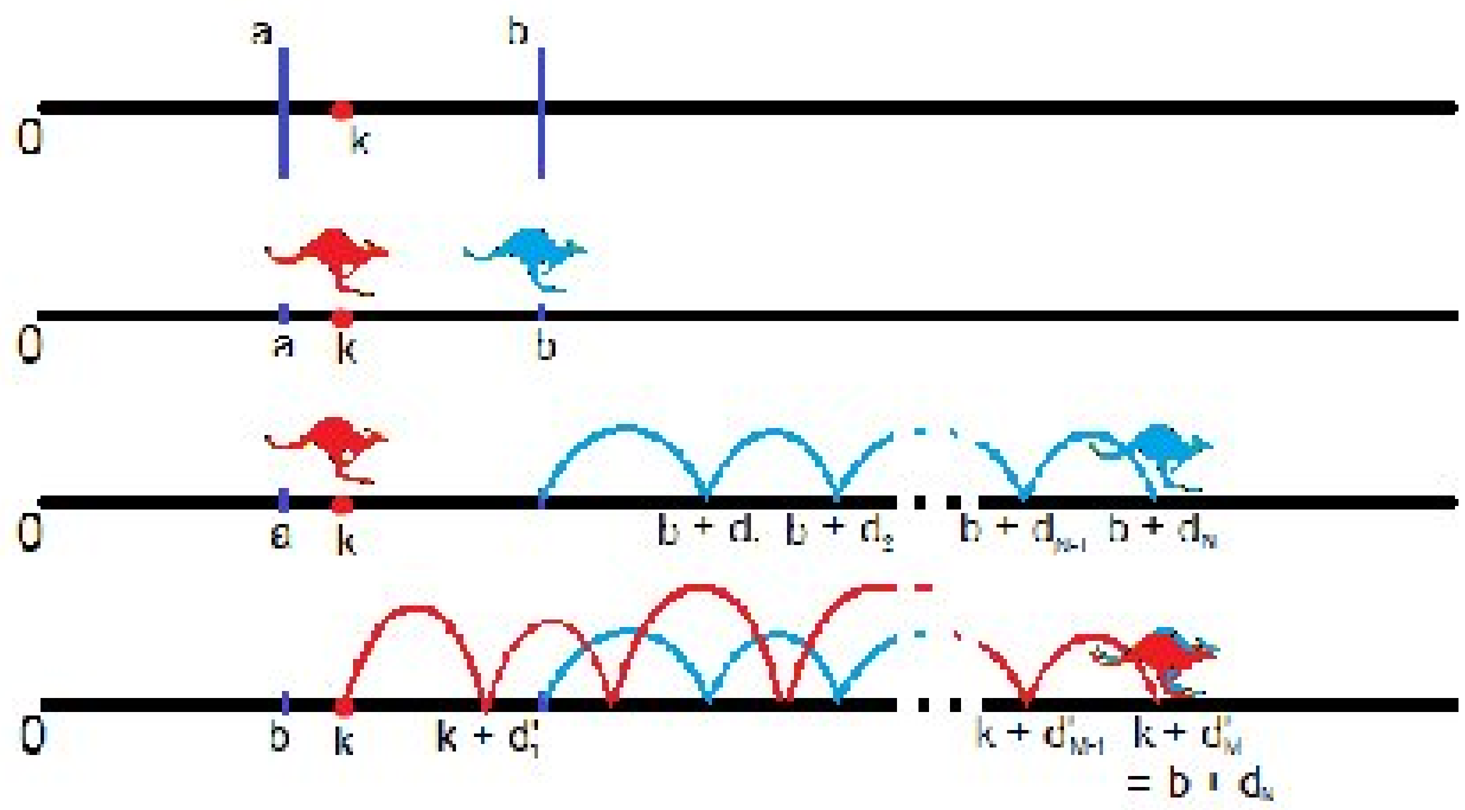
Picture credit:
Christine van Vredendaal.

Parallel kangaroo method

Use an entire herd

of tame kangaroos,
all starting
around $((b - a)/2)P$ . . .

Tame kangaroo starts at
$X_0 = bP$,
wild kangaroo starts at
$X'_0 = Q = nP$.
Trap: distance d_N,
endpoint $X_N = (b + d_N)P$.



Picture credit:
Christine van Vredendaal.

Parallel kangaroo method

Use an entire herd

of tame kangaroos,
all starting
around $((b - a)/2)P$ ...

angaroo starts at

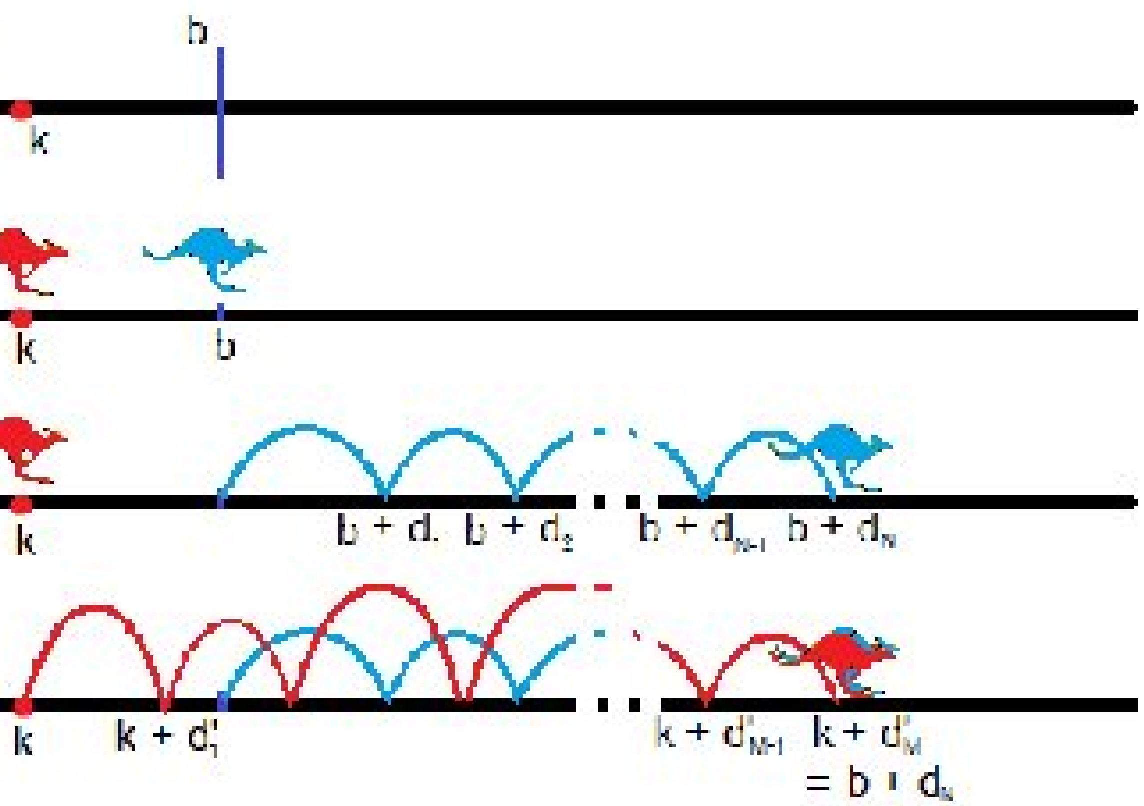
P,

garoo starts at

$= nP$.

stance d_N,

t $X_N = (b + d_N)P$.



credit:

e van Vredendaal.

Parallel kangaroo method

Use an entire herd

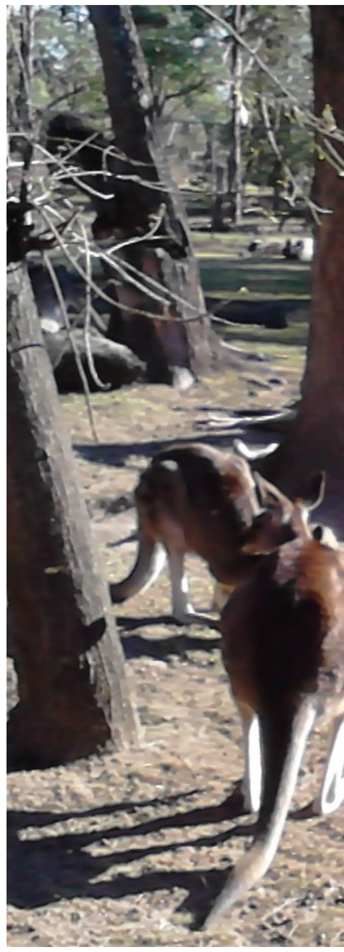
of tame kangaroos,

all starting

around $((b - a)/2)P$ ...

... and c

distingui



Also sta

wild kan

Hope th

one tam

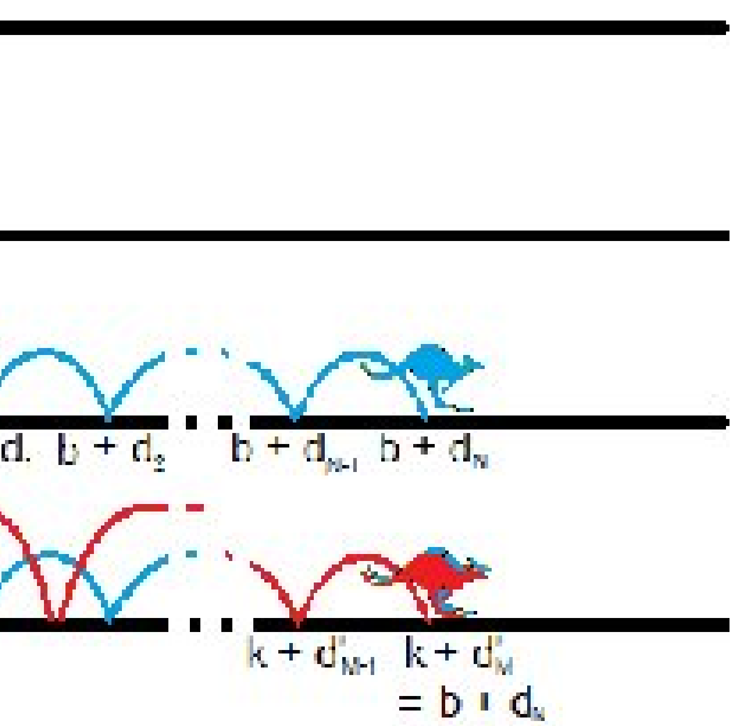
meet at

arts at

rts at

,

$+ d_N)P$.



dendaal.

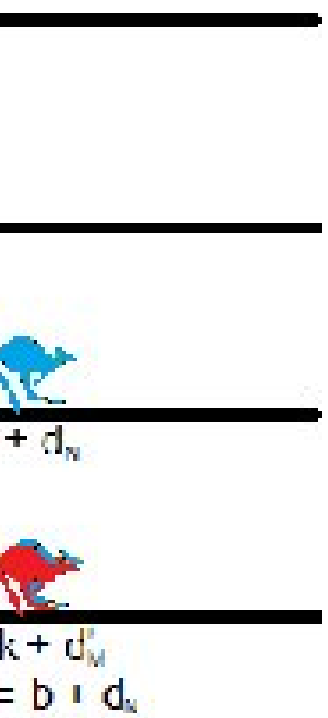
Parallel kangaroo method

Use an entire herd

of tame kangaroos,
all starting
around $((b - a)/2)P$ ...

...and define cert
distinguished poin

Also start a herd o
wild kangaroos aro
Hope that one wil
one tame kangaroo
meet at one distin



Parallel kangaroo method

Use an entire herd

of tame kangaroos,
all starting
around $((b-a)/2)P$ ...

... and define certain spots a
distinguished points

Also start a herd of
wild kangaroos around Q.
Hope that one wild and
one tame kangaroo
meet at one distinguished po

Parallel kangaroo method

Use an entire herd

of tame kangaroos,
all starting
around $((b-a)/2)P$ ...

...and define certain spots as distinguished points

Also start a herd of
wild kangaroos around Q.
Hope that one wild and
one tame kangaroo
meet at one distinguished point.

kangaroo method

entire herd

kangaroos,
ng
$((b - a)/2)P$ . . .

. . . and define certain spots as distinguished points

Also start a herd of
wild kangaroos around Q.
Hope that one wild and
one tame kangaroo
meet at one distinguished point.

Pairings

Let $(G_1,$
be group
$e : G_1 \times$
be a ma
$e(P + Q$
$e(P, R' +$

Request
non-deg
argumen
$e(P, R')$
then P i

Such an
or *pairin*

method

l

s,

$)P \ldots$

…and define certain spots as distinguished points

Also start a herd of
wild kangaroos around Q.
Hope that one wild and
one tame kangaroo
meet at one distinguished point.

Pairings

Let $(G_1, +), (G_2, +$
be groups of prime
$e : G_1 \times G_2 \rightarrow G_T$
be a map satisfyin
$e(P + Q, R') = e($
$e(P, R' + S') = e($

Request further th
non-degenerate in
argument, i.e., if f
$e(P, R') = 1$ for al
then P is the iden

Such an e is called
or *pairing*.

. . . and define certain spots as distinguished points

Also start a herd of wild kangaroos around Q. Hope that one wild and one tame kangaroo meet at one distinguished point.

Pairings

Let $(G_1, +), (G_2, +)$ and $(G$
be groups of prime order ℓ a
$e : G_1 \times G_2 \to G_T$
be a map satisfying
$e(P + Q, R') = e(P, R')e(Q$
$e(P, R' + S') = e(P, R')e(P$

Request further that e is non-degenerate in the first argument, i.e., if for some P
$e(P, R') = 1$ for all $R' \in G_2$,
then P is the identity in G_1

Such an e is called a *bilinea* or *pairing*.

. . . and define certain spots as distinguished points

Also start a herd of
wild kangaroos around Q.
Hope that one wild and
one tame kangaroo
meet at one distinguished point.

Pairings

Let $(G_1, +), (G_2, +)$ and $(G_T, \cdot)$ be groups of prime order ℓ and let
$e : G_1 \times G_2 \to G_T$
be a map satisfying
$e(P + Q, R') = e(P, R')e(Q, R')$,
$e(P, R' + S') = e(P, R')e(P, S')$.

Request further that e is non-degenerate in the first argument, i.e., if for some P $e(P, R') = 1$ for all $R' \in G_2$, then P is the identity in G_1

Such an e is called a *bilinear map* or *pairing*.

define certain spots as
ished points

rt a herd of
garoos around Q.
at one wild and
e kangaroo
one distinguished point.

Pairings

Let $(G_1, +), (G_2, +)$ and $(G_T, \cdot)$ be groups of prime order ℓ and let
$e : G_1 \times G_2 \to G_T$
be a map satisfying

$$e(P + Q, R') = e(P, R')e(Q, R'),$$
$$e(P, R' + S') = e(P, R')e(P, S').$$

Request further that e is non-degenerate in the first argument, i.e., if for some P $e(P, R') = 1$ for all $R' \in G_2$, then P is the identity in G_1

Such an e is called a *bilinear map* or *pairing*.

Consequ

Assume
in partic

Then for
$(P_1, P_2,$
one can
in $\log \ell$
$\log_P(P_3$
by comp
$e(P_1, P_2$
This me
Diffie-He

ain spots as

ts

of

ound Q.

d and

o

guished point.

Pairings

Let $(G_1, +), (G_2, +)$ and $(G_T, \cdot)$ be groups of prime order ℓ and let
$e : G_1 \times G_2 \to G_T$
be a map satisfying
$e(P + Q, R') = e(P, R')e(Q, R')$,
$e(P, R' + S') = e(P, R')e(P, S')$.

Request further that e is non-degenerate in the first argument, i.e., if for some P $e(P, R') = 1$ for all $R' \in G_2$, then P is the identity in G_1

Such an e is called a *bilinear map* or *pairing*.

Consequences of p

Assume that $G_1 =$

in particular $e(P, P$

Then for all triples

$(P_1, P_2, P_3) \in \langle P \rangle$

one can decide in

in $\log \ell$ whether

$\log_P(P_3) = \log_P($

by comparing

$e(P_1, P_2)$ and $e(P$

This means that t

Diffie-Hellman pro

as

oint.

Pairings

Let $(G_1, +), (G_2, +)$ and $(G_T, \cdot)$ be groups of prime order ℓ and let
$e : G_1 \times G_2 \to G_T$
be a map satisfying
$e(P + Q, R') = e(P, R')e(Q, R')$,
$e(P, R' + S') = e(P, R')e(P, S')$.

Request further that e is non-degenerate in the first argument, i.e., if for some P
$e(P, R') = 1$ for all $R' \in G_2$,
then P is the identity in G_1

Such an e is called a *bilinear map* or *pairing*.

Consequences of pairings

Assume that $G_1 = G_2$,
in particular $e(P, P) \neq 1$.

Then for all triples
$(P_1, P_2, P_3) \in \langle P \rangle^3$
one can decide in time polyn
in $\log \ell$ whether
$\log_P(P_3) = \log_P(P_1) \log_P(P$
by comparing
$e(P_1, P_2)$ and $e(P, P_3)$.
This means that the decision
Diffie-Hellman problem is ea

Pairings

Let $(G_1,+),(G_2,+)$ and $(G_T,\cdot)$ be groups of prime order ℓ and let
$e : G_1 \times G_2 \to G_T$
be a map satisfying
$e(P+Q,R') = e(P,R')e(Q,R'),$
$e(P,R'+S') = e(P,R')e(P,S').$

Request further that e is non-degenerate in the first argument, i.e., if for some P $e(P,R') = 1$ for all $R' \in G_2$, then P is the identity in G_1

Such an e is called a *bilinear map* or *pairing*.

Consequences of pairings

Assume that $G_1 = G_2$, in particular $e(P,P) \neq 1$.

Then for all triples $(P_1,P_2,P_3) \in \langle P\rangle^3$ one can decide in time polynomial in $\log \ell$ whether
$\log_P(P_3) = \log_P(P_1)\log_P(P_2)$
by comparing
$e(P_1,P_2)$ and $e(P,P_3)$.
This means that the decisional Diffie-Hellman problem is easy.

$+), (G_2, +)$ and $(G_T, \cdot)$
ps of prime order ℓ and let
$G_2 \to G_T$
p satisfying
$, R') = e(P, R')e(Q, R'),$
$+ S') = e(P, R')e(P, S').$

further that e is
enerate in the first
nt, i.e., if for some P
$= 1$ for all $R' \in G_2,$
s the identity in G_1

e is called a *bilinear map*
ng.

Consequences of pairings

Assume that $G_1 = G_2$,
in particular $e(P, P) \neq 1$.

Then for all triples
$(P_1, P_2, P_3) \in \langle P \rangle^3$
one can decide in time polynomial
in $\log \ell$ whether
$\log_P(P_3) = \log_P(P_1) \log_P(P_2)$
by comparing
$e(P_1, P_2)$ and $e(P, P_3)$.
This means that the decisional
Diffie-Hellman problem is easy.

The DL
secure a

Even if
transfer
to a DL
provided
$P' \in G_2$
$P \to e($

Pairings
tool if D
to solve;
calculus

+) and $(G_T, \cdot)$
e order ℓ and let

g
$P, R')e(Q, R')$,
$P, R')e(P, S')$.

at e is
the first
or some P
ll $R' \in G_2$,
tity in G_1

d a *bilinear map*

Consequences of pairings

Assume that $G_1 = G_2$,
in particular $e(P, P) \neq 1$.

Then for all triples
$(P_1, P_2, P_3) \in \langle P \rangle^3$
one can decide in time polynomial
in $\log \ell$ whether
$\log_P(P_3) = \log_P(P_1) \log_P(P_2)$
by comparing
$e(P_1, P_2)$ and $e(P, P_3)$.
This means that the decisional
Diffie-Hellman problem is easy.

The DL system G_1
secure as the syste

Even if $G_1 \neq G_2$ o
transfer the DLP i
to a DLP in G_T,
provided one can f
$P' \in G_2$ such that
$P \to e(P, P')$ is in

Pairings are intere
tool if DLP in G_T
to solve; e.g. if G_T
calculus attacks.

$_T, \cdot)$
nd let

$, R'),$
$, S').$

r map

Consequences of pairings

Assume that $G_1 = G_2$,
in particular $e(P, P) \neq 1$.

Then for all triples
$(P_1, P_2, P_3) \in \langle P \rangle^3$
one can decide in time polynomial
in $\log \ell$ whether
$\log_P(P_3) = \log_P(P_1) \log_P(P_2)$
by comparing
$e(P_1, P_2)$ and $e(P, P_3)$.
This means that the decisional
Diffie-Hellman problem is easy.

The DL system G_1 is at mos
secure as the system G_T.

Even if $G_1 \neq G_2$ one can
transfer the DLP in G_1
to a DLP in G_T,
provided one can find an ele
$P' \in G_2$ such that the map
$P \rightarrow e(P, P')$ is injective.

Pairings are interesting attac
tool if DLP in G_T is easier
to solve; e.g. if G_T has inde
calculus attacks.

Consequences of pairings

Assume that $G_1 = G_2$,
in particular $e(P, P) \neq 1$.

Then for all triples
$(P_1, P_2, P_3) \in \langle P \rangle^3$
one can decide in time polynomial in $\log \ell$ whether
$\log_P(P_3) = \log_P(P_1) \log_P(P_2)$
by comparing
$e(P_1, P_2)$ and $e(P, P_3)$.
This means that the decisional Diffie-Hellman problem is easy.

The DL system G_1 is at most as secure as the system G_T.

Even if $G_1 \neq G_2$ one can transfer the DLP in G_1 to a DLP in G_T, provided one can find an element $P' \in G_2$ such that the map $P \to e(P, P')$ is injective.

Pairings are interesting attack tool if DLP in G_T is easier to solve; e.g. if G_T has index calculus attacks.

ences of pairings

that $G_1 = G_2$,
ular $e(P, P) \neq 1$.

all triples
$P_3) \in \langle P \rangle^3$
decide in time polynomial
whether
$) = \log_P(P_1) \log_P(P_2)$
paring
) and $e(P, P_3)$.
ans that the decisional
ellman problem is easy.

The DL system G_1 is at most as secure as the system G_T.

Even if $G_1 \neq G_2$ one can transfer the DLP in G_1 to a DLP in G_T, provided one can find an element $P' \in G_2$ such that the map $P \to e(P, P')$ is injective.

Pairings are interesting attack tool if DLP in G_T is easier to solve; e.g. if G_T has index calculus attacks.

We wan
$G_1 \times G_2$
preservi

The pai
map to
a finite
More pr

To embe
into $\mathbf{F}_{q^k}$
roots of

The *em*
k is min

pairings

$= G_2$,

$P) \neq 1$.

s

$_3$

time polynomial

$P_1) \log_P(P_2)$

$, P_3)$.

he decisional

blem is easy.

The DL system G_1 is at most as secure as the system G_T.

Even if $G_1 \neq G_2$ one can transfer the DLP in G_1 to a DLP in G_T, provided one can find an element $P' \in G_2$ such that the map $P \to e(P, P')$ is injective.

Pairings are interesting attack tool if DLP in G_T is easier to solve; e.g. if G_T has index calculus attacks.

We want to define

$G_1 \times G_2 \to G_T$

preserving the gro

The pairings we w

map to the multip

a finite extension

More precisely, G_T

To embed the poi

into $\mathbf{F}_{q^k}$ there nee

roots of unity are

The *embedding de*

k is minimal with

nomial

$P_2)$

nal

sy.

The DL system G_1 is at most as secure as the system G_T.

Even if $G_1 \neq G_2$ one can transfer the DLP in G_1 to a DLP in G_T, provided one can find an element $P' \in G_2$ such that the map $P \to e(P, P')$ is injective.

Pairings are interesting attack tool if DLP in G_T is easier to solve; e.g. if G_T has index calculus attacks.

We want to define pairings

$G_1 \times G_2 \to G_T$

preserving the group structu

The pairings we will use map to the multiplicative gr a finite extension field $\mathbf{F}_{q^k}$. More precisely, $G_T \subset \mathbf{F}_{q^k}$, o

To embed the points of orde into $\mathbf{F}_{q^k}$ there need to be ℓ- roots of unity are in $\mathbf{F}^*_{q^k}$.

The *embedding degree* k sat k is minimal with $\ell \mid q^k - 1$.

The DL system G_1 is at most as secure as the system G_T.

Even if $G_1 \neq G_2$ one can transfer the DLP in G_1 to a DLP in G_T, provided one can find an element $P' \in G_2$ such that the map $P \to e(P, P')$ is injective.

Pairings are interesting attack tool if DLP in G_T is easier to solve; e.g. if G_T has index calculus attacks.

We want to define pairings
$G_1 \times G_2 \to G_T$
preserving the group structure.

The pairings we will use map to the multiplicative group of a finite extension field $\mathbf{F}_{q^k}$.
More precisely, $G_T \subset \mathbf{F}_{q^k}$, order ℓ.

To embed the points of order ℓ into $\mathbf{F}_{q^k}$ there need to be ℓ-th roots of unity are in $\mathbf{F}^*_{q^k}$.

The *embedding degree* k satisfies k is minimal with $\ell \mid q^k - 1$.

system G_1 is at most as
s the system G_T.

$G_1 \neq G_2$ one can
the DLP in G_1
P in G_T,
one can find an element
such that the map
P, P') is injective.

are interesting attack
DLP in G_T is easier
e.g. if G_T has index
attacks.

We want to define pairings
$G_1 \times G_2 \to G_T$
preserving the group structure.

The pairings we will use
map to the multiplicative group of
a finite extension field $\mathbf{F}_{q^k}$.
More precisely, $G_T \subset \mathbf{F}_{q^k}$, order ℓ.

To embed the points of order ℓ
into $\mathbf{F}_{q^k}$ there need to be ℓ-th
roots of unity are in $\mathbf{F}^*_{q^k}$.

The *embedding degree* k satisfies
k is minimal with $\ell \mid q^k - 1$.

E is sup
for $\mid E($F
it holds

Otherwis

Example
$y^2 + y =$
is supers
Each $(x$
$(x, y + 1$
All point
except f
so $\mid E($F
so $t \equiv 0$

1 is at most as
em G_T.

ne can
n G_1

find an element
the map
jective.

sting attack
is easier
$_T$ has index

We want to define pairings
$G_1 \times G_2 \to G_T$
preserving the group structure.

The pairings we will use
map to the multiplicative group of
a finite extension field $\mathbf{F}_{q^k}$.
More precisely, $G_T \subset \mathbf{F}_{q^k}$, order ℓ.

To embed the points of order ℓ
into $\mathbf{F}_{q^k}$ there need to be ℓ-th
roots of unity are in $\mathbf{F}^*_{q^k}$.

The *embedding degree* k satisfies
k is minimal with $\ell \mid q^k - 1$.

E is supersingular
for $\mid E(\mathbf{F}_q) \mid = q +$
it holds that $t \equiv 0$

Otherwise it is ord

Example:
$y^2 + y = x^3 + a_4 x$
is supersingular:
Each (x, y) point
$(x, y+1) \neq (x, y)$
All points come in
except for ∞,
so $\mid E(\mathbf{F}_{2^r}) \mid = 1 +$
so $t \equiv 0 \bmod 2$.

st as

ment

ck

x

We want to define pairings

$G_1 \times G_2 \to G_T$

preserving the group structure.

The pairings we will use

map to the multiplicative group of

a finite extension field $\mathbf{F}_{q^k}$.

More precisely, $G_T \subset \mathbf{F}_{q^k}$, order ℓ.

To embed the points of order ℓ

into $\mathbf{F}_{q^k}$ there need to be ℓ-th

roots of unity are in $\mathbf{F}^*_{q^k}$.

The *embedding degree* k satisfies

k is minimal with $\ell \mid q^k - 1$.

E is supersingular if

for $\mid E(\mathbf{F}_q) \mid = q + 1 - t$, $q =$

it holds that $t \equiv 0 \bmod p$.

Otherwise it is ordinary.

Example:

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

is supersingular:

Each (x, y) point also gives

$(x, y + 1) \neq (x, y)$.

All points come in pairs,

except for ∞,

so $\mid E(\mathbf{F}_{2^r}) \mid = 1+$even,

so $t \equiv 0 \bmod 2$.

We want to define pairings

$G_1 \times G_2 \to G_T$

preserving the group structure.

The pairings we will use

map to the multiplicative group of

a finite extension field $\mathbf{F}_{q^k}$.

More precisely, $G_T \subset \mathbf{F}_{q^k}$, order ℓ.

To embed the points of order ℓ

into $\mathbf{F}_{q^k}$ there need to be ℓ-th

roots of unity are in $\mathbf{F}^*_{q^k}$.

The *embedding degree* k satisfies

k is minimal with $\ell \mid q^k - 1$.

E is supersingular if

for $\mid E(\mathbf{F}_q) \mid = q + 1 - t$, $q = p^r$,

it holds that $t \equiv 0 \bmod p$.

Otherwise it is ordinary.

Example:

$y^2 + y = x^3 + a_4 x + a_6$ over $\mathbf{F}_{2^r}$

is supersingular:

Each (x, y) point also gives

$(x, y + 1) \neq (x, y)$.

All points come in pairs,

except for ∞,

so $\mid E(\mathbf{F}_{2^r}) \mid = 1+$even,

so $t \equiv 0 \bmod 2$.

t to define pairings

$\rightarrow G_T$

ng the group structure.

rings we will use

the multiplicative group of

extension field $\mathbf{F}_{q^k}$.

ecisely, $G_T \subset \mathbf{F}_{q^k}$, order ℓ.

ed the points of order ℓ

there need to be ℓ-th

unity are in $\mathbf{F}^*_{q^k}$.

bedding degree k satisfies

imal with $\ell \mid q^k - 1$.

E is supersingular if

for $\mid E(\mathbf{F}_q) \mid = q + 1 - t$, $q = p^r$,

it holds that $t \equiv 0 \bmod p$.

Otherwise it is ordinary.

Example:

$y^2 + y = x^3 + a_4 x + a_6$ over $\mathbf{F}_{2^r}$

is supersingular:

Each (x, y) point also gives

$(x, y + 1) \neq (x, y)$.

All points come in pairs,

except for ∞,

so $\mid E(\mathbf{F}_{2^r}) \mid = 1+$even,

so $t \equiv 0 \bmod 2$.

Embedd

Let E be

$q = p \geq$

Hasse's

$\mid t \mid \leq 2\sqrt{}$

E supers

$t \equiv 0$ mo

$\mid E(\mathbf{F}_p)$

Obvious

$(p + 1)$

so $k \leq$

over prin

e pairings

up structure.

ill use
licative group of
field $\mathbf{F}_{q^k}$.
$\subset \mathbf{F}_{q^k}$, order ℓ.

nts of order ℓ
d to be ℓ-th
in $\mathbf{F}^*_{q^k}$.

egree k satisfies
$\ell \mid q^k - 1$.

E is supersingular if
for $\mid E(\mathbf{F}_q) \mid = q + 1 - t$, $q = p^r$,
it holds that $t \equiv 0 \bmod p$.

Otherwise it is ordinary.

Example:
$y^2 + y = x^3 + a_4 x + a_6$ over $\mathbf{F}_{2^r}$
is supersingular:
Each (x, y) point also gives
$(x, y + 1) \neq (x, y)$.
All points come in pairs,
except for ∞,
so $\mid E(\mathbf{F}_{2^r}) \mid = 1+$even,
so $t \equiv 0 \bmod 2$.

Embedding degree

Let E be supersing
$q = p \geq 5$, i.e $p >$

Hasse's Theorem s
$\mid t \mid \leq 2\sqrt{p}$.
E supersingular im
$t \equiv 0 \bmod p$, so t
$\mid E(\mathbf{F}_p) \mid = p + 1$.

Obviously
$(p + 1) \mid p^2 - 1 =$
so $k \leq 2$ for super
over prime fields.

re.

oup of

rder ℓ.

er ℓ

th

tisfies

E is supersingular if
for $|E(\mathbf{F}_q)| = q + 1 - t$, $q = p^r$,
it holds that $t \equiv 0 \bmod p$.

Otherwise it is ordinary.

Example:
$y^2 + y = x^3 + a_4x + a_6$ over $\mathbf{F}_{2^r}$
is supersingular:
Each (x, y) point also gives
$(x, y + 1) \neq (x, y)$.

All points come in pairs,
except for ∞,
so $|E(\mathbf{F}_{2^r})| = 1 +$even,
so $t \equiv 0 \bmod 2$.

Embedding degrees

Let E be supersingular and
$q = p \geq 5$, i.e $p > 2\sqrt{p}$.

Hasse's Theorem states
$|t| \leq 2\sqrt{p}$.
E supersingular implies
$t \equiv 0 \bmod p$, so $t = 0$ and
$|E(\mathbf{F}_p)| = p + 1$.

Obviously
$(p + 1) \mid p^2 - 1 = (p + 1)(p$
so $k \leq 2$ for supersingular c
over prime fields.

E is supersingular if
for $|E(\mathbf{F}_q)| = q + 1 - t$, $q = p^r$,
it holds that $t \equiv 0 \bmod p$.

Otherwise it is ordinary.

Example:
$y^2 + y = x^3 + a_4x + a_6$ over $\mathbf{F}_{2^r}$
is supersingular:
Each (x, y) point also gives
$(x, y + 1) \neq (x, y)$.

All points come in pairs,
except for ∞,
so $|E(\mathbf{F}_{2^r})| = 1+$even,
so $t \equiv 0 \bmod 2$.

Embedding degrees

Let E be supersingular and
$q = p \geq 5$, i.e $p > 2\sqrt{p}$.

Hasse's Theorem states
$|t| \leq 2\sqrt{p}$.
E supersingular implies
$t \equiv 0 \bmod p$, so $t = 0$ and
$|E(\mathbf{F}_p)| = p + 1$.

Obviously
$(p + 1) \mid p^2 - 1 = (p + 1)(p - 1)$
so $k \leq 2$ for supersingular curves
over prime fields.

ersingular if

$F_q) |= q + 1 - t$, $q = p^r$,

that $t \equiv 0 \bmod p$.

se it is ordinary.

e:

$= x^3 + a_4 x + a_6$ over $\mathbf{F}_{2^r}$

singular:

$, y)$ point also gives

$1) \neq (x, y)$.

ts come in pairs,

or ∞,

$_{2^r}) |= 1+$even,

mod 2.

Embedding degrees

Let E be supersingular and

$q = p \geq 5$, i.e $p > 2\sqrt{p}$.

Hasse's Theorem states

$| t | \leq 2\sqrt{p}$.

E supersingular implies

$t \equiv 0 \bmod p$, so $t = 0$ and

$| E(\mathbf{F}_p) | = p + 1$.

Obviously

$(p + 1) \mid p^2 - 1 = (p + 1)(p - 1)$

so $k \leq 2$ for supersingular curves

over prime fields.

Distortio

For supe

exist ma

$\phi : E(\mathbf{F}_q$

i.e. map

$\tilde{e}(P, P)$

$e(P, \phi(P$

Such a r

distortio

These m

the only

compute

Weil pai

which ha

if

$1 - t$, $q = p^r$,

mod p.

linary.

$x + a_6$ over $\mathbf{F}_{2^r}$

also gives

).

pairs,

even,

Embedding degrees

Let E be supersingular and $q = p \geq 5$, i.e $p > 2\sqrt{p}$.

Hasse's Theorem states
$| \ t \ | \leq 2\sqrt{p}$.
E supersingular implies
$t \equiv 0 \bmod p$, so $t = 0$ and
$| \ E(\mathbf{F}_p) \ | = p + 1$.

Obviously
$(p+1) \mid p^2 - 1 = (p+1)(p-1)$
so $k \leq 2$ for supersingular curves over prime fields.

Distortion maps

For supersingular
exist maps
$\phi : E(\mathbf{F}_q) \rightarrow E(\mathbf{F}$
i.e. maps $G_1 \rightarrow G$
$\tilde{e}(P, P) \neq 1$ for
$e(P, \phi(P))$.
Such a map is call
distortion map.

These maps are im
the only pairings w
compute are varia
Weil pairing and
which have $e(P, P$

$= p^r$,

er $\mathbf{F}_{2^r}$

Embedding degrees

Let E be supersingular and $q = p \geq 5$, i.e $p > 2\sqrt{p}$.

Hasse's Theorem states
$| t | \leq 2\sqrt{p}$.
E supersingular implies
$t \equiv 0 \bmod p$, so $t = 0$ and
$| E(\mathbf{F}_p) | = p + 1$.

Obviously
$(p+1) \mid p^2 - 1 = (p+1)(p-1)$
so $k \leq 2$ for supersingular curves over prime fields.

Distortion maps

For supersingular curves the
exist maps
$\phi : E(\mathbf{F}_q) \to E(\mathbf{F}_{q^k})$
i.e. maps $G_1 \to G_2$, giving
$\tilde{e}(P, P) \neq 1$ for $\tilde{e}(P, P) =$
$e(P, \phi(P))$.
Such a map is called a
distortion map.

These maps are important s
the only pairings we know h
compute are variants of
Weil pairing and *Tate pairin*
which have $e(P, P) = 1$.

Embedding degrees

Let E be supersingular and $q = p \geq 5$, i.e $p > 2\sqrt{p}$.

Hasse's Theorem states $| \, t \, | \leq 2\sqrt{p}$.
E supersingular implies $t \equiv 0 \bmod p$, so $t = 0$ and $| \, E(\mathbf{F}_p) \, | = p + 1$.

Obviously $(p+1) \mid p^2 - 1 = (p+1)(p-1)$ so $k \leq 2$ for supersingular curves over prime fields.

Distortion maps

For supersingular curves there exist maps
$\phi : E(\mathbf{F}_q) \to E(\mathbf{F}_{q^k})$
i.e. maps $G_1 \to G_2$, giving $\tilde{e}(P, P) \neq 1$ for $\tilde{e}(P, P) = e(P, \phi(P))$.
Such a map is called a *distortion map*.

These maps are important since the only pairings we know how to compute are variants of *Weil pairing* and *Tate pairing* which have $e(P, P) = 1$.

ing degrees

e supersingular and
5, i.e $p > 2\sqrt{p}$.

Theorem states
$\sqrt{p}$.
singular implies
od p, so $t = 0$ and
$| = p + 1$.

ly
$| \, p^2 - 1 = (p+1)(p-1)$
2 for supersingular curves
me fields.

Distortion maps

For supersingular curves there exist maps
$\phi : E(\mathbf{F}_q) \to E(\mathbf{F}_{q^k})$
i.e. maps $G_1 \to G_2$, giving
$\tilde{e}(P, P) \neq 1$ for $\tilde{e}(P, P) = e(P, \phi(P))$.
Such a map is called a *distortion map*.

These maps are important since the only pairings we know how to compute are variants of *Weil pairing* and *Tate pairing* which have $e(P, P) = 1$.

Example
$y^2 = x^3$
for $p \equiv 3$
Distortio
$(x, y) \mapsto$

$y^2 = x^3$
Distortio
with j^3

In both
so $k = 2$

es

gular and
$2\sqrt{p}$.

states

nplies
$= 0$ and

$(p+1)(p-1)$
rsingular curves

Distortion maps

For supersingular curves there exist maps
$\phi : E(\mathbf{F}_q) \to E(\mathbf{F}_{q^k})$
i.e. maps $G_1 \to G_2$, giving $\tilde{e}(P,P) \neq 1$ for $\tilde{e}(P,P) = e(P,\phi(P))$.
Such a map is called a *distortion map*.

These maps are important since the only pairings we know how to compute are variants of *Weil pairing* and *Tate pairing* which have $e(P,P) = 1$.

Examples:
$y^2 = x^3 + a_4 x$,
for $p \equiv 3 \pmod 4$
Distortion map
$(x,y) \mapsto (-x, \sqrt{-}$

$y^2 = x^3 + a_6$, for
Distortion map $(x$
with $j^3 = 1, j \neq 1$

In both cases, $\#E$
so $k = 2$.

$p - 1)$

urves

Distortion maps

For supersingular curves there exist maps

$\phi : E(\mathbf{F}_q) \to E(\mathbf{F}_{q^k})$

i.e. maps $G_1 \to G_2$, giving $\tilde{e}(P, P) \neq 1$ for $\tilde{e}(P, P) = e(P, \phi(P))$.

Such a map is called a *distortion map*.

These maps are important since the only pairings we know how to compute are variants of *Weil pairing* and *Tate pairing* which have $e(P, P) = 1$.

Examples:

$y^2 = x^3 + a_4 x$,

for $p \equiv 3 \pmod 4$.

Distortion map

$(x, y) \mapsto (-x, \sqrt{-1}y)$.

$y^2 = x^3 + a_6$, for $p \equiv 2 \quad (\mathrm{m}$

Distortion map $(x, y) \mapsto (j$

with $j^3 = 1, j \neq 1$.

In both cases, $\#E(\mathbf{F}_p) = p$

so $k = 2$.

Distortion maps

For supersingular curves there exist maps
$\phi : E(\mathbf{F}_q) \to E(\mathbf{F}_{q^k})$
i.e. maps $G_1 \to G_2$, giving $\tilde{e}(P,P) \neq 1$ for $\tilde{e}(P,P) = e(P,\phi(P))$.
Such a map is called a *distortion map*.

These maps are important since the only pairings we know how to compute are variants of *Weil pairing* and *Tate pairing* which have $e(P,P)=1$.

Examples:
$y^2 = x^3 + a_4 x$,
for $p \equiv 3 \pmod 4$.
Distortion map
$(x,y) \mapsto (-x, \sqrt{-1}y)$.

$y^2 = x^3 + a_6$, for $p \equiv 2 \pmod 3$.
Distortion map $(x,y) \mapsto (jx, y)$ with $j^3 = 1, j \neq 1$.

In both cases, $\#E(\mathbf{F}_p) = p+1$, so $k=2$.

on maps

ersingular curves there

ps

$_q) \to E(\mathbf{F}_{q^k})$

s $G_1 \to G_2$, giving

$\neq 1$ for $\tilde{e}(P, P) =$

$P))$.

map is called a

n map.

aps are important since

pairings we know how to

e are variants of

iring and *Tate pairing*

ave $e(P, P) = 1$.

Examples:

$y^2 = x^3 + a_4 x$,

for $p \equiv 3 \pmod 4$.

Distortion map

$(x, y) \mapsto (-x, \sqrt{-1}y)$.

$y^2 = x^3 + a_6$, for $p \equiv 2 \pmod 3$.

Distortion map $(x, y) \mapsto (j x, y)$

with $j^3 = 1, j \neq 1$.

In both cases, $\#E(\mathbf{F}_p) = p + 1$,

so $k = 2$.

Example

$p = 10$

$y^2 = x^3$

Has 100

$P = (1$

of order

$nP = (6$

Construc

$\phi(P) =$

Invoke n

$e(P, \phi(P$

$e(Q, \phi(P$

Solve wi

$n = 786$

(Btw. th

curves there

q^k)

$_2$, giving

$\check{e}(P, P) =$

ed a

nportant since

ve know how to

nts of

Tate pairing

$) = 1$.

Examples:

$y^2 = x^3 + a_4 x$,

for $p \equiv 3 \pmod 4$.

Distortion map

$(x, y) \mapsto (-x, \sqrt{-1} y)$.

$y^2 = x^3 + a_6$, for $p \equiv 2 \pmod 3$.

Distortion map $(x, y) \mapsto (jx, y)$

with $j^3 = 1, j \neq 1$.

In both cases, $\#E(\mathbf{F}_p) = p + 1$,

so $k = 2$.

Example from Tue

$p = 1000003 \equiv$

$y^2 = x^3 - x$ over

Has $1000004 = p$

$P = (101384, 614$

of order 500002.

$nP = (670366, 74$

Construct $\mathbf{F}_{p^2}$ as

$\phi(P) = (898619, 6$

Invoke magma and

$e(P, \phi(P)) = 3872$

$e(Q, \phi(P)) = 6094$

Solve with index c

$n = 78654$.

(Btw. this is the c

re

=

ince

ow to

g

Examples:

$y^2 = x^3 + a_4 x$,

for $p \equiv 3 \pmod 4$.

Distortion map

$(x, y) \mapsto (-x, \sqrt{-1} y)$.

$y^2 = x^3 + a_6$, for $p \equiv 2 \pmod 3$.

Distortion map $(x, y) \mapsto (jx, y)$

with $j^3 = 1, j \neq 1$.

In both cases, $\#E(\mathbf{F}_p) = p + 1$,

so $k = 2$.

Example from Tuesday:

$p = 1000003 \equiv 3 \bmod 4$ a

$y^2 = x^3 - x$ over $\mathbf{F}_p$.

Has $1000004 = p + 1$ points

$P = (101384, 614510)$ is a

of order 500002.

$nP = (670366, 740819)$.

Construct $\mathbf{F}_{p^2}$ as $\mathbf{F}_p(i)$.

$\phi(P) = (898619, 614510i)$.

Invoke magma and compute

$e(P, \phi(P)) = 387265 + 276$

$e(Q, \phi(P)) = 609466 + 807$

Solve with index calculus to

$n = 78654$.

(Btw. this is the clock).

Examples:

$y^2 = x^3 + a_4 x$,

for $p \equiv 3 \pmod 4$.

Distortion map

$(x, y) \mapsto (-x, \sqrt{-1}y)$.

$y^2 = x^3 + a_6$, for $p \equiv 2 \pmod 3$.

Distortion map $(x, y) \mapsto (jx, y)$

with $j^3 = 1, j \neq 1$.

In both cases, $\#E(\mathbf{F}_p) = p + 1$,

so $k = 2$.

Example from Tuesday:

$p = 1000003 \equiv 3$ mod 4 and

$y^2 = x^3 - x$ over $\mathbf{F}_p$.

Has $1000004 = p + 1$ points.

$P = (101384, 614510)$ is a point

of order 500002.

$nP = (670366, 740819)$.

Construct $\mathbf{F}_{p^2}$ as $\mathbf{F}_p(i)$.

$\phi(P) = (898619, 614510i)$.

Invoke magma and compute

$e(P, \phi(P)) = 387265 + 276048i$;

$e(Q, \phi(P)) = 609466 + 807033i$.

Solve with index calculus to get

$n = 78654$.

(Btw. this is the clock).

es:

$+ a_4 x$,

3 (mod 4).

on map

$\mapsto (-x, \sqrt{-1}y)$.

$+ a_6$, for $p \equiv 2 \pmod 3$.

on map $(x, y) \mapsto (jx, y)$

$= 1, j \neq 1$.

cases, $\#E(\mathbf{F}_p) = p + 1$,

2.

Example from Tuesday:

$p = 1000003 \equiv 3 \bmod 4$ and

$y^2 = x^3 - x$ over $\mathbf{F}_p$.

Has $1000004 = p + 1$ points.

$P = (101384, 614510)$ is a point of order 500002.

$nP = (670366, 740819)$.

Construct $\mathbf{F}_{p^2}$ as $\mathbf{F}_p(i)$.

$\phi(P) = (898619, 614510i)$.

Invoke magma and compute

$e(P, \phi(P)) = 387265 + 276048i$;

$e(Q, \phi(P)) = 609466 + 807033i$.

Solve with index calculus to get

$n = 78654$.

(Btw. this is the clock).

Summar

Menezes

for E su

For $p =$

For $p =$

Over $\mathbf{F}_p$

These b

Not only

MNT cu

curves w

Other ex

pairing-b

but sma

random

4).

$\overline{1}y)$.

$p \equiv 2 \pmod 3$.

$, y) \mapsto (j x, y)$

.

$(\mathbf{F}_p) = p + 1$,

Example from Tuesday:

$p = 1000003 \equiv 3 \bmod 4$ and $y^2 = x^3 - x$ over $\mathbf{F}_p$.

Has $1000004 = p + 1$ points.

$P = (101384, 614510)$ is a point of order 500002.

$nP = (670366, 740819)$.

Construct $\mathbf{F}_{p^2}$ as $\mathbf{F}_p(i)$.

$\phi(P) = (898619, 614510i)$.

Invoke magma and compute

$e(P, \phi(P)) = 387265 + 276048i$;

$e(Q, \phi(P)) = 609466 + 807033i$.

Solve with index calculus to get $n = 78654$.

(Btw. this is the clock).

Summary of pairin

Menezes, Okamot

for E supersingula

For $p = 2$ have k

For $p = 3$ we $k \leq$

Over $\mathbf{F}_p$, $p \geq 5$ ha

These bounds are

Not only supersin

MNT curves are n

curves with small

Other examples co

pairing-based cryp

but small k unlike

random curve.

nod 3).

x, y)

+ 1,

Example from Tuesday:
$p = 1000003 \equiv 3 \bmod 4$ and
$y^2 = x^3 - x$ over $\mathbf{F}_p$.
Has $1000004 = p + 1$ points.
$P = (101384, 614510)$ is a point of order 500002.
$nP = (670366, 740819)$.
Construct $\mathbf{F}_{p^2}$ as $\mathbf{F}_p(i)$.
$\phi(P) = (898619, 614510i)$.

Invoke magma and compute
$e(P, \phi(P)) = 387265 + 276048i$;
$e(Q, \phi(P)) = 609466 + 807033i$.
Solve with index calculus to get
$n = 78654$.
(Btw. this is the clock).

Summary of pairings

Menezes, Okamoto, and Va
for E supersingular:
For $p = 2$ have $k \leq 4$.
For $p = 3$ we $k \leq 6$
Over $\mathbf{F}_p$, $p \geq 5$ have $k \leq 2$.
These bounds are attained.

Not only supersingular curve
MNT curves are non-supersi
curves with small k.
Other examples constructed
pairing-based cryptography
but small k unlikely to occu
random curve.

Example from Tuesday:

$p = 1000003 \equiv 3 \bmod 4$ and $y^2 = x^3 - x$ over $\mathbf{F}_p$.

Has $1000004 = p + 1$ points.

$P = (101384, 614510)$ is a point of order 500002.

$nP = (670366, 740819)$.

Construct $\mathbf{F}_{p^2}$ as $\mathbf{F}_p(i)$.

$\phi(P) = (898619, 614510i)$.

Invoke magma and compute

$e(P, \phi(P)) = 387265 + 276048i$;

$e(Q, \phi(P)) = 609466 + 807033i$.

Solve with index calculus to get $n = 78654$.

(Btw. this is the clock).

Summary of pairings

Menezes, Okamoto, and Vanstone for E supersingular:

For $p = 2$ have $k \le 4$.

For $p = 3$ we $k \le 6$

Over $\mathbf{F}_p$, $p \ge 5$ have $k \le 2$.

These bounds are attained.

Not only supersingular curves:

MNT curves are non-supersingular curves with small k.

Other examples constructed for pairing-based cryptography – but small k unlikely to occur for random curve.

e from Tuesday:
00003 $\equiv$ 3 mod 4 and
$-x$ over $\mathbf{F}_p$.
0004 $= p + 1$ points.
01384, 614510) is a point
500002.
570366, 740819).
ct $\mathbf{F}_{p^2}$ as $\mathbf{F}_p(i)$.
(898619, 614510i).

nagma and compute
$)) = 387265 + 276048i$;
$)) = 609466 + 807033i$.
th index calculus to get
54.
his is the clock).

<u>Summary of pairings</u>

Menezes, Okamoto, and Vanstone
for E supersingular:
For $p = 2$ have $k \leq 4$.
For $p = 3$ we $k \leq 6$
Over $\mathbf{F}_p$, $p \geq 5$ have $k \leq 2$.
These bounds are attained.

Not only supersingular curves:
MNT curves are non-supersingular
curves with small k.
Other examples constructed for
pairing-based cryptography –
but small k unlikely to occur for
random curve.

<u>Summar</u>

Definitio
does not
For $\mathbf{F}_{p^n}$
can map
Watch o
curves o

Anomalo
If $E/\mathbf{F}_p$
then tra
Very eas
Not a pr
attack a
order-p s

esday:

3 mod 4 and

$\mathbf{F}_p$.

$+ 1$ points.

510) is a point

0819).

$\mathbf{F}_p(i)$.

614510i).

d compute

265 + 276048i;

466 + 807033i.

alculus to get

clock).

Summary of pairings

Menezes, Okamoto, and Vanstone for E supersingular:
For $p = 2$ have $k \leq 4$.
For $p = 3$ we $k \leq 6$
Over $\mathbf{F}_p$, $p \geq 5$ have $k \leq 2$.
These bounds are attained.

Not only supersingular curves:
MNT curves are non-supersingular curves with small k.
Other examples constructed for pairing-based cryptography – but small k unlikely to occur for random curve.

Summary of other

Definition of embe
does not cover all
For $\mathbf{F}_{p^n}$ watch out
can map to $\mathbf{F}_{p^{km}}$
Watch out for this
curves over $\mathbf{F}_{p^n}$.

Anomalous curves
If $E/\mathbf{F}_p$ has $\#E(\mathbf{F}$
then transfer $E(\mathbf{F}_p$
Very easy DLP.
Not a problem for
attack applies to
order-p subgroup.

nd

s.
point

e
048i;
033i.
get

Summary of pairings

Menezes, Okamoto, and Vanstone for E supersingular:
For $p = 2$ have $k \leq 4$.
For $p = 3$ we $k \leq 6$
Over $\mathbf{F}_p$, $p \geq 5$ have $k \leq 2$.
These bounds are attained.

Not only supersingular curves:
MNT curves are non-supersingular curves with small k.
Other examples constructed for pairing-based cryptography – but small k unlikely to occur for random curve.

Summary of other attacks

Definition of embedding deg
does not cover all attacks.
For $\mathbf{F}_{p^n}$ watch out that pair
can map to $\mathbf{F}_{p^{km}}$ with $m <$
Watch out for this when sele
curves over $\mathbf{F}_{p^n}$.

Anomalous curves:
If $E/\mathbf{F}_p$ has $\#E(\mathbf{F}_p) = p$
then transfer $E(\mathbf{F}_p)$ to $(\mathbf{F}_p,$
Very easy DLP.
Not a problem for Koblitz cu
attack applies to
order-p subgroup.

Summary of pairings

Menezes, Okamoto, and Vanstone for E supersingular:
For $p = 2$ have $k \le 4$.
For $p = 3$ we $k \le 6$
Over $\mathbf{F}_p$, $p \ge 5$ have $k \le 2$.
These bounds are attained.

Not only supersingular curves:
MNT curves are non-supersingular curves with small k.
Other examples constructed for pairing-based cryptography – but small k unlikely to occur for random curve.

Summary of other attacks

Definition of embedding degree does not cover all attacks.
For $\mathbf{F}_{p^n}$ watch out that pairing can map to $\mathbf{F}_{p^{km}}$ with $m < n$.
Watch out for this when selecting curves over $\mathbf{F}_{p^n}$.

Anomalous curves:
If $E/\mathbf{F}_p$ has $\#E(\mathbf{F}_p) = p$ then transfer $E(\mathbf{F}_p)$ to $(\mathbf{F}_p, +)$.
Very easy DLP.
Not a problem for Koblitz curves, attack applies to order-p subgroup.

y of pairings

s, Okamoto, and Vanstone
persingular:
2 have $k \leq 4$.
3 we $k \leq 6$
, $p \geq 5$ have $k \leq 2$.
ounds are attained.

y supersingular curves:
rves are non-supersingular
ith small k.
kamples constructed for
based cryptography –
ll k unlikely to occur for
curve.

Summary of other attacks

Definition of embedding degree does not cover all attacks.
For $\mathbf{F}_{p^n}$ watch out that pairing can map to $\mathbf{F}_{p^{km}}$ with $m < n$.
Watch out for this when selecting curves over $\mathbf{F}_{p^n}$.

Anomalous curves:
If $E/\mathbf{F}_p$ has $\#E(\mathbf{F}_p) = p$ then transfer $E(\mathbf{F}_p)$ to $(\mathbf{F}_p, +)$.
Very easy DLP.
Not a problem for Koblitz curves, attack applies to order-p subgroup.

Weil des
Maps D
to DLP
J has la
represen
degree.
This is e
is not to
Particula
with J i
hyperelli
For genu
$\tilde{O}(p^{2-\frac{2}{g}}$
base des
polynom

ngs

o, and Vanstone
r:
≤ 4.
6
ave $k \leq 2$.
attained.

gular curves:
on-supersingular
k.
onstructed for
tography –
ly to occur for

Summary of other attacks

Definition of embedding degree does not cover all attacks.
For $\mathbf{F}_{p^n}$ watch out that pairing can map to $\mathbf{F}_{p^{km}}$ with $m < n$.
Watch out for this when selecting curves over $\mathbf{F}_{p^n}$.

Anomalous curves:
If $E/\mathbf{F}_p$ has $\#E(\mathbf{F}_p) = p$ then transfer $E(\mathbf{F}_p)$ to $(\mathbf{F}_p, +)$.
Very easy DLP.
Not a problem for Koblitz curves, attack applies to order-p subgroup.

Weil descent:
Maps DLP in E o
to DLP on variety
J has larger dimen
represented as pol
degree. $\Rightarrow$ index c
This is efficient if
is not too big.
Particularly nice to
with J if it is the
hyperelliptic curve
For genus g get co
$\tilde{O}(p^{2-\frac{2}{g+1}})$ with th
base described bef
polynomials have

nstone

es:
ngular

for

–
r for

Summary of other attacks

Definition of embedding degree does not cover all attacks.
For $\mathbf{F}_{p^n}$ watch out that pairing can map to $\mathbf{F}_{p^{km}}$ with $m < n$.
Watch out for this when selecting curves over $\mathbf{F}_{p^n}$.

Anomalous curves:
If $E/\mathbf{F}_p$ has $\#E(\mathbf{F}_p) = p$ then transfer $E(\mathbf{F}_p)$ to $(\mathbf{F}_p, +)$.
Very easy DLP.
Not a problem for Koblitz curves, attack applies to order-p subgroup.

Weil descent:
Maps DLP in E over $\mathbf{F}_{p^{mn}}$
to DLP on variety J over $\mathbf{F}_p$
J has larger dimension; elem
represented as polynomials o
degree. $\Rightarrow$ index calculus.

This is efficient if dimension
is not too big.

Particularly nice to compute
with J if it is the Jacobian o
hyperelliptic curve C.
For genus g get complexity
$\tilde{O}(p^{2-\frac{2}{g+1}})$ with the factor
base described before, since
polynomials have degree $<=$

Summary of other attacks

Definition of embedding degree does not cover all attacks.
For $\mathbf{F}_{p^n}$ watch out that pairing can map to $\mathbf{F}_{p^{km}}$ with $m < n$.
Watch out for this when selecting curves over $\mathbf{F}_{p^n}$.

Anomalous curves:
If $E/\mathbf{F}_p$ has $\#E(\mathbf{F}_p) = p$ then transfer $E(\mathbf{F}_p)$ to $(\mathbf{F}_p, +)$.
Very easy DLP.
Not a problem for Koblitz curves, attack applies to order-p subgroup.

Weil descent:
Maps DLP in E over $\mathbf{F}_{p^{mn}}$ to DLP on variety J over $\mathbf{F}_{p^n}$.
J has larger dimension; elements represented as polynomials of low degree. $\Rightarrow$ index calculus.

This is efficient if dimension of J is not too big.

Particularly nice to compute with J if it is the Jacobian of a hyperelliptic curve C.
For genus g get complexity $\tilde{O}(p^{2-\frac{2}{g+1}})$ with the factor base described before, since polynomials have degree $<= g$.