Modern ECC signatures

2011 Bernstein–Duif–Lange–Schwabe–Yang:
Ed25519 signature scheme = EdDSA using conservative Curve25519 elliptic curve.
https://ed25519.cr.yp.to

32-byte public keys,
64-byte signatures,
$\approx 2^{125.8}$ security level.

Deployed in SSH, Signal, many more applications:
https://ianix.com/pub/ed25519-deployment.html

Many papers have explored Curve25519/Ed25519 speed.

e.g. 2015 Chou software:
on Intel Sandy Bridge (2011),
57164 cycles for keygen,
63526 cycles for signature,
205741 cycles for verification,
159128 cycles for ECDH.

Compare to, e.g., 2000 Brown–Hankerson–López–Menezes:
on Intel Pentium II (1997),
1920000 cycles for ECDH
using NIST P-256 curve.

ECC signatures

rnstein–Duif–Lange–
e–Yang:
9 signature scheme =
using conservative
519 elliptic curve.
//ed25519.cr.yp.to

public keys,
signatures,
security level.

d in SSH, Signal,
ore applications:
//ianix.com/pub
19-deployment.html

Many papers have explored Curve25519/Ed25519 speed.

e.g. 2015 Chou software: on Intel Sandy Bridge (2011), 57164 cycles for keygen, 63526 cycles for signature, 205741 cycles for verification, 159128 cycles for ECDH.

Compare to, e.g., 2000 Brown–Hankerson–López–Menezes: on Intel Pentium II (1997), 1920000 cycles for ECDH using NIST P-256 curve.

A_C: cyc
Does A_C
A is bett

atures

uif–Lange–

e scheme =
servative
c curve.
9.cr.yp.to

s,
,
evel.

Signal,
ations:
com/pub
yment.html

Many papers have explored Curve25519/Ed25519 speed.

e.g. 2015 Chou software: on Intel Sandy Bridge (2011), 57164 cycles for keygen, 63526 cycles for signature, 205741 cycles for verification, 159128 cycles for ECDH.

Compare to, e.g., 2000 Brown–Hankerson–López–Menezes: on Intel Pentium II (1997), 1920000 cycles for ECDH using NIST P-256 curve.

A_C: cycles for alg
Does $A_C < B_D$ pr
A is better than B

–

=

to

ml

Many papers have explored Curve25519/Ed25519 speed.

e.g. 2015 Chou software: on Intel Sandy Bridge (2011), 57164 cycles for keygen, 63526 cycles for signature, 205741 cycles for verification, 159128 cycles for ECDH.

Compare to, e.g., 2000 Brown–Hankerson–López–Menezes: on Intel Pentium II (1997), 1920000 cycles for ECDH using NIST P-256 curve.

A_C: cycles for alg A on CPU

Does $A_C < B_D$ prove that A is better than B?

Many papers have explored Curve25519/Ed25519 speed.

e.g. 2015 Chou software: on Intel Sandy Bridge (2011), 57164 cycles for keygen, 63526 cycles for signature, 205741 cycles for verification, 159128 cycles for ECDH.

Compare to, e.g., 2000 Brown–Hankerson–López–Menezes: on Intel Pentium II (1997), 1920000 cycles for ECDH using NIST P-256 curve.

A_C: cycles for alg A on CPU C.

Does $A_C < B_D$ prove that A is better than B?

Many papers have explored Curve25519/Ed25519 speed.

e.g. 2015 Chou software: on Intel Sandy Bridge (2011), 57164 cycles for keygen, 63526 cycles for signature, 205741 cycles for verification, 159128 cycles for ECDH.

Compare to, e.g., 2000 Brown–Hankerson–López–Menezes: on Intel Pentium II (1997), 1920000 cycles for ECDH using NIST P-256 curve.

A_C: cycles for alg A on CPU C. Does $A_C < B_D$ prove that A is better than B?

No! Beware change in CPU.

Maybe $A_C > B_C$; $A_D > B_D$; C does more work per cycle than D, thanks to CPU manufacturer.

Sometimes people measure cost in seconds instead of cycles. Then they benefit from more work per cycle and from more cycles per second.

apers have explored
519/Ed25519 speed.

5 Chou software:
Sandy Bridge (2011),
ycles for keygen,
ycles for signature,
cycles for verification,
cycles for ECDH.

e to, e.g., 2000 Brown–
on–López–Menezes:
Pentium II (1997),
) cycles for ECDH
ST P-256 curve.

A_C: cycles for alg A on CPU C.
Does $A_C < B_D$ prove that
A is better than B?

No! Beware change in CPU.

Maybe $A_C > B_C$; $A_D > B_D$;
C does more work per cycle than
D, thanks to CPU manufacturer.

Sometimes people measure cost
in seconds instead of cycles.
Then they benefit
from more work per cycle and
from more cycles per second.

Better c
(still rais

ECDH o
(still not
1920000
832457 c

ECDH o
374000 c
(from 20
159128 c

Verificat
529000 c
205741 c

explored
519 speed.

ftware:
dge (2011),
eygen,
gnature,
verification,
ECDH.

2000 Brown–
-Menezes:
I (1997),
ECDH
curve.

A_C: cycles for alg A on CPU C.
Does $A_C < B_D$ prove that
A is better than B?

No! Beware change in CPU.

Maybe $A_C > B_C$; $A_D > B_D$;
C does more work per cycle than
D, thanks to CPU manufacturer.

Sometimes people measure cost
in seconds instead of cycles.
Then they benefit
from more work per cycle and
from more cycles per second.

Better comparison
(still raising many

ECDH on Intel Pe
(still not exactly t
1920000 cycles for
832457 cycles for

ECDH on Sandy B
374000 cycles for
(from 2013 Gueron
159128 cycles for

Verification on Sa
529000 cycles for
205741 cycles for

.

),

n,

vn–

A_C: cycles for alg A on CPU C.

Does $A_C < B_D$ prove that A is better than B?

No! Beware change in CPU.

Maybe $A_C > B_C$; $A_D > B_D$; C does more work per cycle than D, thanks to CPU manufacturer.

Sometimes people measure cost in seconds instead of cycles.
Then they benefit
from more work per cycle and
from more cycles per second.

Better comparisons
(still raising many questions

ECDH on Intel Pentium II/I
(still not exactly the same):
1920000 cycles for NIST P-2
832457 cycles for Curve2551

ECDH on Sandy Bridge:
374000 cycles for NIST P-25
(from 2013 Gueron–Krasnov
159128 cycles for Curve2551

Verification on Sandy Bridge
529000 cycles for ECDSA-P-
205741 cycles for Ed25519.

A_C: cycles for alg A on CPU C.
Does $A_C < B_D$ prove that A is better than B?

No! Beware change in CPU.

Maybe $A_C > B_C$; $A_D > B_D$; C does more work per cycle than D, thanks to CPU manufacturer.

Sometimes people measure cost in seconds instead of cycles.
Then they benefit from more work per cycle and from more cycles per second.

Better comparisons (still raising many questions):

ECDH on Intel Pentium II/III (still not exactly the same):
1920000 cycles for NIST P-256,
832457 cycles for Curve25519.

ECDH on Sandy Bridge:
374000 cycles for NIST P-256 (from 2013 Gueron–Krasnov),
159128 cycles for Curve25519.

Verification on Sandy Bridge:
529000 cycles for ECDSA-P-256,
205741 cycles for Ed25519.

les for alg A on CPU C.
$_C < B_D$ prove that
ter than B?

vare change in CPU.

$A_C > B_C$; $A_D > B_D$;
more work per cycle than
ks to CPU manufacturer.

nes people measure cost
ds instead of cycles.
ey benefit
ore work per cycle and
ore cycles per second.

Better comparisons
(still raising many questions):

ECDH on Intel Pentium II/III
(still not exactly the same):
1920000 cycles for NIST P-256,
832457 cycles for Curve25519.

ECDH on Sandy Bridge:
374000 cycles for NIST P-256
(from 2013 Gueron–Krasnov),
159128 cycles for Curve25519.

Verification on Sandy Bridge:
529000 cycles for ECDSA-P-256,
205741 cycles for Ed25519.

For each
on each
on each

Simplest
are much

Question
and soft
How to
on, e.g.,
Ed2551

Answers
design:

A on CPU C.
ove that
3?

ge in CPU.

$A_D > B_D$;
per cycle than
manufacturer.

measure cost
of cycles.

er cycle and
per second.

Better comparisons
(still raising many questions):

ECDH on Intel Pentium II/III
(still not exactly the same):
1920000 cycles for NIST P-256,
832457 cycles for Curve25519.

ECDH on Sandy Bridge:
374000 cycles for NIST P-256
(from 2013 Gueron–Krasnov),
159128 cycles for Curve25519.

Verification on Sandy Bridge:
529000 cycles for ECDSA-P-256,
205741 cycles for Ed25519.

For each of these
on each of these c
on each of these C

Simplest implemen
are much, much, m

Questions in algor
and software engin
How to build the f
on, e.g., an ARM
Ed25519 signature

Answers feed back
design: e.g., choos

J C.

.

;

than

turer.

cost

nd

l.

Better comparisons
(still raising many questions):

ECDH on Intel Pentium II/III
(still not exactly the same):
1920000 cycles for NIST P-256,
832457 cycles for Curve25519.

ECDH on Sandy Bridge:
374000 cycles for NIST P-256
(from 2013 Gueron–Krasnov),
159128 cycles for Curve25519.

Verification on Sandy Bridge:
529000 cycles for ECDSA-P-256,
205741 cycles for Ed25519.

For each of these operations
on each of these curves,
on each of these CPUs:

Simplest implementations
are much, much, much slow

Questions in algorithm desig
and software engineering:
How to build the fastest sof
on, e.g., an ARM Cortex-A8
Ed25519 signature verificati

Answers feed back into cryp
design: e.g., choosing fast c

Better comparisons
(still raising many questions):

ECDH on Intel Pentium II/III
(still not exactly the same):
1920000 cycles for NIST P-256,
832457 cycles for Curve25519.

ECDH on Sandy Bridge:
374000 cycles for NIST P-256
(from 2013 Gueron–Krasnov),
159128 cycles for Curve25519.

Verification on Sandy Bridge:
529000 cycles for ECDSA-P-256,
205741 cycles for Ed25519.

For each of these operations,
on each of these curves,
on each of these CPUs:

Simplest implementations
are much, much, much slower.

Questions in algorithm design
and software engineering:
How to build the fastest software
on, e.g., an ARM Cortex-A8 for
Ed25519 signature verification?

Answers feed back into crypto
design: e.g., choosing fast curves.

omparisons
sing many questions):

n Intel Pentium II/III
exactly the same):
) cycles for NIST P-256,
cycles for Curve25519.

n Sandy Bridge:
cycles for NIST P-256
013 Gueron–Krasnov),
cycles for Curve25519.

ion on Sandy Bridge:
cycles for ECDSA-P-256,
cycles for Ed25519.

For each of these operations,
on each of these curves,
on each of these CPUs:

Simplest implementations
are much, much, much slower.

Questions in algorithm design
and software engineering:
How to build the fastest software
on, e.g., an ARM Cortex-A8 for
Ed25519 signature verification?

Answers feed back into crypto
design: e.g., choosing fast curves.

Several l

ECC
verify S

Point
P, Q

Field
$x_1, x_2 \mapsto$

Machin
32-bit r

Gat
AND,

s
questions):

ntium II/III
he same):
NIST P-256,
Curve25519.

Bridge:
NIST P-256
n–Krasnov),
Curve25519.

ndy Bridge:
ECDSA-P-256,
Ed25519.

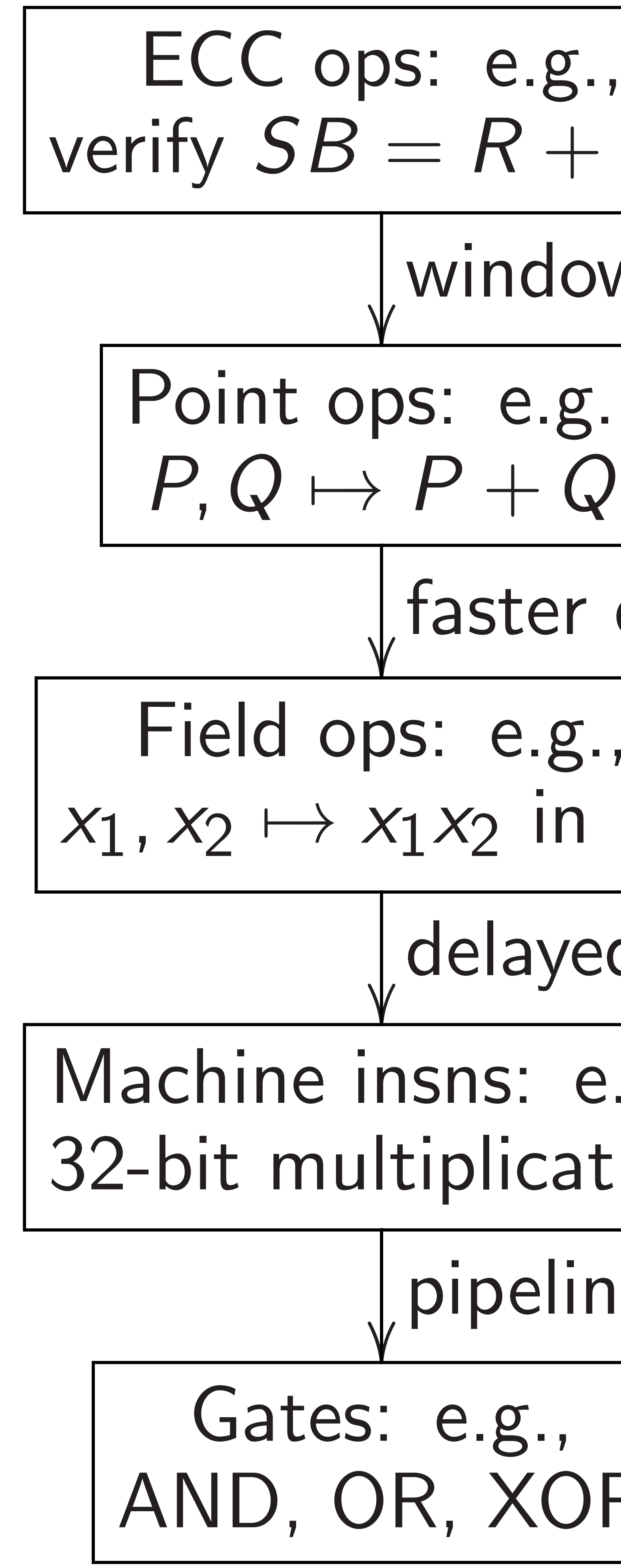
For each of these operations,
on each of these curves,
on each of these CPUs:

Simplest implementations
are much, much, much slower.

Questions in algorithm design
and software engineering:
How to build the fastest software
on, e.g., an ARM Cortex-A8 for
Ed25519 signature verification?

Answers feed back into crypto
design: e.g., choosing fast curves.

Several levels to o



):

ll

256,

19.

56

),

19.

e:

-256,

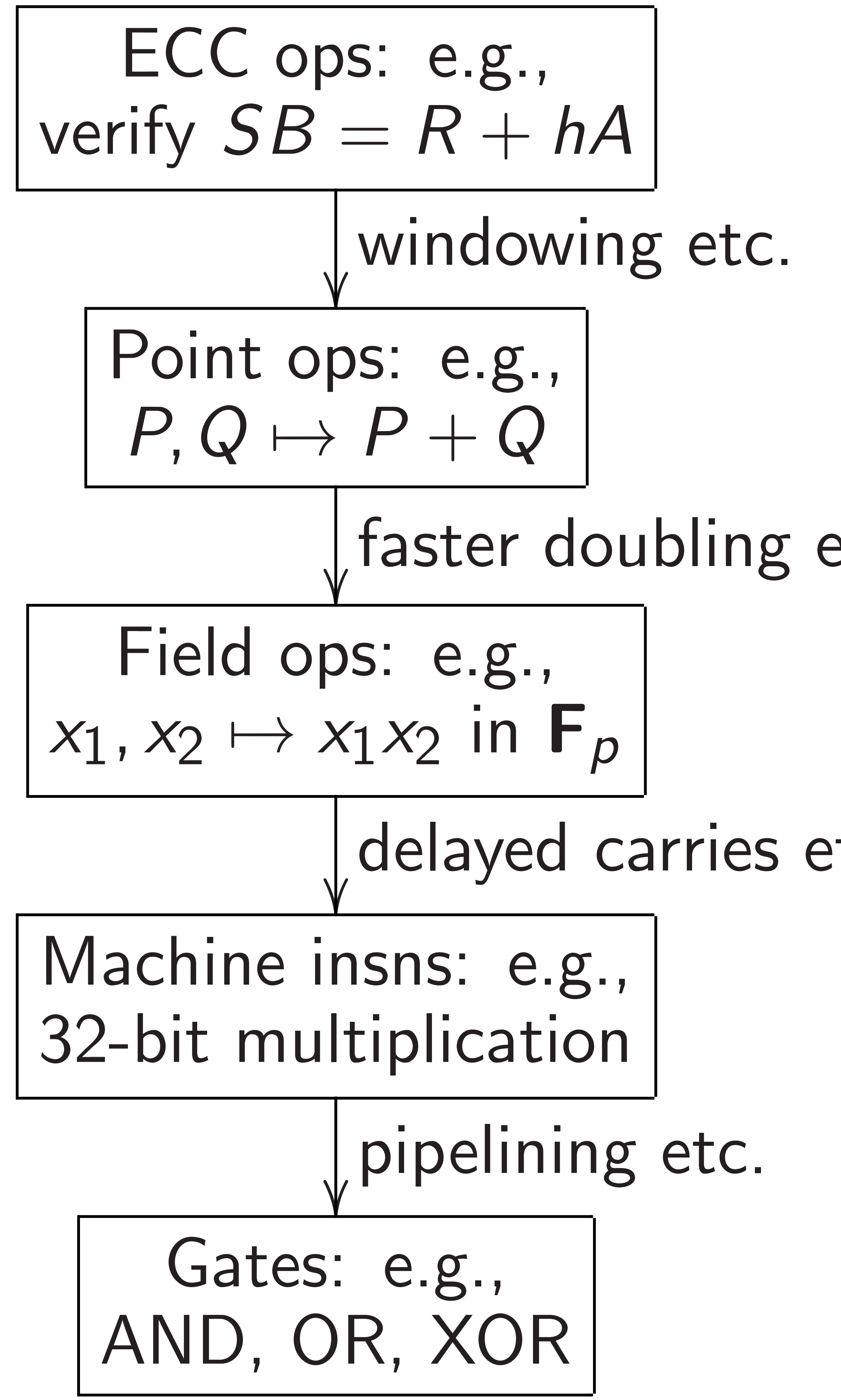
For each of these operations,
on each of these curves,
on each of these CPUs:

Simplest implementations
are much, much, much slower.

Questions in algorithm design
and software engineering:
How to build the fastest software
on, e.g., an ARM Cortex-A8 for
Ed25519 signature verification?

Answers feed back into crypto
design: e.g., choosing fast curves.

Several levels to optimize:



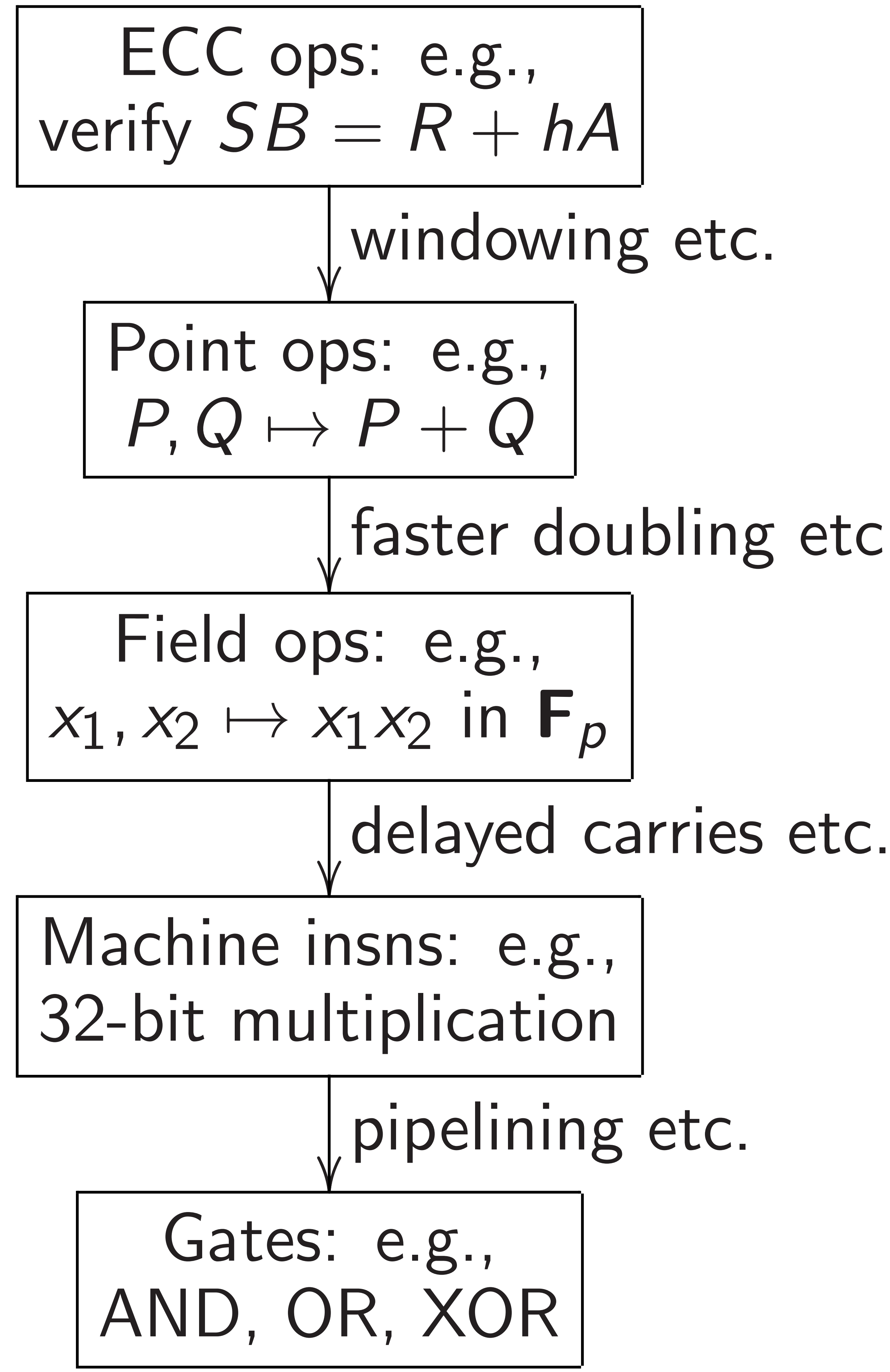
For each of these operations,
on each of these curves,
on each of these CPUs:

Simplest implementations
are much, much, much slower.

Questions in algorithm design
and software engineering:
How to build the fastest software
on, e.g., an ARM Cortex-A8 for
Ed25519 signature verification?

Answers feed back into crypto
design: e.g., choosing fast curves.

Several levels to optimize:



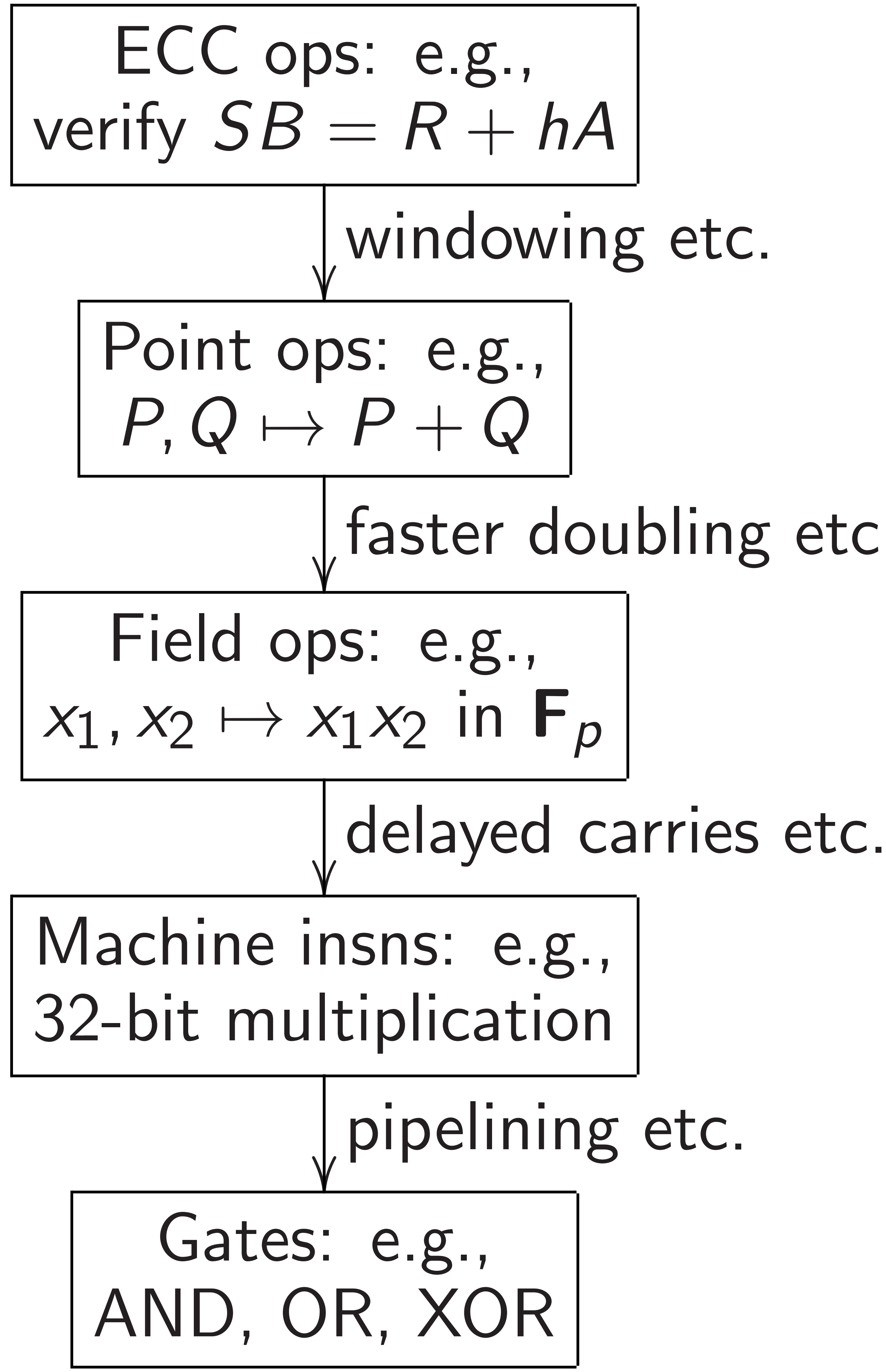
n of these operations,
of these curves,
of these CPUs:

t implementations
h, much, much slower.

ns in algorithm design
ware engineering:
build the fastest software
an ARM Cortex-A8 for
9 signature verification?

feed back into crypto
e.g., choosing fast curves.

Several levels to optimize:



Single-sc

Fundame
$n, P \mapsto$

Input n
$\{0, 1, \ldots$

Input P

Will buil
using ad
and subt

Later wi
double-s
$m, P, n,$

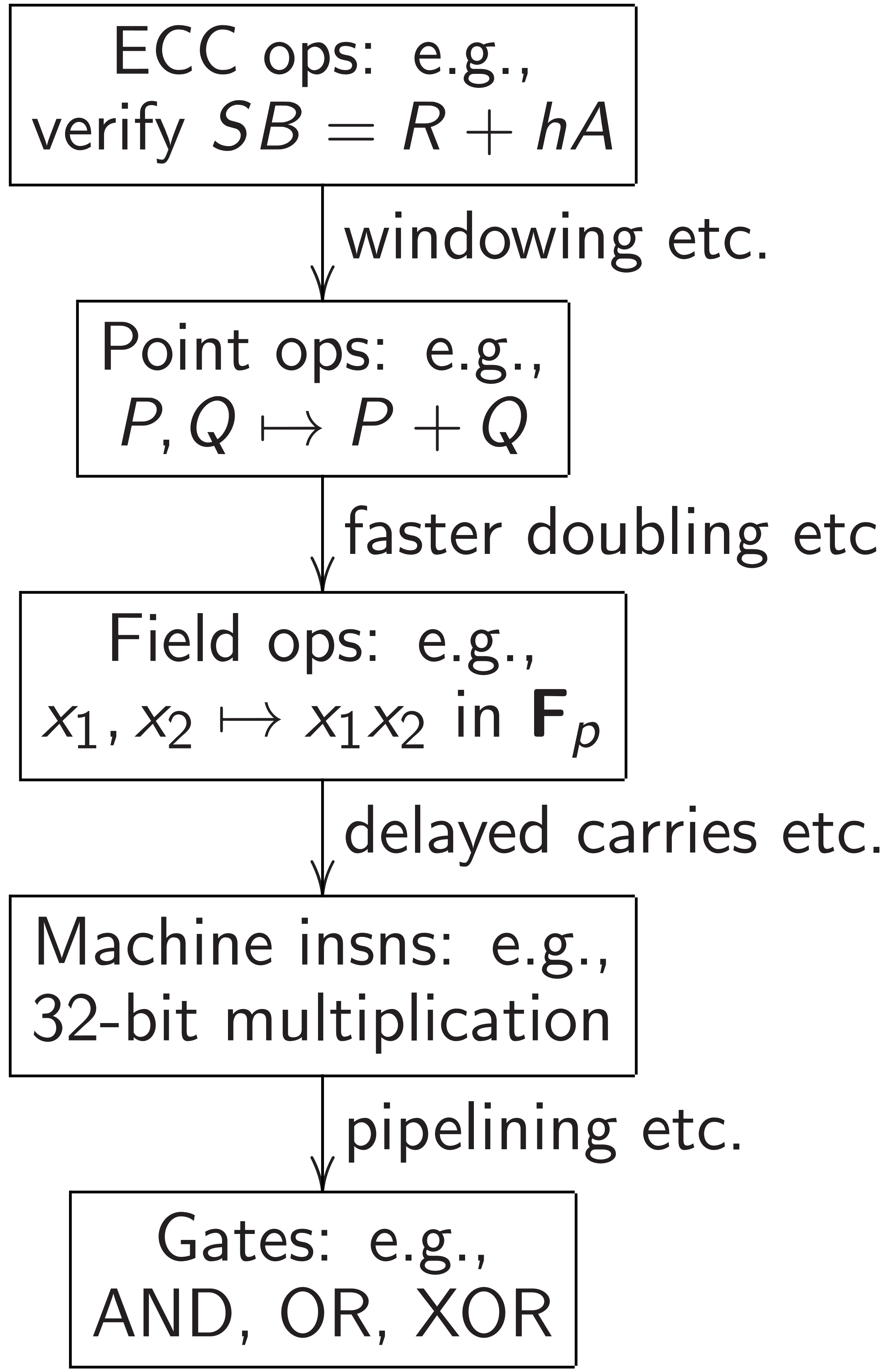
operations,
curves,
CPUs:

ntations
much slower.

ithm design
neering:
fastest software
Cortex-A8 for
e verification?

into crypto
sing fast curves.

Several levels to optimize:



Single-scalar multi

Fundamental ECC
$n, P \mapsto nP$.

Input n is integer
$\{0, 1, \ldots, 2^{256} - 1$

Input P is point o

Will build $n, P \mapsto$
using additions $P,$
and subtractions F

Later will also lool
double-scalar mult
$m, P, n, Q \mapsto mP$ -

s,

er.

gn

tware

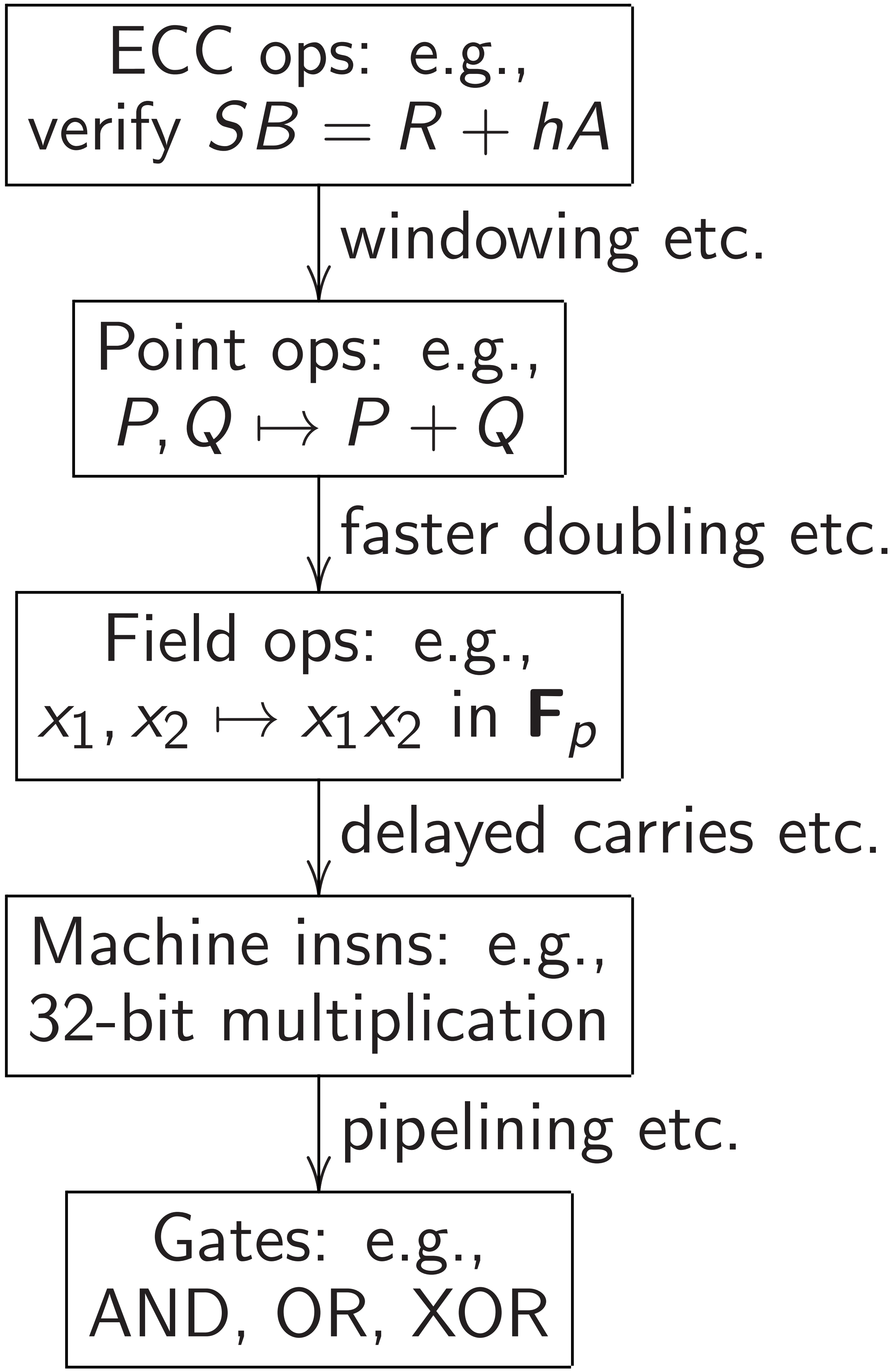
3 for

on?

to

urves.

Several levels to optimize:



Single-scalar multiplication

Fundamental ECC operation
$n, P \mapsto nP$.

Input n is integer in, e.g.,
$\{0, 1, \ldots, 2^{256} - 1\}$.

Input P is point on elliptic c

Will build $n, P \mapsto nP$
using additions $P, Q \mapsto P +$
and subtractions $P, Q \mapsto P -$

Later will also look at
double-scalar multiplication
$m, P, n, Q \mapsto mP + nQ$.

Several levels to optimize:

ECC ops: e.g., verify $SB = R + hA$

↓ windowing etc.

Point ops: e.g., $P, Q \mapsto P + Q$

↓ faster doubling etc.

Field ops: e.g., $x_1, x_2 \mapsto x_1 x_2$ in $\mathbf{F}_p$

↓ delayed carries etc.

Machine insns: e.g., 32-bit multiplication

↓ pipelining etc.

Gates: e.g., AND, OR, XOR

Single-scalar multiplication

Fundamental ECC operation: $n, P \mapsto nP$.

Input n is integer in, e.g., $\{0, 1, \ldots, 2^{256} - 1\}$.

Input P is point on elliptic curve.

Will build $n, P \mapsto nP$ using additions $P, Q \mapsto P + Q$ and subtractions $P, Q \mapsto P - Q$.

Later will also look at double-scalar multiplication $m, P, n, Q \mapsto mP + nQ$.

levels to optimize:

ops: e.g.,
$\ldots B = R + hA$

↓ windowing etc.

ops: e.g.,
$\mapsto P + Q$

↓ faster doubling etc.

ops: e.g.,
$\to x_1 x_2$ in $\mathbf{F}_p$

↓ delayed carries etc.

e insns: e.g.,
multiplication

↓ pipelining etc.

tes: e.g.,
, OR, XOR

Single-scalar multiplication

Fundamental ECC operation:
$n, P \mapsto nP$.

Input n is integer in, e.g.,
$\{0, 1, \ldots, 2^{256} - 1\}$.

Input P is point on elliptic curve.

Will build $n, P \mapsto nP$
using additions $P, Q \mapsto P + Q$
and subtractions $P, Q \mapsto P - Q$.

Later will also look at
double-scalar multiplication
$m, P, n, Q \mapsto mP + nQ$.

Left-to-r

```
def scal
  if n =
  if n =
  R = sc
  R = R
  if n %
  return
```

Two Pyt

- n//2 i
- Recurs
  See sy

ptimize:

hA

ving etc.

,

doubling etc.

$\mathbf{F}_p$

d carries etc.

g.,
ion

ing etc.

R

Single-scalar multiplication

Fundamental ECC operation:
$n, P \mapsto nP$.

Input n is integer in, e.g.,
$\{0, 1, \ldots, 2^{256} - 1\}$.

Input P is point on elliptic curve.

Will build $n, P \mapsto nP$
using additions $P, Q \mapsto P + Q$
and subtractions $P, Q \mapsto P - Q$.

Later will also look at
double-scalar multiplication
$m, P, n, Q \mapsto mP + nQ$.

Left-to-right binar

```
def scalarmult(n
  if n == 0: ret
  if n == 1: ret
  R = scalarmult
  R = R + R
  if n % 2: R =
  return R
```

Two Python notes

- `n//2` in Python
- Recursion depth
  See `sys.setrec`

etc.

tc.

Single-scalar multiplication

Fundamental ECC operation:
$n, P \mapsto nP$.

Input n is integer in, e.g.,
$\{0, 1, \ldots, 2^{256} - 1\}$.

Input P is point on elliptic curve.

Will build $n, P \mapsto nP$
using additions $P, Q \mapsto P + Q$
and subtractions $P, Q \mapsto P - Q$.

Later will also look at
double-scalar multiplication
$m, P, n, Q \mapsto mP + nQ$.

Left-to-right binary method

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  R = scalarmult(n//2,P)
  R = R + R
  if n % 2: R = R + P
  return R
```

Two Python notes:

- n//2 in Python means ⌊n/
- Recursion depth is limited
  See sys.setrecursionli

Single-scalar multiplication

Fundamental ECC operation:
$n, P \mapsto nP$.

Input n is integer in, e.g.,
$\{0, 1, \ldots, 2^{256} - 1\}$.

Input P is point on elliptic curve.

Will build $n, P \mapsto nP$
using additions $P, Q \mapsto P + Q$
and subtractions $P, Q \mapsto P - Q$.

Later will also look at
double-scalar multiplication
$m, P, n, Q \mapsto mP + nQ$.

Left-to-right binary method

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  R = scalarmult(n//2,P)
  R = R + R
  if n % 2: R = R + P
  return R
```

Two Python notes:

- `n//2` in Python means $\lfloor \texttt{n}/2 \rfloor$.
- Recursion depth is limited.
  See `sys.setrecursionlimit`.

calar multiplication

ental ECC operation:
nP.

is integer in, e.g.,
$\ldots, 2^{256} - 1\}$.

is point on elliptic curve.

d $n, P \mapsto nP$
ditions $P, Q \mapsto P + Q$
tractions $P, Q \mapsto P - Q$.

ll also look at
calar multiplication
$Q \mapsto mP + nQ$.

Left-to-right binary method

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  R = scalarmult(n//2,P)
  R = R + R
  if n % 2: R = R + P
  return R
```

Two Python notes:

- `n//2` in Python means $\lfloor \mathtt{n}/2 \rfloor$.
- Recursion depth is limited. See `sys.setrecursionlimit`.

This rec

- $2\left(\frac{n}{2}P\right.$

  e.g. 20

- $2\left(\frac{n-}{2}\right.$

  e.g. 21

Base cas
$0P = 0$.
$1P = P$.

Assumin

Otherwis

plication

operation:

in, e.g.,

}.

n elliptic curve.

nP

$Q \mapsto P + Q$

$P, Q \mapsto P - Q$.

k at

tiplication

$+ nQ$.

Left-to-right binary method

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  R = scalarmult(n//2,P)
  R = R + R
  if n % 2: R = R + P
  return R
```

Two Python notes:

- `n//2` in Python means $\lfloor \mathtt{n}/2 \rfloor$.
- Recursion depth is limited. See `sys.setrecursionlimit`.

This recursion con

- $2\left(\frac{n}{2}P\right)$ if $n \in 2$

  e.g. $20P = 2 \cdot 1$

- $2\left(\frac{n-1}{2}P\right) +$

  e.g. $21P = 2 \cdot 1$

Base cases in recu

$0P = 0$. For Edwa

$1P = P$. Could or

Assuming $n \geq 0$ fo

Otherwise use nP

n:

curve.

Q

− Q.

Left-to-right binary method

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  R = scalarmult(n//2,P)
  R = R + R
  if n % 2: R = R + P
  return R
```

Two Python notes:

- `n//2` in Python means $\lfloor n/2 \rfloor$.
- Recursion depth is limited.
  See `sys.setrecursionlimit`.

This recursion computes nP

- $2\left(\frac{n}{2}P\right)$ if $n \in 2\mathbf{Z}$.
  e.g. $20P = 2 \cdot 10P$.
- $2\left(\frac{n-1}{2}P\right) + P$ if $n \in 1$
  e.g. $21P = 2 \cdot 10P + P$.

Base cases in recursion:

$0P = 0$. For Edwards: $0 = ($

$1P = P$. Could omit this ca

Assuming $n \geq 0$ for simplicit

Otherwise use $nP = -(-n)$

Left-to-right binary method

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  R = scalarmult(n//2,P)
  R = R + R
  if n % 2: R = R + P
  return R
```

Two Python notes:

- `n//2` in Python means $\lfloor \mathtt{n}/2 \rfloor$.
- Recursion depth is limited. See `sys.setrecursionlimit`.

This recursion computes nP as

- $2\left(\frac{n}{2}P\right)$ if $n \in 2\mathbf{Z}$.
  e.g. $20P = 2 \cdot 10P$.
- $2\left(\frac{n-1}{2}P\right) + P$ if $n \in 1 + 2\mathbf{Z}$.
  e.g. $21P = 2 \cdot 10P + P$.

Base cases in recursion:
$0P = 0$. For Edwards: $0 = (0, 1)$.
$1P = P$. Could omit this case.

Assuming $n \geq 0$ for simplicity.
Otherwise use $nP = -(-n)P$.

ight binary method

```
larmult(n,P):
== 0: return 0
== 1: return P
calarmult(n//2,P)
 + R
% 2: R = R + P
n R
```

thon notes:

n Python means $\lfloor n/2 \rfloor$.

sion depth is limited.

ys.setrecursionlimit.

This recursion computes nP as

- $2\left(\frac{n}{2}P\right)$ if $n \in 2\mathbf{Z}$.

  e.g. $20P = 2 \cdot 10P$.

- $2\left(\frac{n-1}{2}P\right) + P$ if $n \in 1 + 2\mathbf{Z}$.

  e.g. $21P = 2 \cdot 10P + P$.

Base cases in recursion:

$0P = 0$. For Edwards: $0 = (0, 1)$.

$1P = P$. Could omit this case.

Assuming $n \geq 0$ for simplicity.

Otherwise use $nP = -(-n)P$.

If $0 \leq n$

this algo

$\leq 2b - 2$

$\leq b - 1$

$\leq b - 1$

Example

$31P = 2$

$31 = (11$

4 doubli

Average

$35P = 2$

$35 = (10$

5 doubli

y method

```
,P):
urn 0
urn P
(n//2,P)

R + P
```

s:

means $\lfloor n/2 \rfloor$.

is limited.

`cursionlimit`.

This recursion computes nP as

- $2\left(\frac{n}{2}P\right)$ if $n \in 2\mathbf{Z}$.
  e.g. $20P = 2 \cdot 10P$.

- $2\left(\frac{n-1}{2}P\right) + P$ if $n \in 1 + 2\mathbf{Z}$.
  e.g. $21P = 2 \cdot 10P + P$.

Base cases in recursion:
$0P = 0$. For Edwards: $0 = (0, 1)$.
$1P = P$. Could omit this case.

Assuming $n \geq 0$ for simplicity.
Otherwise use $nP = -(-n)P$.

If $0 \leq n < 2^b$ the
this algorithm use
$\leq 2b - 2$ addition
$\leq b - 1$ doublings
$\leq b - 1$ additions

Example of worst
$31P = 2(2(2(2P+$
$31 = (11111)_2$; b
4 doublings; 4 mo

Average case is be
$35P = 2(2(2(2(2P$
$35 = (100011)_2$; b
5 doublings; 2 add

$/2\rfloor$.

.

imit.

This recursion computes nP as

- $2\left(\frac{n}{2}P\right)$ if $n \in 2\mathbf{Z}$.
  e.g. $20P = 2 \cdot 10P$.

- $2\left(\frac{n-1}{2}P\right) + P$ if $n \in 1 + 2\mathbf{Z}$.
  e.g. $21P = 2 \cdot 10P + P$.

Base cases in recursion:
$0P = 0$. For Edwards: $0 = (0, 1)$.
$1P = P$. Could omit this case.

Assuming $n \geq 0$ for simplicity.
Otherwise use $nP = -(-n)P$.

If $0 \leq n < 2^b$ then
this algorithm uses
$\leq 2b - 2$ additions: specifica
$\leq b - 1$ doublings and
$\leq b - 1$ additions of P.

Example of worst case:
$31P = 2(2(2(2P+P)+P)+$
$31 = (11111)_2$; $b = 5$;
4 doublings; 4 more additior

Average case is better: e.g.
$35P = 2(2(2(2(2P))) + P)$
$35 = (100011)_2$; $b = 6$;
5 doublings; 2 additions.

This recursion computes nP as

- $2\left(\frac{n}{2}P\right)$ if $n \in 2\mathbf{Z}$.
  e.g. $20P = 2 \cdot 10P$.
- $2\left(\frac{n-1}{2}P\right) + P$ if $n \in 1 + 2\mathbf{Z}$.
  e.g. $21P = 2 \cdot 10P + P$.

Base cases in recursion:
$0P = 0$. For Edwards: $0 = (0, 1)$.
$1P = P$. Could omit this case.

Assuming $n \geq 0$ for simplicity.
Otherwise use $nP = -(-n)P$.

If $0 \leq n < 2^b$ then
this algorithm uses
$\leq 2b - 2$ additions: specifically
$\leq b - 1$ doublings and
$\leq b - 1$ additions of P.

Example of worst case:
$31P = 2(2(2(2P+P)+P)+P)+P$.
$31 = (11111)_2$; $b = 5$;
4 doublings; 4 more additions.

Average case is better: e.g.
$35P = 2(2(2(2(2P))) + P) + P$.
$35 = (100011)_2$; $b = 6$;
5 doublings; 2 additions.

ursion computes nP as

$P\Big)$ if $n \in 2\mathbf{Z}$.

$0P = 2 \cdot 10P$.

$\frac{-1}{2} P\Big) + P$ if $n \in 1 + 2\mathbf{Z}$.

$1P = 2 \cdot 10P + P$.

ses in recursion:

For Edwards: $0 = (0, 1)$.

. Could omit this case.

g $n \geq 0$ for simplicity.

se use $nP = -(-n)P$.

If $0 \leq n < 2^b$ then
this algorithm uses
$\leq 2b - 2$ additions: specifically
$\leq b - 1$ doublings and
$\leq b - 1$ additions of P.

Example of worst case:
$31P = 2(2(2(2P+P)+P)+P)+P$.
$31 = (11111)_2$; $b = 5$;
4 doublings; 4 more additions.

Average case is better: e.g.
$35P = 2(2(2(2(2P))) + P) + P$.
$35 = (100011)_2$; $b = 6$;
5 doublings; 2 additions.

Non-adj

```
def sca
  if n =
  if n =
  if n %
    R =
    R =
    ret
  if n %
    R =
    R =
    ret
  R = s
  retur
```

nputes nP as

$2\mathbf{Z}$.

$0P$.

P if $n \in 1 + 2\mathbf{Z}$.

$0P + P$.

rsion:

ards: $0 = (0, 1)$.

mit this case.

or simplicity.

$= -(-n)P$.

If $0 \le n < 2^b$ then
this algorithm uses
$\le 2b - 2$ additions: specifically
$\le b - 1$ doublings and
$\le b - 1$ additions of P.

Example of worst case:
$31P = 2(2(2(2P{+}P){+}P){+}P){+}P$.
$31 = (11111)_2$; $b = 5$;
4 doublings; 4 more additions.

Average case is better: e.g.
$35P = 2(2(2(2(2P))) + P) + P$.
$35 = (100011)_2$; $b = 6$;
5 doublings; 2 additions.

Non-adjacent form

```
def scalarmult(n
  if n == 0: ret
  if n == 1: ret
  if n % 4 == 1:
    R = scalarmu
    R = R + R
    return (R +
  if n % 4 == 3:
    R = scalarmu
    R = R + R
    return (R +
  R = scalarmult
  return R + R
```

as

$+ 2\mathbf{Z}$.

$(0, 1)$.

se.

ty.

P.

If $0 \le n < 2^b$ then this algorithm uses $\le 2b - 2$ additions: specifically $\le b - 1$ doublings and $\le b - 1$ additions of P.

Example of worst case:
$31P = 2(2(2(2P{+}P){+}P){+}P){+}P$.
$31 = (11111)_2$; $b = 5$;
4 doublings; 4 more additions.

Average case is better: e.g.
$35P = 2(2(2(2(2P))) + P) + P$.
$35 = (100011)_2$; $b = 6$;
5 doublings; 2 additions.

<u>Non-adjacent form (NAF)</u>

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  if n % 4 == 1:
    R = scalarmult((n-1)/
    R = R + R
    return (R + R) + P
  if n % 4 == 3:
    R = scalarmult((n+1)/
    R = R + R
    return (R + R) - P
  R = scalarmult(n/2,P)
  return R + R
```

If $0 \leq n < 2^b$ then
this algorithm uses
$\leq 2b - 2$ additions: specifically
$\leq b - 1$ doublings and
$\leq b - 1$ additions of P.

Example of worst case:
$31P = 2(2(2(2P+P)+P)+P)+P$.
$31 = (11111)_2$; $b = 5$;
4 doublings; 4 more additions.

Average case is better: e.g.
$35P = 2(2(2(2(2P))) + P) + P$.
$35 = (100011)_2$; $b = 6$;
5 doublings; 2 additions.

Non-adjacent form (NAF)

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  if n % 4 == 1:
    R = scalarmult((n-1)/4,P)
    R = R + R
    return (R + R) + P
  if n % 4 == 3:
    R = scalarmult((n+1)/4,P)
    R = R + R
    return (R + R) - P
  R = scalarmult(n/2,P)
  return R + R
```

$< 2^b$ then
orithm uses
2 additions: specifically
doublings and
additions of P.

e of worst case:
$2(2(2(2P+P)+P)+P)+P$.
$1111)_2$; $b = 5$;
ngs; 4 more additions.

case is better: e.g.
$2(2(2(2(2P))) + P) + P$.
$00011)_2$; $b = 6$;
ngs; 2 additions.

Non-adjacent form (NAF)

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  if n % 4 == 1:
    R = scalarmult((n-1)/4,P)
    R = R + R
    return (R + R) + P
  if n % 4 == 3:
    R = scalarmult((n+1)/4,P)
    R = R + R
    return (R + R) - P
  R = scalarmult(n/2,P)
  return R + R
```

Subtract
is as che
NAF tak

$31P = 2$
$31 = (10$

$35P = 2$
$35 = (10$

“Non-ad
separate

Worst ca
plus $\approx b$
On avera

n
s
: specifically
and
of P.

case:
$P)+P)+P)+P$.
$= 5$;
re additions.

etter: e.g.
$P))) + P) + P$.
$b = 6$;
litions.

Non-adjacent form (NAF)

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  if n % 4 == 1:
    R = scalarmult((n-1)/4,P)
    R = R + R
    return (R + R) + P
  if n % 4 == 3:
    R = scalarmult((n+1)/4,P)
    R = R + R
    return (R + R) - P
  R = scalarmult(n/2,P)
  return R + R
```


Subtraction on the
is as cheap as add
NAF takes advant

$31P = 2(2(2(2(2P$
$31 = (10000\bar{1})_2$; $\bar{1}$

$35P = 2(2(2(2(2P$
$35 = (10010\bar{1})_2$.

“Non-adjacent”: ±
separated by $\geq$2 d

Worst case: $\approx b$ d
plus $\approx b/2$ additio
On average $\approx b/3$

ally

$P)+P$.

ns.

$+\ P$.

Non-adjacent form (NAF)

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  if n % 4 == 1:
    R = scalarmult((n-1)/4,P)
    R = R + R
    return (R + R) + P
  if n % 4 == 3:
    R = scalarmult((n+1)/4,P)
    R = R + R
    return (R + R) - P
  R = scalarmult(n/2,P)
  return R + R
```


Subtraction on the curve
is as cheap as addition.
NAF takes advantage of this

$31P = 2(2(2(2(2P)))) - P$.
$31 = (10000\bar{1})_2$; $\bar{1}$ denotes

$35P = 2(2(2(2(2P)) + P))$
$35 = (10010\bar{1})_2$.

"Non-adjacent": $\pm P$ ops ar
separated by $\geq$2 doublings.

Worst case: $\approx b$ doublings
plus $\approx b/2$ additions of $\pm P$.
On average $\approx b/3$ additions.

Non-adjacent form (NAF)

```
def scalarmult(n,P):
  if n == 0: return 0
  if n == 1: return P
  if n % 4 == 1:
    R = scalarmult((n-1)/4,P)
    R = R + R
    return (R + R) + P
  if n % 4 == 3:
    R = scalarmult((n+1)/4,P)
    R = R + R
    return (R + R) - P
  R = scalarmult(n/2,P)
  return R + R
```


Subtraction on the curve is as cheap as addition. NAF takes advantage of this.

$31P = 2(2(2(2(2P)))) - P$.
$31 = (10000\bar{1})_2$; $\bar{1}$ denotes -1.

$35P = 2(2(2(2(2P)) + P)) - P$.
$35 = (10010\bar{1})_2$.

"Non-adjacent": $\pm P$ ops are separated by ≥ 2 doublings.

Worst case: $\approx b$ doublings plus $\approx b/2$ additions of $\pm P$.
On average $\approx b/3$ additions.

acent form (NAF)

```
larmult(n,P):
== 0: return 0
== 1: return P
% 4 == 1:
 scalarmult((n-1)/4,P)
 R + R
urn (R + R) + P
% 4 == 3:
 scalarmult((n+1)/4,P)
 R + R
urn (R + R) - P
calarmult(n/2,P)
n R + R
```


Subtraction on the curve is as cheap as addition. NAF takes advantage of this.

$31P = 2(2(2(2(2P)))) - P$.
$31 = (10000\bar{1})_2$; $\bar{1}$ denotes -1.

$35P = 2(2(2(2(2P)) + P)) - P$.
$35 = (10010\bar{1})_2$.

"Non-adjacent": $\pm P$ ops are separated by ≥ 2 doublings.

Worst case: $\approx b$ doublings plus $\approx b/2$ additions of $\pm P$.
On average $\approx b/3$ additions.

Width-2

```
def wind
  if n =
  if n =
  if n =
  if n %
    R =
    R =
    R =
    retu
  if n %
    R =
    R =
    R =
    retu
```

n (NAF)

```
,P):
urn 0
urn P

lt((n-1)/4,P)

R) + P

lt((n+1)/4,P)

R) - P
(n/2,P)
```


Subtraction on the curve
is as cheap as addition.
NAF takes advantage of this.

$31P = 2(2(2(2(2P)))) - P$.
$31 = (10000\bar{1})_2$; $\bar{1}$ denotes -1.

$35P = 2(2(2(2(2P)) + P)) - P$.
$35 = (10010\bar{1})_2$.

"Non-adjacent": $\pm P$ ops are separated by ≥ 2 doublings.

Worst case: $\approx b$ doublings plus $\approx b/2$ additions of $\pm P$.
On average $\approx b/3$ additions.

Width-2 signed sli

```
def window2(n,P,
  if n == 0: ret
  if n == 1: ret
  if n == 3: ret
  if n % 8 == 1:
    R = window2(
    R = R + R
    R = R + R
    return (R +
  if n % 8 == 3:
    R = window2(
    R = R + R
    R = R + R
    return (R +
```

/4,P)

/4,P)

Subtraction on the curve is as cheap as addition. NAF takes advantage of this.

$31P = 2(2(2(2(2P)))) - P$.
$31 = (10000\bar{1})_2$; $\bar{1}$ denotes -1.

$35P = 2(2(2(2(2P)) + P)) - P$.
$35 = (10010\bar{1})_2$.

"Non-adjacent": $\pm P$ ops are separated by ≥ 2 doublings.

Worst case: $\approx b$ doublings plus $\approx b/2$ additions of $\pm P$. On average $\approx b/3$ additions.

Width-2 signed sliding windo

```
def window2(n,P,P3):
  if n == 0: return 0
  if n == 1: return P
  if n == 3: return P3
  if n % 8 == 1:
    R = window2((n-1)/8,P
    R = R + R
    R = R + R
    return (R + R) + P
  if n % 8 == 3:
    R = window2((n-3)/8,P
    R = R + R
    R = R + R
    return (R + R) + P3
```

Subtraction on the curve is as cheap as addition. NAF takes advantage of this.

$31P = 2(2(2(2(2P)))) - P$.
$31 = (10000\bar{1})_2$; $\bar{1}$ denotes -1.

$35P = 2(2(2(2(2P)) + P)) - P$.
$35 = (10010\bar{1})_2$.

"Non-adjacent": $\pm P$ ops are separated by ≥ 2 doublings.

Worst case: $\approx b$ doublings plus $\approx b/2$ additions of $\pm P$. On average $\approx b/3$ additions.

Width-2 signed sliding windows

```
def window2(n,P,P3):
  if n == 0: return 0
  if n == 1: return P
  if n == 3: return P3
  if n % 8 == 1:
    R = window2((n-1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) + P
  if n % 8 == 3:
    R = window2((n-3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) + P3
```

tion on the curve
eap as addition.
kes advantage of this.

$2(2(2(2(2P)))) - P$.
$00001\bar{1})_2$; $\bar{1}$ denotes -1.

$2(2(2(2(2P)) + P)) - P$.
$00101\bar{1})_2$.

ljacent": $\pm P$ ops are
d by ≥ 2 doublings.

ase: $\approx b$ doublings
$/2$ additions of $\pm P$.
age $\approx b/3$ additions.

Width-2 signed sliding windows

```
def window2(n,P,P3):
  if n == 0: return 0
  if n == 1: return P
  if n == 3: return P3
  if n % 8 == 1:
    R = window2((n-1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) + P
  if n % 8 == 3:
    R = window2((n-3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) + P3
```

```
  if n %
    R =
    R =
    R =
    retu
  if n %
    R =
    R =
    R =
    retu
  R = wi
  return

def scal
  return
```

e curve
ition.
age of this.

$P)))) - P$.
denotes -1.

$P)) + P)) - P$.

$\pm P$ ops are
loublings.

oublings
ns of $\pm P$.
additions.

Width-2 signed sliding windows

```
def window2(n,P,P3):
  if n == 0: return 0
  if n == 1: return P
  if n == 3: return P3
  if n % 8 == 1:
    R = window2((n-1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) + P
  if n % 8 == 3:
    R = window2((n-3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) + P3
```

```
  if n % 8 == 5:
    R = window2(
    R = R + R
    R = R + R
    return (R +
  if n % 8 == 7:
    R = window2(
    R = R + R
    R = R + R
    return (R +
  R = window2(n/
  return R + R

def scalarmult(n
  return window2
```

s.

−1.

− *P*.

e

Width-2 signed sliding windows

```
def window2(n,P,P3):
  if n == 0: return 0
  if n == 1: return P
  if n == 3: return P3
  if n % 8 == 1:
    R = window2((n-1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) + P
  if n % 8 == 3:
    R = window2((n-3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) + P3
```



```
  if n % 8 == 5:
    R = window2((n+3)/8,P
    R = R + R
    R = R + R
    return (R + R) - P3
  if n % 8 == 7:
    R = window2((n+1)/8,P
    R = R + R
    R = R + R
    return (R + R) - P
  R = window2(n/2,P,P3)
  return R + R

def scalarmult(n,P):
  return window2(n,P,P+P+
```

Width-2 signed sliding windows

```
def window2(n,P,P3):
  if n == 0: return 0
  if n == 1: return P
  if n == 3: return P3
  if n % 8 == 1:
    R = window2((n-1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) + P
  if n % 8 == 3:
    R = window2((n-3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) + P3
```



```
  if n % 8 == 5:
    R = window2((n+3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P3
  if n % 8 == 7:
    R = window2((n+1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P
  R = window2(n/2,P,P3)
  return R + R

def scalarmult(n,P):
  return window2(n,P,P+P+P)
```

signed sliding windows

```
dow2(n,P,P3):
== 0: return 0
== 1: return P
== 3: return P3
% 8 == 1:
 window2((n-1)/8,P,P3)
 R + R
 R + R
urn (R + R) + P
% 8 == 3:
 window2((n-3)/8,P,P3)
 R + R
 R + R
urn (R + R) + P3
```



```
  if n % 8 == 5:
    R = window2((n+3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P3
  if n % 8 == 7:
    R = window2((n+1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P
  R = window2(n/2,P,P3)
  return R + R

def scalarmult(n,P):
  return window2(n,P,P+P+P)
```

Worst ca

$\approx b/3$ ac

On avera

ding windows

```
P3):
urn 0
urn P
urn P3

(n-1)/8,P,P3)

R) + P

(n-3)/8,P,P3)

R) + P3
```



```
  if n % 8 == 5:
    R = window2((n+3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P3
  if n % 8 == 7:
    R = window2((n+1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P
  R = window2(n/2,P,P3)
  return R + R

def scalarmult(n,P):
  return window2(n,P,P+P+P)
```

Worst case: $\approx b$ d

$\approx b/3$ additions of

On average $\approx b/4$

ows

,P3)

,P3)


```
  if n % 8 == 5:
    R = window2((n+3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P3
  if n % 8 == 7:
    R = window2((n+1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P
  R = window2(n/2,P,P3)
  return R + R

def scalarmult(n,P):
  return window2(n,P,P+P+P)
```

Worst case: $\approx b$ doublings p

$\approx b/3$ additions of $\pm P$ or $\pm$

On average $\approx b/4$ additions.

```
  if n % 8 == 5:
    R = window2((n+3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P3
  if n % 8 == 7:
    R = window2((n+1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P
  R = window2(n/2,P,P3)
  return R + R

def scalarmult(n,P):
  return window2(n,P,P+P+P)
```


Worst case: $\approx b$ doublings plus $\approx b/3$ additions of $\pm P$ or $\pm 3P$.

On average $\approx b/4$ additions.

```
  if n % 8 == 5:
    R = window2((n+3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P3
  if n % 8 == 7:
    R = window2((n+1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P
  R = window2(n/2,P,P3)
  return R + R

def scalarmult(n,P):
  return window2(n,P,P+P+P)
```


Worst case: $\approx b$ doublings plus $\approx b/3$ additions of $\pm P$ or $\pm 3P$.
On average $\approx b/4$ additions.

Width-3 signed sliding windows:
Precompute $P, 3P, 5P, 7P$.
On average $\approx b/5$ additions.

```
  if n % 8 == 5:
    R = window2((n+3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P3
  if n % 8 == 7:
    R = window2((n+1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P
  R = window2(n/2,P,P3)
  return R + R

def scalarmult(n,P):
  return window2(n,P,P+P+P)
```


Worst case: $\approx b$ doublings plus $\approx b/3$ additions of $\pm P$ or $\pm 3P$. On average $\approx b/4$ additions.

Width-3 signed sliding windows: Precompute $P, 3P, 5P, 7P$. On average $\approx b/5$ additions.

Width 4: Precompute $P, 3P, 5P, 7P, 9P, 11P, 13P, 15P$. On average $\approx b/6$ additions.

```
  if n % 8 == 5:
    R = window2((n+3)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P3
  if n % 8 == 7:
    R = window2((n+1)/8,P,P3)
    R = R + R
    R = R + R
    return (R + R) - P
  R = window2(n/2,P,P3)
  return R + R

def scalarmult(n,P):
  return window2(n,P,P+P+P)
```


Worst case: $\approx b$ doublings plus $\approx b/3$ additions of $\pm P$ or $\pm 3P$.
On average $\approx b/4$ additions.

Width-3 signed sliding windows:
Precompute $P, 3P, 5P, 7P$.
On average $\approx b/5$ additions.

Width 4: Precompute
$P, 3P, 5P, 7P, 9P, 11P, 13P, 15P$.
On average $\approx b/6$ additions.

Cost of precomputation eventually outweighs savings.
Optimal: $\approx b$ doublings plus roughly $b/\lg b$ additions.

```
% 8 == 5:
 window2((n+3)/8,P,P3)
 R + R
 R + R
urn (R + R) - P3
% 8 == 7:
 window2((n+1)/8,P,P3)
 R + R
 R + R
urn (R + R) - P
indow2(n/2,P,P3)
n R + R

larmult(n,P):
n window2(n,P,P+P+P)
```


Worst case: $\approx b$ doublings plus $\approx b/3$ additions of $\pm P$ or $\pm 3P$. On average $\approx b/4$ additions.

Width-3 signed sliding windows: Precompute $P, 3P, 5P, 7P$. On average $\approx b/5$ additions.

Width 4: Precompute $P, 3P, 5P, 7P, 9P, 11P, 13P, 15P$. On average $\approx b/6$ additions.

Cost of precomputation eventually outweighs savings. Optimal: $\approx b$ doublings plus roughly $b/\lg b$ additions.

Double-s

Want to
$m, P, n,$ (

e.g. veri
by comp
computi
checking

Obvious
Comput

e.g. $b =$
$\approx$256 do
$\approx$256 do
$\approx$50 add
$\approx$50 add

```
(n+3)/8,P,P3)

R) - P3

(n+1)/8,P,P3)

R) - P
2,P,P3)

,P):
(n,P,P+P+P)
```


Worst case: $\approx b$ doublings plus $\approx b/3$ additions of $\pm P$ or $\pm 3P$. On average $\approx b/4$ additions.

Width-3 signed sliding windows: Precompute $P, 3P, 5P, 7P$. On average $\approx b/5$ additions.

Width 4: Precompute $P, 3P, 5P, 7P, 9P, 11P, 13P, 15P$. On average $\approx b/6$ additions.

Cost of precomputation eventually outweighs savings. Optimal: $\approx b$ doublings plus roughly $b/\lg b$ additions.

Double-scalar mul

Want to quickly c
$m, P, n, Q \mapsto mP$ -

e.g. verify signatu
by computing $h =$
computing $SB -$
checking whether

Obvious approach
Compute mP; co

e.g. $b = 256$:
$\approx$256 doublings fo
$\approx$256 doublings fo
$\approx$50 additions for
$\approx$50 additions for

,P3)

,P3)

P)

Worst case: $\approx b$ doublings plus $\approx b/3$ additions of $\pm P$ or $\pm 3P$.
On average $\approx b/4$ additions.

Width-3 signed sliding windows:
Precompute $P, 3P, 5P, 7P$.
On average $\approx b/5$ additions.

Width 4: Precompute
$P, 3P, 5P, 7P, 9P, 11P, 13P, 15P$.
On average $\approx b/6$ additions.

Cost of precomputation eventually outweighs savings.
Optimal: $\approx b$ doublings plus roughly $b/\lg b$ additions.

Double-scalar multiplication

Want to quickly compute
$m, P, n, Q \mapsto mP + nQ$.

e.g. verify signature (R, S)
by computing $h = H(R, M)$,
computing $SB - hA$,
checking whether $R = SB -$

Obvious approach:
Compute mP; compute nQ;

e.g. $b = 256$:
≈ 256 doublings for mP,
≈ 256 doublings for nQ,
≈ 50 additions for mP,
≈ 50 additions for nQ.

Worst case: $\approx b$ doublings plus $\approx b/3$ additions of $\pm P$ or $\pm 3P$.
On average $\approx b/4$ additions.

Width-3 signed sliding windows:
Precompute $P, 3P, 5P, 7P$.
On average $\approx b/5$ additions.

Width 4: Precompute
$P, 3P, 5P, 7P, 9P, 11P, 13P, 15P$.
On average $\approx b/6$ additions.

Cost of precomputation eventually outweighs savings.
Optimal: $\approx b$ doublings plus roughly $b/\lg b$ additions.

Double-scalar multiplication

Want to quickly compute
$m, P, n, Q \mapsto mP + nQ$.

e.g. verify signature (R, S)
by computing $h = H(R, M)$,
computing $SB - hA$,
checking whether $R = SB - hA$.

Obvious approach:
Compute mP; compute nQ; add.

e.g. $b = 256$:
$\approx$256 doublings for mP,
$\approx$256 doublings for nQ,
$\approx$50 additions for mP,
$\approx$50 additions for nQ.

ase: $\approx b$ doublings plus
dditions of $\pm P$ or $\pm 3P$.
age $\approx b/4$ additions.

signed sliding windows:
pute $P, 3P, 5P, 7P$.
age $\approx b/5$ additions.

: Precompute
$P, 7P, 9P, 11P, 13P, 15P$.
age $\approx b/6$ additions.

precomputation
ly outweighs savings.
: $\approx b$ doublings plus
$b/\lg b$ additions.

Double-scalar multiplication

Want to quickly compute
$m, P, n, Q \mapsto mP + nQ$.

e.g. verify signature (R, S)
by computing $h = H(R, M)$,
computing $SB - hA$,
checking whether $R = SB - hA$.

Obvious approach:
Compute mP; compute nQ; add.

e.g. $b = 256$:
$\approx$256 doublings for mP,
$\approx$256 doublings for nQ,
$\approx$50 additions for mP,
$\approx$50 additions for nQ.

Joint do

Do much
$2X + 2Y$

```
def sca
  if m =
    retu
  if n =
    retu
  R = sc
  R = R
  if m %
  if n %
  returr
```

oublings plus
$\pm P$ or $\pm 3P$.
additions.

ding windows:
$, 5P, 7P$.
additions.

oute
$1P, 13P, 15P$.
additions.

tation
hs savings.
blings plus
ditions.

Double-scalar multiplication

Want to quickly compute
$m, P, n, Q \mapsto mP + nQ$.

e.g. verify signature (R, S)
by computing $h = H(R, M)$,
computing $SB - hA$,
checking whether $R = SB - hA$.

Obvious approach:
Compute mP; compute nQ; add.

e.g. $b = 256$:
$\approx$256 doublings for mP,
$\approx$256 doublings for nQ,
$\approx$50 additions for mP,
$\approx$50 additions for nQ.

Joint doublings

Do much better b
$2X + 2Y$ into $2(X$

```
def scalarmult2(
  if m == 0:
    return scala
  if n == 0:
    return scala
  R = scalarmult
  R = R + R
  if m % 2: R =
  if n % 2: R =
  return R
```

lus

$3P$.

ows:

$.5P$.

s.

Double-scalar multiplication

Want to quickly compute
$m, P, n, Q \mapsto mP + nQ$.

e.g. verify signature (R, S)
by computing $h = H(R, M)$,
computing $SB - hA$,
checking whether $R = SB - hA$.

Obvious approach:
Compute mP; compute nQ; add.

e.g. $b = 256$:
$\approx$256 doublings for mP,
$\approx$256 doublings for nQ,
$\approx$50 additions for mP,
$\approx$50 additions for nQ.

Joint doublings

Do much better by merging
$2X + 2Y$ into $2(X + Y)$.

```
def scalarmult2(m,P,n,Q):
  if m == 0:
    return scalarmult(n,Q
  if n == 0:
    return scalarmult(m,P
  R = scalarmult2(m//2,P,
  R = R + R
  if m % 2: R = R + P
  if n % 2: R = R + Q
  return R
```

Double-scalar multiplication

Want to quickly compute
$m, P, n, Q \mapsto mP + nQ$.

e.g. verify signature (R, S)
by computing $h = H(R, M)$,
computing $SB - hA$,
checking whether $R = SB - hA$.

Obvious approach:
Compute mP; compute nQ; add.

e.g. $b = 256$:
≈256 doublings for mP,
≈256 doublings for nQ,
≈50 additions for mP,
≈50 additions for nQ.

Joint doublings

Do much better by merging
$2X + 2Y$ into $2(X + Y)$.

```
def scalarmult2(m,P,n,Q):
  if m == 0:
    return scalarmult(n,Q)
  if n == 0:
    return scalarmult(m,P)
  R = scalarmult2(m//2,P,n//2,Q)
  R = R + R
  if m % 2: R = R + P
  if n % 2: R = R + Q
  return R
```

scalar multiplication

quickly compute
$Q \mapsto mP + nQ$.

fy signature (R, S)
puting $h = H(R, M)$,
ng $SB - hA$,
whether $R = SB - hA$.

approach:
e mP; compute nQ; add.

256:
oublings for mP,
oublings for nQ,
ditions for mP,
ditions for nQ.

Joint doublings

Do much better by merging
$2X + 2Y$ into $2(X + Y)$.

```
def scalarmult2(m,P,n,Q):
  if m == 0:
    return scalarmult(n,Q)
  if n == 0:
    return scalarmult(m,P)
  R = scalarmult2(m//2,P,n//2,Q)
  R = R + R
  if m % 2: R = R + P
  if n % 2: R = R + Q
  return R
```

For exan
$35P = 2$
$31Q = 2$
into $35P$
$2(2(2(2($
$+P$

$\approx b$ doub
$\approx b/2$ ad
$\approx b/2$ ad

Combine
≈ 256 do
≈ 50 add
≈ 50 add

tiplication

ompute

$+ nQ$.

re (R, S)

$H(R, M)$,

hA,

$R = SB - hA$.

:

npute nQ; add.

or mP,

or nQ,

mP,

nQ.

Joint doublings

Do much better by merging $2X + 2Y$ into $2(X + Y)$.

```
def scalarmult2(m,P,n,Q):
  if m == 0:
    return scalarmult(n,Q)
  if n == 0:
    return scalarmult(m,P)
  R = scalarmult2(m//2,P,n//2,Q)
  R = R + R
  if m % 2: R = R + P
  if n % 2: R = R + Q
  return R
```

For example: merg

$35P = 2(2(2(2(2F$

$31Q = 2(2(2(2Q+$

into $35P + 31Q =$

$2(2(2(2(2P+Q)+$

$+P+Q$.

$\approx b$ doublings (me

$\approx b/2$ additions of

$\approx b/2$ additions of

Combine idea with

≈ 256 doublings fo

≈ 50 additions usin

≈ 50 additions usin

,

$+ hA$.

add.

Joint doublings

Do much better by merging $2X + 2Y$ into $2(X + Y)$.

```
def scalarmult2(m,P,n,Q):
  if m == 0:
    return scalarmult(n,Q)
  if n == 0:
    return scalarmult(m,P)
  R = scalarmult2(m//2,P,n//2,Q)
  R = R + R
  if m % 2: R = R + P
  if n % 2: R = R + Q
  return R
```


For example: merge

$35P = 2(2(2(2(2P))) + P)$ +

$31Q = 2(2(2(2Q{+}Q){+}Q){+}$

into $35P + 31Q =$

$2(2(2(2(2P{+}Q){+}Q){+}Q){+}$

$\quad {+}P{+}Q$.

$\approx b$ doublings (merged!),

$\approx b/2$ additions of P,

$\approx b/2$ additions of Q.

Combine idea with windows:

≈ 256 doublings for $b = 256$

≈ 50 additions using P,

≈ 50 additions using Q.

Joint doublings

Do much better by merging $2X + 2Y$ into $2(X + Y)$.

```
def scalarmult2(m,P,n,Q):
  if m == 0:
    return scalarmult(n,Q)
  if n == 0:
    return scalarmult(m,P)
  R = scalarmult2(m//2,P,n//2,Q)
  R = R + R
  if m % 2: R = R + P
  if n % 2: R = R + Q
  return R
```


For example: merge

$35P = 2(2(2(2(2P))) + P) + P$,

$31Q = 2(2(2(2Q+Q)+Q)+Q)+Q$

into $35P + 31Q =$

$2(2(2(2(2P+Q)+Q)+Q)+P+Q)$
$+P+Q$.

$\approx b$ doublings (merged!),

$\approx b/2$ additions of P,

$\approx b/2$ additions of Q.

Combine idea with windows: e.g.,

≈ 256 doublings for $b = 256$,

≈ 50 additions using P,

≈ 50 additions using Q.

ublings

h better by merging
Y into $2(X+Y)$.

```
larmult2(m,P,n,Q):
== 0:
urn scalarmult(n,Q)
== 0:
urn scalarmult(m,P)
calarmult2(m//2,P,n//2,Q)
 + R
% 2: R = R + P
% 2: R = R + Q
n R
```


For example: merge
$35P = 2(2(2(2(2P))) + P) + P$,
$31Q = 2(2(2(2Q{+}Q){+}Q){+}Q){+}Q$
into $35P + 31Q =$
$2(2(2(2(2P{+}Q){+}Q){+}Q){+}P{+}Q)$
$\quad {+}P{+}Q$.

$\approx b$ doublings (merged!),
$\approx b/2$ additions of P,
$\approx b/2$ additions of Q.

Combine idea with windows: e.g.,
$\approx$256 doublings for $b = 256$,
$\approx$50 additions using P,
$\approx$50 additions using Q.

Batch ve

Verifying
need to
$S_1 B =$
$S_2 B =$
$S_3 B =$
etc.

Obvious
Check ea

y merging

$' + Y)$.

```
m,P,n,Q):

rmult(n,Q)

rmult(m,P)

2(m//2,P,n//2,Q)

R + P

R + Q
```


For example: merge

$35P = 2(2(2(2(2P))) + P) + P$,

$31Q = 2(2(2(2Q+Q)+Q)+Q)+Q$

into $35P + 31Q =$

$2(2(2(2(2P+Q)+Q)+Q)+P+Q)$
$\quad +P+Q$.

$\approx b$ doublings (merged!),

$\approx b/2$ additions of P,

$\approx b/2$ additions of Q.

Combine idea with windows: e.g.,

≈ 256 doublings for $b = 256$,

≈ 50 additions using P,

≈ 50 additions using Q.

Batch verification

Verifying many sig

need to be confide

$S_1 B = R_1 + h_1 A_1$

$S_2 B = R_2 + h_2 A_2$

$S_3 B = R_3 + h_3 A_3$

etc.

Obvious approach:

Check each equati

)

)

n//2,Q)

For example: merge

$35P = 2(2(2(2(2P))) + P) + P$,

$31Q = 2(2(2(2Q+Q)+Q)+Q)+Q$

into $35P + 31Q =$

$2(2(2(2(2P+Q)+Q)+Q)+P+Q)$
$\quad +P+Q$.

$\approx b$ doublings (merged!),

$\approx b/2$ additions of P,

$\approx b/2$ additions of Q.

Combine idea with windows: e.g.,

≈ 256 doublings for $b = 256$,

≈ 50 additions using P,

≈ 50 additions using Q.

Batch verification

Verifying many signatures:

need to be confident that

$S_1 B = R_1 + h_1 A_1$,

$S_2 B = R_2 + h_2 A_2$,

$S_3 B = R_3 + h_3 A_3$,

etc.

Obvious approach:

Check each equation separat

For example: merge

$35P = 2(2(2(2(2P))) + P) + P$,

$31Q = 2(2(2(2Q{+}Q){+}Q){+}Q){+}Q$

into $35P + 31Q =$

$2(2(2(2(2P{+}Q){+}Q){+}Q){+}P{+}Q)$
$\quad {+}P{+}Q$.

$\approx b$ doublings (merged!),

$\approx b/2$ additions of P,

$\approx b/2$ additions of Q.

Combine idea with windows: e.g.,

≈ 256 doublings for $b = 256$,

≈ 50 additions using P,

≈ 50 additions using Q.

Batch verification

Verifying many signatures:

need to be confident that

$S_1 B = R_1 + h_1 A_1$,

$S_2 B = R_2 + h_2 A_2$,

$S_3 B = R_3 + h_3 A_3$,

etc.

Obvious approach:

Check each equation separately.

For example: merge

$35P = 2(2(2(2(2P))) + P) + P$,

$31Q = 2(2(2(2Q+Q)+Q)+Q)+Q$

into $35P + 31Q =$

$2(2(2(2(2P+Q)+Q)+Q)+P+Q)$
$+P+Q$.

$\approx b$ doublings (merged!),

$\approx b/2$ additions of P,

$\approx b/2$ additions of Q.

Combine idea with windows: e.g.,

≈ 256 doublings for $b = 256$,

≈ 50 additions using P,

≈ 50 additions using Q.

Batch verification

Verifying many signatures:
need to be confident that

$S_1 B = R_1 + h_1 A_1$,

$S_2 B = R_2 + h_2 A_2$,

$S_3 B = R_3 + h_3 A_3$,

etc.

Obvious approach:
Check each equation separately.

Much faster approach:
Check random linear combination
of the equations.

mple: merge

$2(2(2(2(2P))) + P) + P,$

$2(2(2(2Q{+}Q){+}Q){+}Q){+}Q$

$P + 31Q =$

$(2P{+}Q){+}Q){+}Q){+}P{+}Q)$

$P{+}Q.$

olings (merged!),

dditions of P,

dditions of Q.

e idea with windows: e.g.,

oublings for $b = 256$,

ditions using P,

ditions using Q.

Batch verification

Verifying many signatures:
need to be confident that
$S_1B = R_1 + h_1A_1,$
$S_2B = R_2 + h_2A_2,$
$S_3B = R_3 + h_3A_3,$
etc.

Obvious approach:
Check each equation separately.

Much faster approach:
Check random linear combination
of the equations.

Pick ind

128-bit

Check w

$(z_1S_1 +$

$z_1R_1 + ($

$z_2R_2 + ($

$z_3R_3 + ($

(If $\neq$: S

Doumen

Easy to

forgeries

of foolin

ge

$P))) + P) + P$,

$Q)+Q)+Q)+Q$

$Q)+Q)+P+Q)$

rged!),

P,

Q.

n windows: e.g.,

or $b = 256$,

ng P,

ng Q.

Batch verification

Verifying many signatures:
need to be confident that
$S_1 B = R_1 + h_1 A_1$,
$S_2 B = R_2 + h_2 A_2$,
$S_3 B = R_3 + h_3 A_3$,
etc.

Obvious approach:
Check each equation separately.

Much faster approach:
Check random linear combination
of the equations.

Pick independent
128-bit $z_1, z_2, z_3, .$

Check whether
$(z_1 S_1 + z_2 S_2 + z_3$
$z_1 R_1 + (z_1 h_1) A_1 -$
$z_2 R_2 + (z_2 h_2) A_2 -$
$z_3 R_3 + (z_3 h_3) A_3 -$

(If $\neq$: See 2012 B
Doumen–Lange–O

Easy to prove:
forgeries have prob
of fooling this che

$+ P$,

$Q)+Q$

$P+Q)$

: e.g.,

,

Batch verification

Verifying many signatures:
need to be confident that
$S_1 B = R_1 + h_1 A_1$,
$S_2 B = R_2 + h_2 A_2$,
$S_3 B = R_3 + h_3 A_3$,
etc.

Obvious approach:
Check each equation separately.

Much faster approach:
Check random linear combination of the equations.

Pick independent uniform ra
128-bit $z_1, z_2, z_3, \ldots$.

Check whether
$(z_1 S_1 + z_2 S_2 + z_3 S_3 + \cdots)B$
$z_1 R_1 + (z_1 h_1) A_1 +$
$z_2 R_2 + (z_2 h_2) A_2 +$
$z_3 R_3 + (z_3 h_3) A_3 + \cdots$.

(If $\neq$: See 2012 Bernstein–
Doumen–Lange–Oosterwijk.

Easy to prove:
forgeries have probability ≤ 2
of fooling this check.

Batch verification

Verifying many signatures:
need to be confident that
$S_1 B = R_1 + h_1 A_1$,
$S_2 B = R_2 + h_2 A_2$,
$S_3 B = R_3 + h_3 A_3$,
etc.

Obvious approach:
Check each equation separately.

Much faster approach:
Check random linear combination of the equations.

Pick independent uniform random 128-bit $z_1, z_2, z_3, \ldots$.

Check whether
$(z_1 S_1 + z_2 S_2 + z_3 S_3 + \cdots)B =$
$z_1 R_1 + (z_1 h_1)A_1 +$
$z_2 R_2 + (z_2 h_2)A_2 +$
$z_3 R_3 + (z_3 h_3)A_3 + \cdots$.

(If $\neq$: See 2012 Bernstein–Doumen–Lange–Oosterwijk.)

Easy to prove:
forgeries have probability $\leq 2^{-128}$ of fooling this check.

erification

g many signatures:
be confident that
$R_1 + h_1 A_1$,
$R_2 + h_2 A_2$,
$R_3 + h_3 A_3$,

approach:
ach equation separately.

ster approach:
andom linear combination
quations.

Pick independent uniform random 128-bit $z_1, z_2, z_3, \ldots$.

Check whether

$$(z_1 S_1 + z_2 S_2 + z_3 S_3 + \cdots)B =$$
$$z_1 R_1 + (z_1 h_1) A_1 +$$
$$z_2 R_2 + (z_2 h_2) A_2 +$$
$$z_3 R_3 + (z_3 h_3) A_3 + \cdots.$$

(If $\neq$: See 2012 Bernstein–Doumen–Lange–Oosterwijk.)

Easy to prove:
forgeries have probability $\le 2^{-128}$ of fooling this check.

Multi-sc

Review o

1939 Bra

$\approx (1 + 1$
additions
$P \mapsto nP$

1964 Str

$\approx (1 + ...$
additions
$P_1, \ldots,$
if $n_1, \ldots$

gnatures:

ent that

,

,

,

:

on separately.

ach:

ear combination

Pick independent uniform random 128-bit $z_1, z_2, z_3, \ldots$.

Check whether

$(z_1 S_1 + z_2 S_2 + z_3 S_3 + \cdots)B =$
$z_1 R_1 + (z_1 h_1)A_1 +$
$z_2 R_2 + (z_2 h_2)A_2 +$
$z_3 R_3 + (z_3 h_3)A_3 + \cdots$.

(If $\neq$: See 2012 Bernstein–Doumen–Lange–Oosterwijk.)

Easy to prove:
forgeries have probability $\leq 2^{-128}$ of fooling this check.

Multi-scalar multi

Review of asympto

1939 Brauer (wind

$\approx (1 + 1/\lg b)b$
additions to comp
$P \mapsto nP$ if $n < 2^b$

1964 Straus (joint

$\approx (1 + k/\lg b)b$
additions to comp
$P_1, \ldots, P_k \mapsto n_1 P$
if $n_1, \ldots, n_k < 2^b$.

tely.

nation

Pick independent uniform random 128-bit $z_1, z_2, z_3, \ldots$.

Check whether
$(z_1 S_1 + z_2 S_2 + z_3 S_3 + \cdots)B =$
$z_1 R_1 + (z_1 h_1)A_1 +$
$z_2 R_2 + (z_2 h_2)A_2 +$
$z_3 R_3 + (z_3 h_3)A_3 + \cdots$.

(If $\neq$: See 2012 Bernstein–Doumen–Lange–Oosterwijk.)

Easy to prove:
forgeries have probability $\le 2^{-128}$ of fooling this check.

Multi-scalar multiplication

Review of asymptotic speeds

1939 Brauer (windows):

$\approx (1 + 1/\lg b)b$
additions to compute
$P \mapsto nP$ if $n < 2^b$.

1964 Straus (joint doublings

$\approx (1 + k/\lg b)b$
additions to compute
$P_1, \ldots, P_k \mapsto n_1 P_1 + \cdots +$
if $n_1, \ldots, n_k < 2^b$.

Pick independent uniform random 128-bit $z_1, z_2, z_3, \ldots$.

Check whether

$(z_1 S_1 + z_2 S_2 + z_3 S_3 + \cdots)B =$
$z_1 R_1 + (z_1 h_1)A_1 +$
$z_2 R_2 + (z_2 h_2)A_2 +$
$z_3 R_3 + (z_3 h_3)A_3 + \cdots$.

(If $\neq$: See 2012 Bernstein–Doumen–Lange–Oosterwijk.)

Easy to prove: forgeries have probability $\le 2^{-128}$ of fooling this check.

Multi-scalar multiplication

Review of asymptotic speeds:

1939 Brauer (windows):

$\approx (1 + 1/\lg b)b$ additions to compute $P \mapsto nP$ if $n < 2^b$.

1964 Straus (joint doublings):

$\approx (1 + k/\lg b)b$ additions to compute $P_1, \ldots, P_k \mapsto n_1 P_1 + \cdots + n_k P_k$ if $n_1, \ldots, n_k < 2^b$.

ependent uniform random

$z_1, z_2, z_3, \ldots$.

hether

$z_2S_2 + z_3S_3 + \cdots)B =$

$(z_1h_1)A_1 +$

$(z_2h_2)A_2 +$

$(z_3h_3)A_3 + \cdots$.

ee 2012 Bernstein–

–Lange–Oosterwijk.)

prove:

have probability $\le 2^{-128}$

g this check.

Multi-scalar multiplication

Review of asymptotic speeds:

1939 Brauer (windows):

$\approx (1 + 1/\lg b)b$
additions to compute
$P \mapsto nP$ if $n < 2^b$.

1964 Straus (joint doublings):

$\approx (1 + k/\lg b)b$
additions to compute
$P_1, \ldots, P_k \mapsto n_1P_1 + \cdots + n_kP_k$
if $n_1, \ldots, n_k < 2^b$.

1976 Ya

$\approx (1 +$

addition

$P \mapsto n_1$

if $n_1, \ldots$

1976 Pi

Similar

but repl

Faster t

if k is l

(Knuth

as if spe

uniform random

. . .

$S_3 + \cdots)B =$

$+$

$+$

$+ \cdots.$

ernstein–

Oosterwijk.)

bability $\le 2^{-128}$

ck.

Multi-scalar multiplication

Review of asymptotic speeds:

1939 Brauer (windows):

$\approx (1 + 1/\lg b)b$
additions to compute
$P \mapsto nP$ if $n < 2^b$.

1964 Straus (joint doublings):

$\approx (1 + k/\lg b)b$
additions to compute
$P_1, \ldots, P_k \mapsto n_1 P_1 + \cdots + n_k P_k$
if $n_1, \ldots, n_k < 2^b$.

1976 Yao:

$\approx (1 + k/\lg b)b$
additions to comp
$P \mapsto n_1 P, \ldots, n_k P$
if $n_1, \ldots, n_k < 2^b$.

1976 Pippenger:

Similar asymptotic
but replace lg b wi
Faster than Straus
if k is large.

(Knuth says "gene
as if speed were th

ndom

$B =$

)

2^{-128}

Multi-scalar multiplication

Review of asymptotic speeds:

1939 Brauer (windows):

$\approx (1 + 1/\lg b)b$
additions to compute
$P \mapsto nP$ if $n < 2^b$.

1964 Straus (joint doublings):

$\approx (1 + k/\lg b)b$
additions to compute
$P_1, \ldots, P_k \mapsto n_1 P_1 + \cdots + n_k P_k$
if $n_1, \ldots, n_k < 2^b$.

1976 Yao:

$\approx (1 + k/\lg b)b$
additions to compute
$P \mapsto n_1 P, \ldots, n_k P$
if $n_1, \ldots, n_k < 2^b$.

1976 Pippenger:

Similar asymptotics,
but replace $\lg b$ with $\lg(kb)$.
Faster than Straus and Yao
if k is large.

(Knuth says "generalization"
as if speed were the same.)

Multi-scalar multiplication

Review of asymptotic speeds:

1939 Brauer (windows):

$\approx (1 + 1/\lg b)b$
additions to compute
$P \mapsto nP$ if $n < 2^b$.

1964 Straus (joint doublings):

$\approx (1 + k/\lg b)b$
additions to compute
$P_1, \ldots, P_k \mapsto n_1P_1 + \cdots + n_kP_k$
if $n_1, \ldots, n_k < 2^b$.

1976 Yao:

$\approx (1 + k/\lg b)b$
additions to compute
$P \mapsto n_1P, \ldots, n_kP$
if $n_1, \ldots, n_k < 2^b$.

1976 Pippenger:

Similar asymptotics,
but replace $\lg b$ with $\lg(kb)$.
Faster than Straus and Yao
if k is large.

(Knuth says "generalization"
as if speed were the same.)

alar multiplication

of asymptotic speeds:

auer (windows):

$1/\lg b)b$

s to compute

P if $n < 2^b$.

raus (joint doublings):

$k/\lg b)b$

s to compute

$P_k \mapsto n_1 P_1 + \cdots + n_k P_k$

$, n_k < 2^b$.

1976 Yao:

$\approx (1 + k/\lg b)b$

additions to compute

$P \mapsto n_1 P, \ldots, n_k P$

if $n_1, \ldots, n_k < 2^b$.

1976 Pippenger:

Similar asymptotics,
but replace $\lg b$ with $\lg(kb)$.
Faster than Straus and Yao
if k is large.

(Knuth says "generalization"
as if speed were the same.)

More ge

algorithr

ℓ sums c

$\approx \Big(\min$

if all coe

Within 1

plication

otic speeds:

dows):

ute

.

doublings):

ute

$_1 + \cdots + n_k P_k$

.

1976 Yao:

$\approx (1 + k/\lg b)b$
additions to compute
$P \mapsto n_1 P, \ldots, n_k P$
if $n_1, \ldots, n_k < 2^b$.

1976 Pippenger:

Similar asymptotics,
but replace $\lg b$ with $\lg(kb)$.
Faster than Straus and Yao
if k is large.

(Knuth says "generalization"
as if speed were the same.)

More generally, Pi
algorithm compute
ℓ sums of multiple

$\approx \Big(\min\{k, \ell\} + \frac{}{\lg}$

if all coefficients a
Within $1 + \epsilon$ of op

s:

s):

$n_k P_k$

1976 Yao:

$\approx (1 + k/\lg b)b$
additions to compute
$P \mapsto n_1 P, \ldots, n_k P$
if $n_1, \ldots, n_k < 2^b$.

1976 Pippenger:

Similar asymptotics,
but replace $\lg b$ with $\lg(kb)$.
Faster than Straus and Yao
if k is large.

(Knuth says "generalization"
as if speed were the same.)

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

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

if all coefficients are below 2
Within $1 + \epsilon$ of optimal.

1976 Yao:

$\approx (1 + k/\lg b)b$
additions to compute
$P \mapsto n_1P, \ldots, n_kP$
if $n_1, \ldots, n_k < 2^b$.

1976 Pippenger:

Similar asymptotics,
but replace $\lg b$ with $\lg(kb)$.
Faster than Straus and Yao
if k is large.

(Knuth says "generalization"
as if speed were the same.)

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

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

if all coefficients are below 2^b.
Within $1+\epsilon$ of optimal.

1976 Yao:

$\approx (1 + k/\lg b)b$
additions to compute
$P \mapsto n_1 P, \ldots, n_k P$
if $n_1, \ldots, n_k < 2^b$.

1976 Pippenger:

Similar asymptotics,
but replace $\lg b$ with $\lg(kb)$.
Faster than Straus and Yao
if k is large.

(Knuth says "generalization"
as if speed were the same.)

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

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

if all coefficients are below 2^b.
Within $1 + \epsilon$ of optimal.

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

o:

$k/\lg b)b$
s to compute
$P, \ldots, n_k P$
$, n_k < 2^b$.

ppenger:

asymptotics,
ace $\lg b$ with $\lg(kb)$.
han Straus and Yao
rge.

says “generalization”
ed were the same.)

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

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

if all coefficients are below 2^b.
Within $1 + \epsilon$ of optimal.

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

No! 198

If $n_1 \geq$
$n_1 P_1 +$
$(n_1 - q$
$n_3 P_3 + \cdot$

Remarka
competi
for rand
much be

ute

cs,
ith lg(kb).
s and Yao

eralization"
he same.)

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

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

if all coefficients are below 2^b.
Within $1+\epsilon$ of optimal.

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

No! 1989 Bos–Co

If $n_1 \ge n_2 \ge \cdots$ th
$n_1 P_1 + n_2 P_2 + n_3$
$(n_1 - q n_2) P_1 + n_2$
$n_3 P_3 + \cdots$ where

Remarkably simple
competitive with P
for random choice
much better mem

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

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

if all coefficients are below 2^b.

Within $1+\epsilon$ of optimal.

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

Is that the end of the story?

No! 1989 Bos–Coster:

If $n_1 \ge n_2 \ge \cdots$ then

$n_1P_1 + n_2P_2 + n_3P_3 + \cdots =$

$(n_1 - qn_2)P_1 + n_2(qP_1 + P_2$

$n_3P_3 + \cdots$ where $q = \lfloor n_1/n$

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

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

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

if all coefficients are below 2^b.
Within $1+\epsilon$ of optimal.

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

No! 1989 Bos–Coster:

If $n_1 \ge n_2 \ge \cdots$ then
$n_1P_1 + n_2P_2 + n_3P_3 + \cdots =$
$(n_1 - qn_2)P_1 + n_2(qP_1 + P_2) +$
$n_3P_3 + \cdots$ where $q = \lfloor n_1/n_2 \rfloor$.

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

nerally, Pippenger's
m computes
of multiples of k inputs.

$\{k, \ell\} + \dfrac{k\ell}{\lg(k\ell b)}\Big) b$ adds

efficients are below 2^b.

$1 + \epsilon$ of optimal.

special cases of
er's algorithm were
ed and patented by
ickell–Gordon–McCurley–
1995 Lim–Lee, etc.
he end of the story?

No! 1989 Bos–Coster:

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

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

Example

0001000
0000100
1001011
0100100
0010011
0000000
0000000

Goal: C
$300P$, 1

ppenger's
es
s of k inputs.
$k\ell$
g$(k\ell b)$) b adds
re below 2^b.
ptimal.

ses of
thm were
tented by
don–McCurley–
-Lee, etc.
the story?

No! 1989 Bos–Coster:

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

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

Example of Bos–C

000100000 = 32
000010000 = 16
100101100 = 300
010010010 = 146
001001101 = 77
000000010 = 2
000000001 = 1

Goal: Compute 32
$300P$, $146P$, $77P$,

uts.

adds

^{b}.

urley–

No! 1989 Bos–Coster:

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

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

Example of Bos–Coster:

$000100000 = 32$
$000010000 = 16$
$100101100 = 300$
$010010010 = 146$
$001001101 = 77$
$000000010 = 2$
$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

No! 1989 Bos–Coster:

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

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

Example of Bos–Coster:

000100000 = 32
000010000 = 16
100101100 = 300
010010010 = 146
001001101 = 77
000000010 = 2
000000001 = 1

Goal: Compute $32P$, $16P$,
$300P$, $146P$, $77P$, $2P$, $1P$.

39 Bos–Coster:

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

ably simple;
tive with Pippenger
om choices of n_i's;
etter memory usage.

Example of Bos–Coster:

$000100000 = 32$
$000010000 = 16$
$100101100 = 300$
$010010010 = 146$
$001001101 = 77$
$000000010 = 2$
$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce

0001000
0000100
0100110
0100100
0010011
0000000
0000000

Goal: Co
$154P$, 14
Plus one
add 146
obtainin

ster:

hen

$P_3 + \cdots =$

$(qP_1 + P_2) +$

$q = \lfloor n_1/n_2 \rfloor$.

e;

Pippenger

s of n_i's;

ory usage.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row

$000100000 = 32$

$000010000 = 16$

$010011010 = 154$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute 32

$154P$, $146P$, $77P$,

Plus one extra add

add $146P$ into 154

obtaining $300P$.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$010011010 = 154 \leftarrow$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $154P$, $146P$, $77P$, $2P$, $1P$. Plus one extra addition: add $146P$ into $154P$, obtaining $300P$.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$010011010 = 154 \leftarrow$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $154P$, $146P$, $77P$, $2P$, $1P$.
Plus one extra addition: add $146P$ into $154P$, obtaining $300P$.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$000001000 = 8 \leftarrow$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

plus 2 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$000001000 = 8$

$001000101 = 69 \leftarrow$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

plus 3 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$000001000 = 8$

$001000101 = 69$

$000001000 = 8 \leftarrow$

$000000010 = 2$

$000000001 = 1$

plus 4 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$000001000 = 8$

$000100101 = 37 \leftarrow$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 5 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000100000 = 32$

$000010000 = 16$

$000001000 = 8$

$000000101 = 5 \leftarrow$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 6 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000010000 = 16 \leftarrow$

$000010000 = 16$

$000001000 = 8$

$000000101 = 5$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 7 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000010000 = 16$

$000001000 = 8$

$000000101 = 5$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 7 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000001000 = 8 \leftarrow$

$000001000 = 8$

$000000101 = 5$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 8 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0 \leftarrow$

$000001000 = 8$

$000000101 = 5$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 8 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0 \leftarrow$

$000000101 = 5$

$000001000 = 8$

$000000010 = 2$

$000000001 = 1$

plus 8 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000101 = 5$

$000000011 = 3 \leftarrow$

$000000010 = 2$

$000000001 = 1$

plus 9 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000010 = 2 \leftarrow$

$000000011 = 3$

$000000010 = 2$

$000000001 = 1$

plus 10 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000010 = 2$

$000000001 = 1 \leftarrow$

$000000010 = 2$

$000000001 = 1$

plus 11 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0 \leftarrow$

$000000001 = 1$

$000000010 = 2$

$000000001 = 1$

plus 11 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000001 = 1$

$000000001 = 1 \leftarrow$

$000000001 = 1$

plus 12 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0 \leftarrow$

$000000001 = 1$

$000000001 = 1$

plus 12 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0 \leftarrow$

$000000001 = 1$

plus 12 additions.

Example of Bos–Coster:

$000100000 = 32$

$000010000 = 16$

$100101100 = 300$

$010010010 = 146$

$001001101 = 77$

$000000010 = 2$

$000000001 = 1$

Goal: Compute $32P$, $16P$, $300P$, $146P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0 \leftarrow$

plus 12 additions.

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

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

e of Bos–Coster:

$00 = 32$

$00 = 16$

$00 = 300$

$10 = 146$

$01 = 77$

$10 = 2$

$01 = 1$

ompute $32P$, $16P$,

$46P$, $77P$, $2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0 \leftarrow$

plus 12 additions.

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

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

Revised

$32P_1 +$

$77P_5 +$

First co

and the

$32P_1 +$

$77P_5 +$

Same sc

Ed2551

verify b

about t

verifyin

Coster:

$2P$, $16P$,

$2P$, $1P$.

Reduce largest row:

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0$

$000000000 = 0 \leftarrow$

plus 12 additions.

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

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

Revised goal: Com

$32P_1 + 16P_2 + 30$

$77P_5 + 2P_6 + 1P_7$

First compute $P'_4 =$

and then recursive

$32P_1 + 16P_2 + 15$

$77P_5 + 2P_6 + 1P_7$

Same scalars show

Ed25519 batch ve

verify batch of 64

about twice as fas

verifying each sepa

Reduce largest row:

000000000 = 0
000000000 = 0
000000000 = 0
000000000 = 0
000000000 = 0
000000000 = 0
000000000 = 0 ←

plus 12 additions.

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

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

Revised goal: Compute
$32P_1 + 16P_2 + 300P_3 + 146$
$77P_5 + 2P_6 + 1P_7$.

First compute $P_4' = P_4 + P_3$
and then recursively comput
$32P_1 + 16P_2 + 154P_3 + 146$
$77P_5 + 2P_6 + 1P_7$.

Same scalars show up as bef

Ed25519 batch verification:
verify batch of 64 signatures
about twice as fast as
verifying each separately.

Reduce largest row:

$000000000 = 0$
$000000000 = 0$
$000000000 = 0$
$000000000 = 0$
$000000000 = 0$
$000000000 = 0$
$000000000 = 0 \leftarrow$

plus 12 additions.

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

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

Revised goal: Compute $32P_1 + 16P_2 + 300P_3 + 146P_4 + 77P_5 + 2P_6 + 1P_7$.

First compute $P_4' = P_4 + P_3$ and then recursively compute $32P_1 + 16P_2 + 154P_3 + 146P_4' + 77P_5 + 2P_6 + 1P_7$.

Same scalars show up as before.

Ed25519 batch verification: verify batch of 64 signatures about twice as fast as verifying each separately.