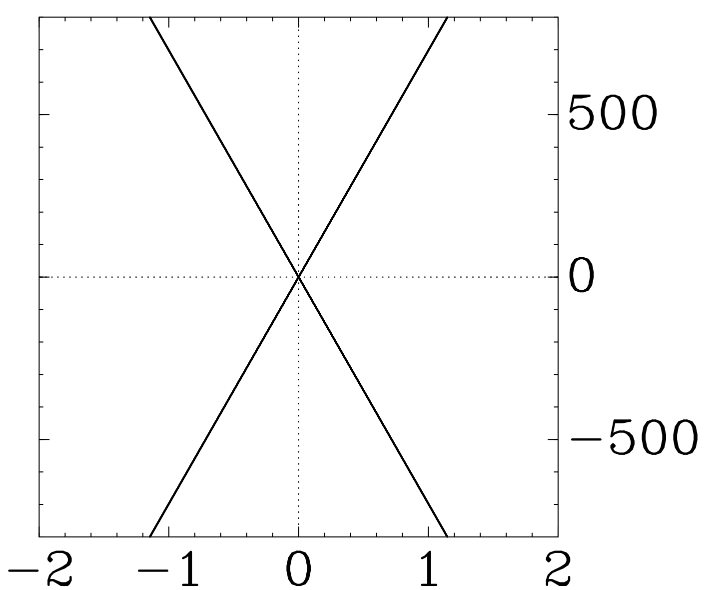
New speed records for point multiplication

D. J. Bernstein



Thanks to:
University of Illinois at Chicago
NSF CCR–9983950
Alfred P. Sloan Foundation

640838 Pentium M cycles to compute a 32-byte secret shared by Dan and Tanja, given Dan's 32-byte secret key n and Tanja's 32-byte public key K.

All known attacks: $> 2^{128}$ cycles.

This is the new speed record for high-security Diffie-Hellman.

Encrypt and authenticate messages using hash of shared secret as key. Diffie-Hellman is the bottleneck if total message length is short.

s
ation

is at Chicago
0
undation

640838 Pentium M cycles
to compute a 32-byte secret
shared by Dan and Tanja,
given Dan's 32-byte secret key n
and Tanja's 32-byte public key K.

All known attacks: $> 2^{128}$ cycles.

This is the new speed record
for high-security Diffie-Hellman.

Encrypt and authenticate messages
using hash of shared secret as key.
Diffie-Hellman is the bottleneck
if total message length is short.

640838 Pentium M
to compute x-coor
multiple of $(K, \ldots$
given $K \in \{0, 1, \ldots$
$n \in 2^{254} + 8\{0, 1,$

Curve25519 is the
$y^2 = x^3 + 486662$
mod the prime 2^{25}

624786 Athlon (62
832457 Pentium II
957904 Pentium 4
I anticipate similar
for UltraSPARC, P

640838 Pentium M cycles
to compute a 32-byte secret
shared by Dan and Tanja,
given Dan's 32-byte secret key n
and Tanja's 32-byte public key K.

All known attacks: $> 2^{128}$ cycles.

This is the new speed record
for high-security Diffie-Hellman.

Encrypt and authenticate messages
using hash of shared secret as key.
Diffie-Hellman is the bottleneck
if total message length is short.

640838 Pentium M (695) cycles
to compute x-coordinate of nth
multiple of $(K, \ldots)$ on Curve25519,
given $K \in \{0, 1, \ldots, 2^{256} - 1\}$ and
$n \in 2^{254} + 8\{0, 1, \ldots, 2^{251} - 1\}$.

Curve25519 is the elliptic curve
$y^2 = x^3 + 486662x^2 + x$
mod the prime $2^{255} - 19$.

624786 Athlon (622) cycles;
832457 Pentium III (686) cycles;
957904 Pentium 4 (f12) cycles.
I anticipate similar cycle counts
for UltraSPARC, PowerPC, etc.

/I cycles
byte secret
d Tanja,
te secret key n
te public key K.

: $> 2^{128}$ cycles.

eed record
Diffie-Hellman.

enticate messages
ed secret as key.
he bottleneck
ngth is short.

640838 Pentium M (695) cycles to compute x-coordinate of nth multiple of $(K, \ldots)$ on Curve25519, given $K \in \{0, 1, \ldots, 2^{256} - 1\}$ and $n \in 2^{254} + 8\{0, 1, \ldots, 2^{251} - 1\}$.

Curve25519 is the elliptic curve
$y^2 = x^3 + 486662x^2 + x$
mod the prime $2^{255} - 19$.

624786 Athlon (622) cycles;
832457 Pentium III (686) cycles;
957904 Pentium 4 (f12) cycles.
I anticipate similar cycle counts for UltraSPARC, PowerPC, etc.

Immune to timing
including cache-tir
including hyperthr
No data-dependen
no data-dependent

Software is in publ
16 kilobytes when
`cr.yp.to/ecdh.h`

No known patent

For comparison, B
much smaller prim
780000 PII cycles;
no timing-attack p

640838 Pentium M (695) cycles to compute x-coordinate of nth multiple of $(K, \ldots)$ on Curve25519, given $K \in \{0, 1, \ldots, 2^{256} - 1\}$ and $n \in 2^{254} + 8\{0, 1, \ldots, 2^{251} - 1\}$.

Curve25519 is the elliptic curve $y^2 = x^3 + 486662x^2 + x$ mod the prime $2^{255} - 19$.

624786 Athlon (622) cycles;
832457 Pentium III (686) cycles;
957904 Pentium 4 (f12) cycles.
I anticipate similar cycle counts for UltraSPARC, PowerPC, etc.

Immune to timing attacks, including cache-timing attacks, including hyperthreading attacks. No data-dependent branches; no data-dependent indexing.

Software is in public domain.
16 kilobytes when compiled.
`cr.yp.to/ecdh.html`

No known patent problems.

For comparison, Brown et al.: much smaller prime, $2^{192} - 2^{64} - 1$; 780000 PII cycles; y given; no timing-attack protection.

M (695) cycles
rdinate of nth
) on Curve25519,
$\ldots, 2^{256}-1\}$ and
$\ldots, 2^{251}-1\}$.

elliptic curve
x^2+x
$^{55}-19$.

22) cycles;
II (686) cycles;
(f12) cycles.
cycle counts
PowerPC, etc.

Immune to timing attacks,
including cache-timing attacks,
including hyperthreading attacks.
No data-dependent branches;
no data-dependent indexing.

Software is in public domain.
16 kilobytes when compiled.
`cr.yp.to/ecdh.html`

No known patent problems.

For comparison, Brown et al.:
much smaller prime, $2^{192}-2^{64}-1$;
780000 PII cycles; y given;
no timing-attack protection.

Where are the cyc

Focus today on Pe

Fastest arithmetic
uses floating-point
fp adds, fp subs, f

Each Pentium M c
≤ 1 fp op.

Point multiplicatio
589825 fp ops; $\approx$

Understand cycle c
by simply counting

Immune to timing attacks,
including cache-timing attacks,
including hyperthreading attacks.
No data-dependent branches;
no data-dependent indexing.

Software is in public domain.
16 kilobytes when compiled.
`cr.yp.to/ecdh.html`

No known patent problems.

For comparison, Brown et al.:
much smaller prime, $2^{192} - 2^{64} - 1$;
780000 PII cycles; y given;
no timing-attack protection.

Where are the cycles going?

Focus today on Pentium M.

Fastest arithmetic on Pentium M
uses floating-point operations:
fp adds, fp subs, fp mults.

Each Pentium M cycle does
≤ 1 fp op.

Point multiplication: 640838 cycles.
589825 fp ops; ≈ 0.92 per cycle.

Understand cycle counts fairly well
by simply counting fp ops.

attacks,
ming attacks,
eading attacks.
t branches;
t indexing.

lic domain.
compiled.
ntml

problems.

rown et al.:
ne, $2^{192} - 2^{64} - 1$;
y given;
protection.

Where are the cycles going?

Focus today on Pentium M.

Fastest arithmetic on Pentium M uses floating-point operations: fp adds, fp subs, fp mults.

Each Pentium M cycle does ≤ 1 fp op.

Point multiplication: 640838 cycles. 589825 fp ops; ≈ 0.92 per cycle.

Understand cycle counts fairly well by simply counting fp ops.

Avoiding all time v
to stop timing att

1. For $b \in \{0,1\}$,
as $bx[1] + (1-b)x$
Avoids data-depen
Costs 36210 fp op

2. Compute final
by Fermat, not ext
Avoids data-depen

3. Don't branch fo
Allow non-least re
No cost—this save

Where are the cycles going?

Focus today on Pentium M.

Fastest arithmetic on Pentium M uses floating-point operations: fp adds, fp subs, fp mults.

Each Pentium M cycle does ≤ 1 fp op.

Point multiplication: 640838 cycles. 589825 fp ops; ≈ 0.92 per cycle.

Understand cycle counts fairly well by simply counting fp ops.

Avoiding all time variability to stop timing attacks:

1. For $b \in \{0, 1\}$, compute $x[b]$ as $bx[1] + (1 - b)x[0]$ or similar. Avoids data-dependent indexing. Costs 36210 fp ops (6%).

2. Compute final reciprocal by Fermat, not extended Euclid. Avoids data-dependent branching.

3. Don't branch for remainders. Allow non-least remainders. No cost—this saves time!

les going?

entium M.

on Pentium M

operations:

p mults.

cycle does

on: 640838 cycles.

0.92 per cycle.

counts fairly well

g fp ops.

Avoiding all time variability
to stop timing attacks:

1. For $b \in \{0, 1\}$, compute $x[b]$ as $bx[1] + (1 - b)x[0]$ or similar. Avoids data-dependent indexing. Costs 36210 fp ops (6%).

2. Compute final reciprocal by Fermat, not extended Euclid. Avoids data-dependent branching.

3. Don't branch for remainders. Allow non-least remainders. No cost—this saves time!

Main loop: 54570

2140 times 255 ite

Reciprocal: 43821

$41148 = 254 \cdot 162$

$2673 = 11 \cdot 243$ fo

Additional work: 3

Inside one main-lo

$80 = 8 \cdot 10$ for 8 a

55 for mult by 121

$648 = 4 \cdot 162$ for 4

$1215 = 5 \cdot 243$ for

142 for $bx[1] + (1$

Avoiding all time variability
to stop timing attacks:

1. For $b \in \{0, 1\}$, compute $x[b]$
as $bx[1] + (1 - b)x[0]$ or similar.
Avoids data-dependent indexing.
Costs 36210 fp ops (6%).

2. Compute final reciprocal
by Fermat, not extended Euclid.
Avoids data-dependent branching.

3. Don't branch for remainders.
Allow non-least remainders.
No cost—this saves time!

Main loop: 545700 fp ops (92.5%).
2140 times 255 iterations.

Reciprocal: 43821 fp ops (7.4%).
$41148 = 254 \cdot 162$ for 254 squarings;
$2673 = 11 \cdot 243$ for 11 more mults.

Additional work: 304 fp ops.

Inside one main-loop iteration:
$80 = 8 \cdot 10$ for 8 adds/subs;
55 for mult by 121665;
$648 = 4 \cdot 162$ for 4 squarings;
$1215 = 5 \cdot 243$ for 5 more mults;
142 for $bx[1] + (1 - b)x[0]$ etc.

variability
acks:

compute $x[b]$
$x[0]$ or similar.
ndent indexing.
s (6%).

reciprocal
tended Euclid.
ndent branching.

or remainders.
mainders.
es time!

Main loop: 545700 fp ops (92.5%).
2140 times 255 iterations.

Reciprocal: 43821 fp ops (7.4%).
$41148 = 254 \cdot 162$ for 254 squarings;
$2673 = 11 \cdot 243$ for 11 more mults.

Additional work: 304 fp ops.

Inside one main-loop iteration:
$80 = 8 \cdot 10$ for 8 adds/subs;
55 for mult by 121665;
$648 = 4 \cdot 162$ for 4 squarings;
$1215 = 5 \cdot 243$ for 5 more mults;
142 for $bx[1] + (1 - b)x[0]$ etc.

An integer mod 2^2
represented in radi
as a sum of 10 fp
in specified ranges

Add/sub: 10 fp ac
Delay reductions a

Mult: poly mult u
10^2 fp mults, 9^2 f
reduce using 9 fp
carry 11 times, eac
overall $2 \cdot 10^2 + 4$

Squaring: start wi
then eliminate 9^2
overall $1 \cdot 10^2 + 6$

Main loop: 545700 fp ops (92.5%).
2140 times 255 iterations.

Reciprocal: 43821 fp ops (7.4%).
$41148 = 254 \cdot 162$ for 254 squarings;
$2673 = 11 \cdot 243$ for 11 more mults.

Additional work: 304 fp ops.

Inside one main-loop iteration:
$80 = 8 \cdot 10$ for 8 adds/subs;
55 for mult by 121665;
$648 = 4 \cdot 162$ for 4 squarings;
$1215 = 5 \cdot 243$ for 5 more mults;
142 for $bx[1] + (1 - b)x[0]$ etc.

An integer mod $2^{255} - 19$ is
represented in radix $2^{25.5}$
as a sum of 10 fp numbers
in specified ranges.

Add/sub: 10 fp adds/subs.
Delay reductions and carries!

Mult: poly mult using
10^2 fp mults, 9^2 fp adds;
reduce using 9 fp mults, 9 fp adds;
carry 11 times, each 4 fp adds;
overall $2 \cdot 10^2 + 4 \cdot 10 + 3$ fp ops.

Squaring: start with 9 fp doublings;
then eliminate $9^2 + 9$ fp ops;
overall $1 \cdot 10^2 + 6 \cdot 10 + 2$ fp ops.

0 fp ops (92.5%).
erations.

fp ops (7.4%).
for 254 squarings;
or 11 more mults.

304 fp ops.

op iteration:
dds/subs;
.665;
4 squarings;
5 more mults;
$-b)x[0]$ etc.

An integer mod $2^{255}-19$ is
represented in radix $2^{25.5}$
as a sum of 10 fp numbers
in specified ranges.

Add/sub: 10 fp adds/subs.
Delay reductions and carries!

Mult: poly mult using
10^2 fp mults, 9^2 fp adds;
reduce using 9 fp mults, 9 fp adds;
carry 11 times, each 4 fp adds;
overall $2 \cdot 10^2 + 4 \cdot 10 + 3$ fp ops.

Squaring: start with 9 fp doublings;
then eliminate $9^2 + 9$ fp ops;
overall $1 \cdot 10^2 + 6 \cdot 10 + 2$ fp ops.

<u>How was the prim</u>

Use prime close to
to save time in fie

Also reduces NFS
so would need larg
traditional discret
but doesn't seem

Use prime not far
to avoid wasting b

Comfortable secur
$2^{253}+39$, $2^{253}+$
$2^{255}-31$, $2^{255}-$

An integer mod $2^{255} - 19$ is represented in radix $2^{25.5}$ as a sum of 10 fp numbers in specified ranges.

Add/sub: 10 fp adds/subs. Delay reductions and carries!

Mult: poly mult using 10^2 fp mults, 9^2 fp adds; reduce using 9 fp mults, 9 fp adds; carry 11 times, each 4 fp adds; overall $2 \cdot 10^2 + 4 \cdot 10 + 3$ fp ops.

Squaring: start with 9 fp doublings; then eliminate $9^2 + 9$ fp ops; overall $1 \cdot 10^2 + 6 \cdot 10 + 2$ fp ops.

How was the prime chosen?

Use prime close to power of 2 to save time in field operations.

Also reduces NFS exponent, so would need larger prime for traditional discrete-log systems; but doesn't seem to affect ECDL.

Use prime not far below 2^{32k} to avoid wasting bandwidth.

Comfortable security, $k = 8$: $2^{253} + 39$, $2^{253} + 51$, $2^{254} + 79$, $2^{255} - 31$, $2^{255} - 19$, $2^{255} + 95$.

$^{255} - 19$ is

ix $2^{25.5}$

numbers

.

dds/subs.

nd carries!

sing

o adds;

mults, 9 fp adds;

ch 4 fp adds;

$\cdot 10 + 3$ fp ops.

th 9 fp doublings;

$+ 9$ fp ops;

$\cdot 10 + 2$ fp ops.

How was the prime chosen?

Use prime close to power of 2 to save time in field operations.

Also reduces NFS exponent, so would need larger prime for traditional discrete-log systems; but doesn't seem to affect ECDL.

Use prime not far below 2^{32k} to avoid wasting bandwidth.

Comfortable security, $k = 8$:
$2^{253} + 39$, $2^{253} + 51$, $2^{254} + 79$,
$2^{255} - 31$, $2^{255} - 19$, $2^{255} + 95$.

Bender, Castagnol

"$2^{127} + 24933$ is p

. . . For this curve

convenient in com

we also give . . ."

I use the prime 2^{25}

convenient for the

No trouble from "

patent 5159632 fil

How was the prime chosen?

Use prime close to power of 2 to save time in field operations.

Also reduces NFS exponent, so would need larger prime for traditional discrete-log systems; but doesn't seem to affect ECDL.

Use prime not far below 2^{32k} to avoid wasting bandwidth.

Comfortable security, $k = 8$:
$2^{253} + 39$, $2^{253} + 51$, $2^{254} + 79$,
$2^{255} - 31$, $2^{255} - 19$, $2^{255} + 95$.

Bender, Castagnoli, CRYPTO '89:

"$2^{127} + 24933$ is prime.
. . . For this curve which is convenient in computer arithmetic we also give . . ."

I use the prime $2^{255} - 19$, convenient for the same reasons. No trouble from "shift and add" patent 5159632 filed 1991.09.17.

e chosen?

power of 2

ld operations.

exponent,

ger prime for

e-log systems;

to affect ECDL.

below 2^{32k}

bandwidth.

ity, $k = 8$:

51, $2^{254} + 79$,

19, $2^{255} + 95$.

Bender, Castagnoli, CRYPTO '89:

"$2^{127} + 24933$ is prime.
. . . For this curve which is convenient in computer arithmetic we also give . . ."

I use the prime $2^{255} - 19$, convenient for the same reasons. No trouble from "shift and add" patent 5159632 filed 1991.09.17.

How was the curve

Use Montgomery s

$y^2 = x^3 + Ax^2 +$

to save time in cu

and to avoid squa

Choose $(A - 2)/4$

to save time in cu

Montgomery's rec

$z_1 = 1$; $x_{2m} = (x$

$z_{2m} = 4x_m z_m (x_m^2$

$x_{2m+1} = 4(x_m x_m$

$z_{2m+1} = 4(x_m z_m$

then $n(K, \ldots) = ($

Bender, Castagnoli, CRYPTO '89:

"$2^{127} + 24933$ is prime.
... For this curve which is convenient in computer arithmetic we also give ..."

I use the prime $2^{255} - 19$, convenient for the same reasons. No trouble from "shift and add" patent 5159632 filed 1991.09.17.

How was the curve chosen?

Use Montgomery shape
$y^2 = x^3 + Ax^2 + x$
to save time in curve operations and to avoid square roots.

Choose $(A-2)/4$ as small integer to save time in curve operations.

Montgomery's recursion: $x_1 = K$;
$z_1 = 1$; $x_{2m} = (x_m^2 - z_m^2)^2$;
$z_{2m} = 4x_m z_m (x_m^2 + Ax_m z_m + z_m^2)$;
$x_{2m+1} = 4(x_m x_{m+1} - z_m z_{m+1})^2$;
$z_{2m+1} = 4(x_m z_{m+1} - z_m x_{m+1})^2 K$;
then $n(K, \ldots) = (x_n/z_n, \ldots)$.

i, CRYPTO '89:

rime.

which is

puter arithmetic

$^{55} - 19$,

same reasons.

shift and add”

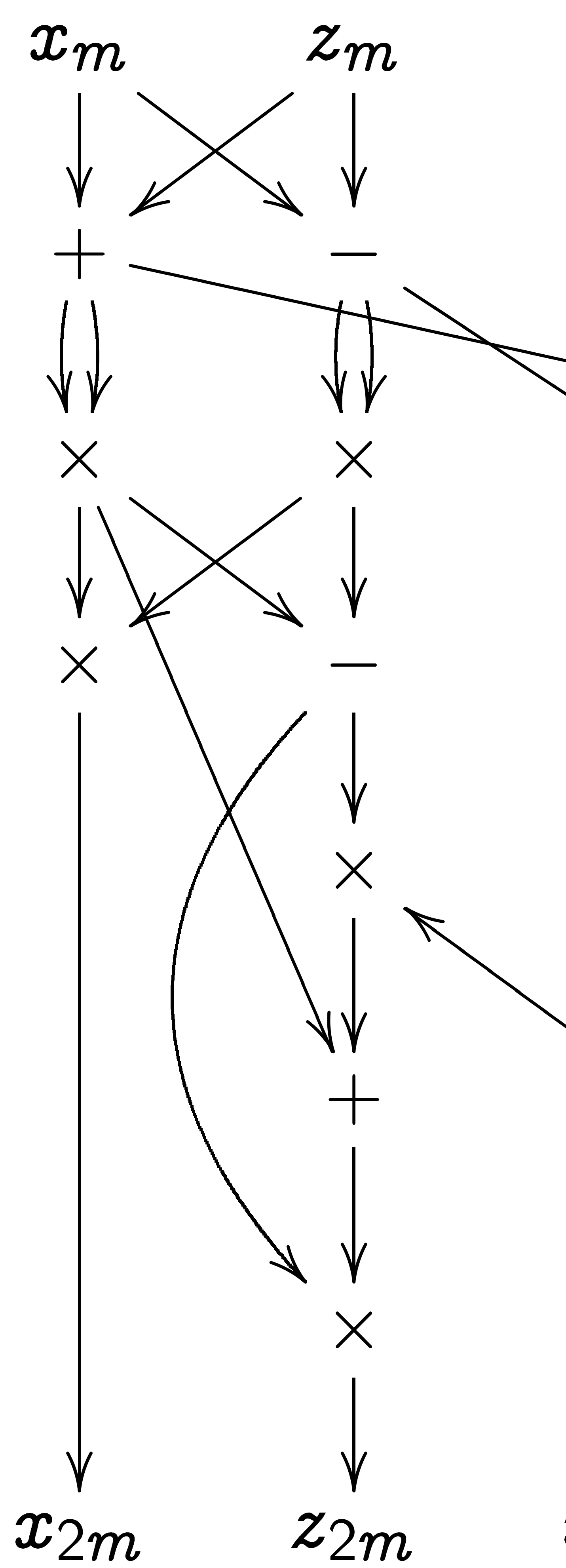
ed 1991.09.17.

How was the curve chosen?

Use Montgomery shape
$y^2 = x^3 + Ax^2 + x$
to save time in curve operations
and to avoid square roots.

Choose $(A - 2)/4$ as small integer
to save time in curve operations.

Montgomery's recursion: $x_1 = K$;
$z_1 = 1$; $x_{2m} = (x_m^2 - z_m^2)^2$;
$z_{2m} = 4x_m z_m(x_m^2 + Ax_m z_m + z_m^2)$;
$x_{2m+1} = 4(x_m x_{m+1} - z_m z_{m+1})^2$;
$z_{2m+1} = 4(x_m z_{m+1} - z_m x_{m+1})^2 K$;
then $n(K, \ldots) = (x_n/z_n, \ldots)$.

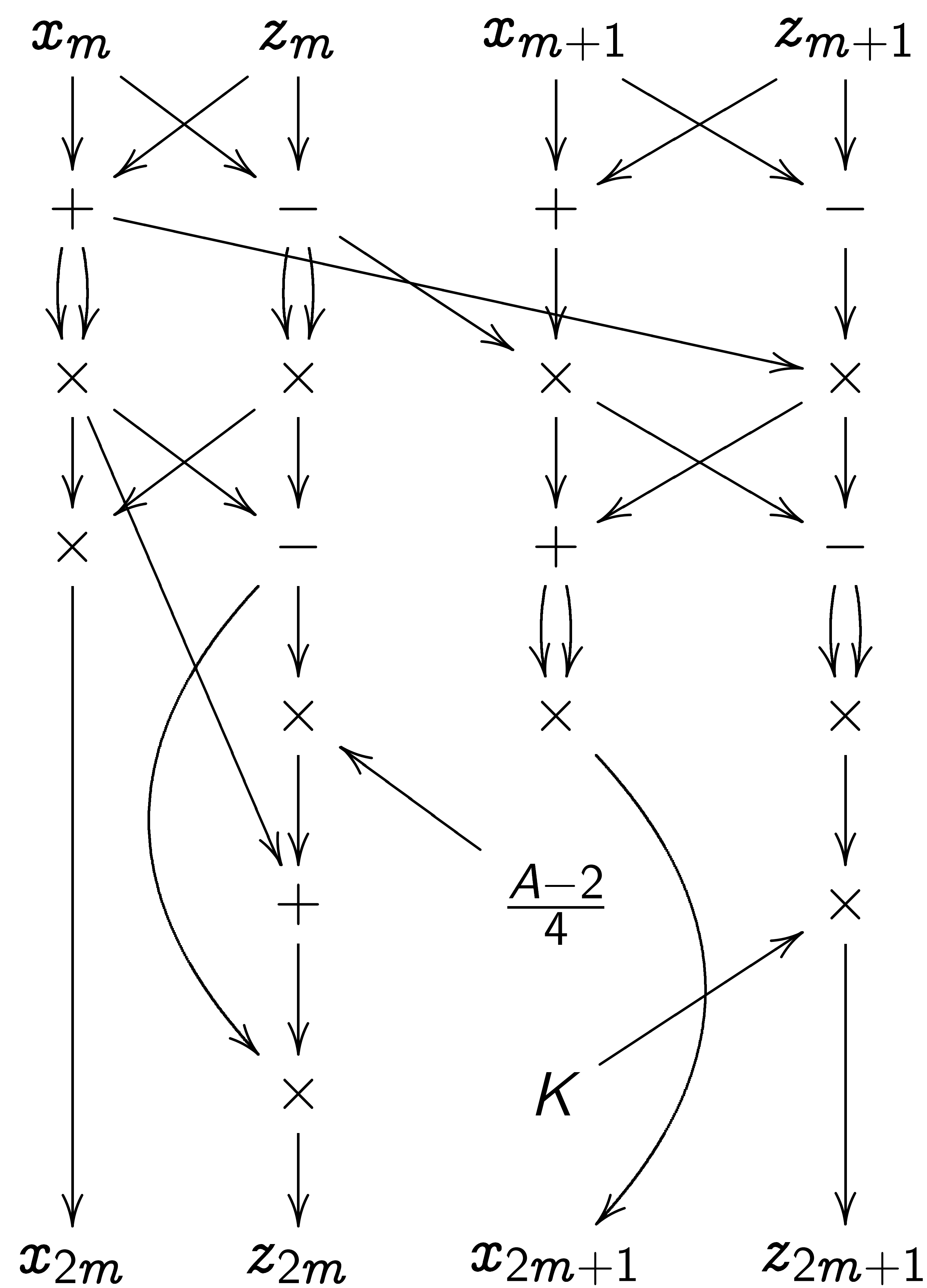


How was the curve chosen?

Use Montgomery shape
$y^2 = x^3 + Ax^2 + x$
to save time in curve operations
and to avoid square roots.

Choose $(A-2)/4$ as small integer
to save time in curve operations.

Montgomery's recursion: $x_1 = K$;
$z_1 = 1$; $x_{2m} = (x_m^2 - z_m^2)^2$;
$z_{2m} = 4x_m z_m(x_m^2 + Ax_m z_m + z_m^2)$;
$x_{2m+1} = 4(x_m x_{m+1} - z_m z_{m+1})^2$;
$z_{2m+1} = 4(x_m z_{m+1} - z_m x_{m+1})^2 K$;
then $n(K, \ldots) = (x_n/z_n, \ldots)$.



e chosen?

shape

x

rve operations

re roots.

as small integer

rve operations.

ursion: $x_1 = K$;

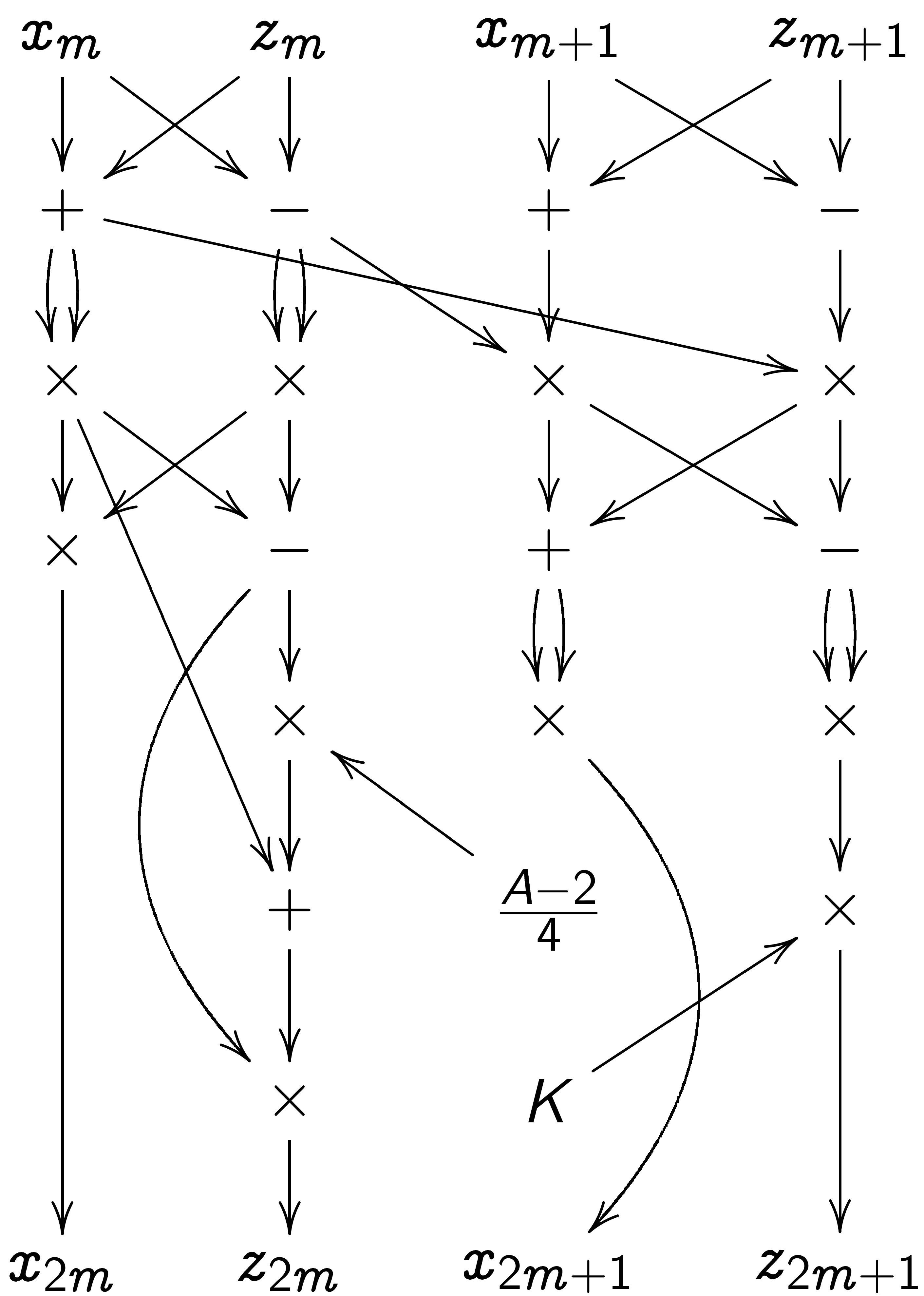
$_m^2 - z_m^2)^2$;

$_m + Ax_m z_m + z_m^2)$;

$_{+1} - z_m z_{m+1})^2$;

$_{+1} - z_m x_{m+1})^2 K$;

$(x_n/z_n, \ldots)$.



Reject A unless cu

orders are $\{4 \cdot$ prin

Montgomery shap

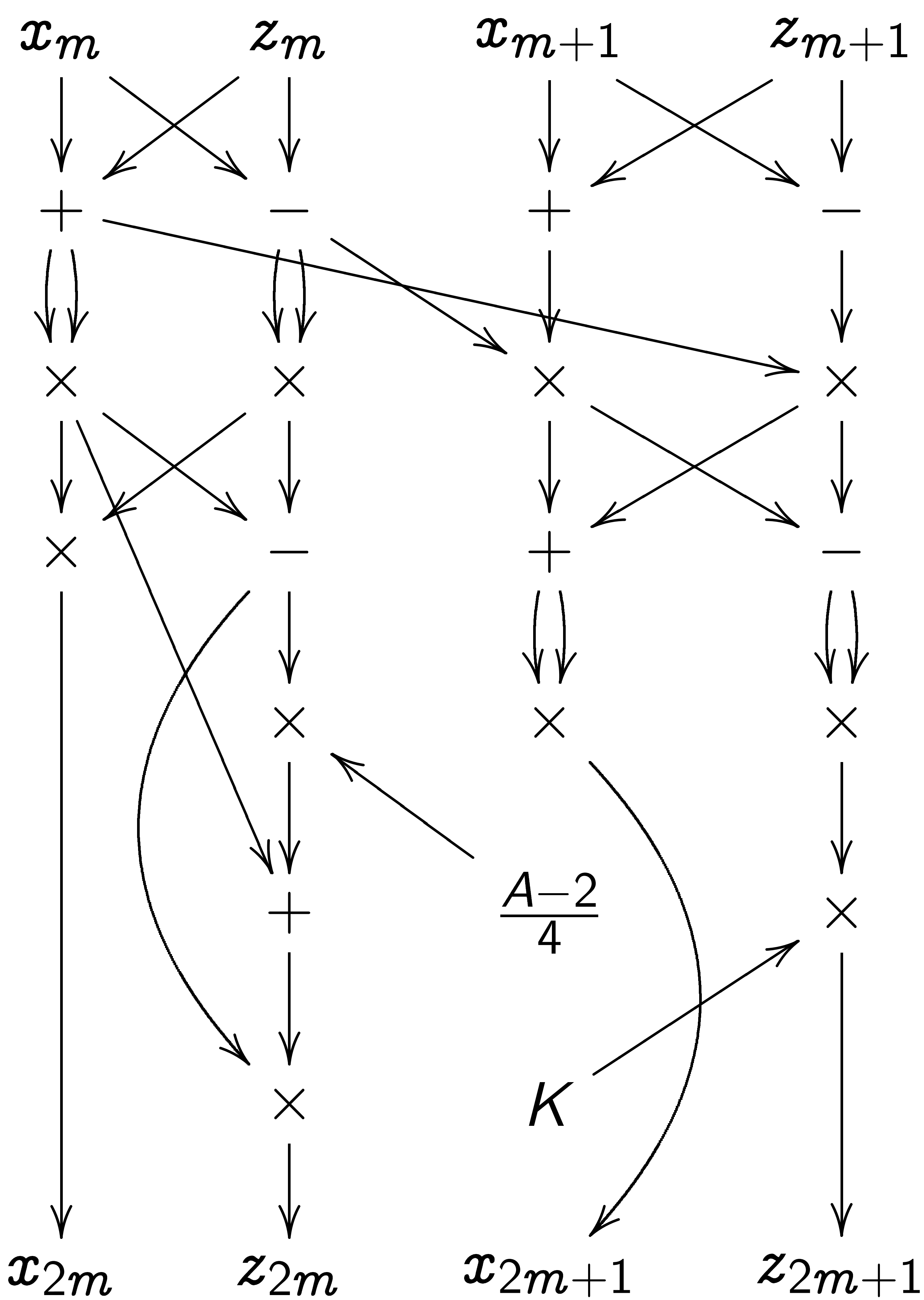
characteristic in 4

For $A = 486662$:

8 times prime $p_1 =$

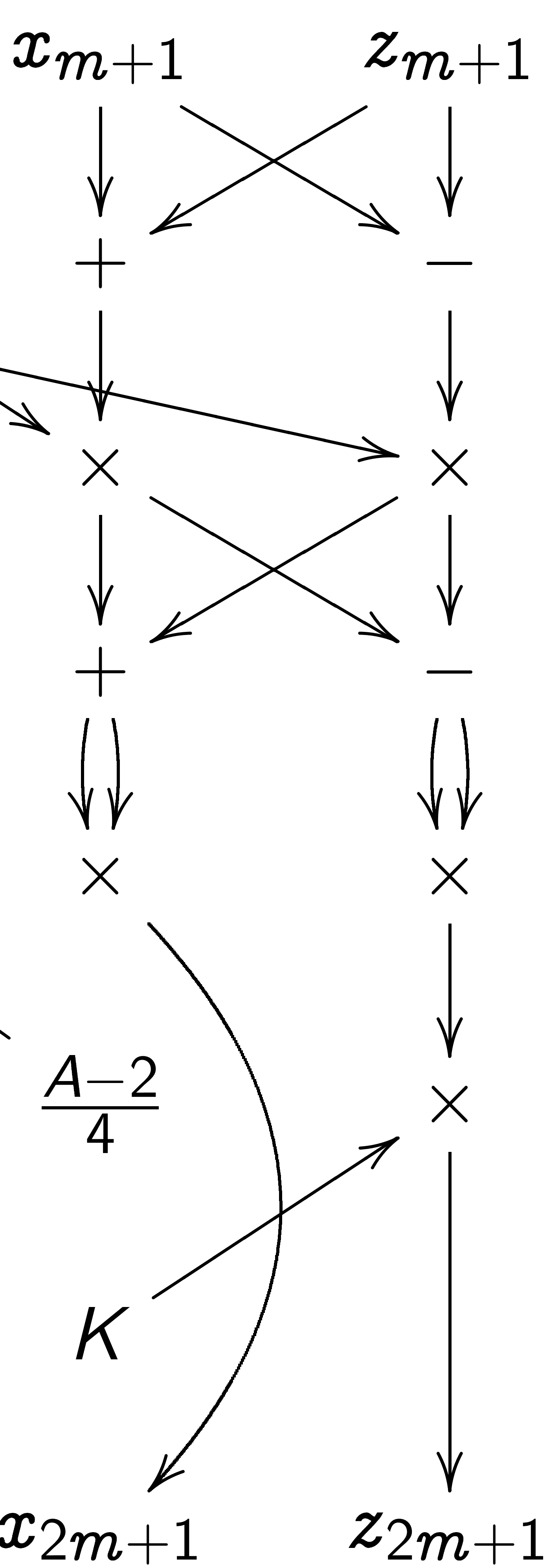
The twist has orde

4 times prime $p_2 =$



Reject A unless curve and twist orders are $\{4 \cdot \text{prime}, 8 \cdot \text{prime}\}$.
Montgomery shape forces 4; characteristic in $4\mathbf{Z}+1$ forces 4, 8.

For $A = 486662$: Curve has order 8 times prime $p_1 = 2^{252} + \cdots$.
The twist has order 4 times prime $p_2 = 2^{253} - \cdots$.



Reject A unless curve and twist orders are $\{4 \cdot \text{prime}, 8 \cdot \text{prime}\}$.
Montgomery shape forces 4; characteristic in $4\mathbf{Z} + 1$ forces $4, 8$.

For $A = 486662$: Curve has order 8 times prime $p_1 = 2^{252} + \cdots$.
The twist has order 4 times prime $p_2 = 2^{253} - \cdots$.

For $A = 358990$:
One prime is 2^{252}
so user's secret ke
$n \in 2^{254} + 8\{0, 1,$
could be 8 times t
Extremely unlikely,
but annoys implem
so reject this A.

Reject A unless curve and twist orders are $\{4 \cdot \text{prime}, 8 \cdot \text{prime}\}$. Montgomery shape forces 4; characteristic in $4\mathbf{Z}+1$ forces $4, 8$.

For $A = 486662$: Curve has order 8 times prime $p_1 = 2^{252} + \cdots$. The twist has order 4 times prime $p_2 = 2^{253} - \cdots$.

For $A = 358990$: One prime is $2^{252} - \cdots$, so user's secret key $n \in 2^{254} + 8\{0, 1, \ldots, 2^{251} - 1\}$ could be 8 times that prime. Extremely unlikely, but annoys implementors, so reject this A.

rve and twist

ne, $8 \cdot$ prime}.

e forces 4;

$\mathbf{Z} + 1$ forces $4, 8$.

Curve has order

$= 2^{252} + \cdots$.

er

$= 2^{253} - \cdots$.

For $A = 358990$:

One prime is $2^{252} - \cdots$,

so user's secret key

$n \in 2^{254} + 8\{0, 1, \ldots, 2^{251} - 1\}$

could be 8 times that prime.

Extremely unlikely,

but annoys implementors,

so reject this A.

Note on comparin

and comparing co

Count fp ops, not

Otherwise you ma

Reality: mult by s

is as expensive as

Reality: square-to-

is $2/3$ for this fiel

Reality: $a^2 + b^2 +$

faster than $(a^2, b^2$

For $A = 358990$:

One prime is $2^{252} - \cdots$,

so user's secret key

$n \in 2^{254} + 8\{0, 1, \ldots, 2^{251} - 1\}$

could be 8 times that prime.

Extremely unlikely,

but annoys implementors,

so reject this A.

Note on comparing curves

and comparing coordinate systems:

Count fp ops, not field ops!

Otherwise you make bad choices.

Reality: mult by small constant

is as expensive as several adds.

Reality: square-to-multiply ratio

is $2/3$ for this field, not $4/5$.

Reality: $a^2 + b^2 + c^2$ is

faster than (a^2, b^2, c^2).

$- \cdots,$
y
$\ldots, 2^{251} - 1\}$
hat prime.
,
nentors,

Note on comparing curves
and comparing coordinate systems:
Count fp ops, not field ops!
Otherwise you make bad choices.

Reality: mult by small constant
is as expensive as several adds.

Reality: square-to-multiply ratio
is 2/3 for this field, not 4/5.

Reality: $a^2 + b^2 + c^2$ is
faster than (a^2, b^2, c^2).

How was the key r

Public key for secr
is x-coordinate of
of standard base p

Base-point order is
so uniform random
$2^{251} + \{0, 1, 2, \ldots$
produces almost ex
random public key
among $\approx 2^{251}$ pos

The addition of 2^{2}
and avoids timing

Note on comparing curves and comparing coordinate systems: Count fp ops, not field ops! Otherwise you make bad choices.

Reality: mult by small constant is as expensive as several adds.

Reality: square-to-multiply ratio is $2/3$ for this field, not $4/5$.

Reality: $a^2 + b^2 + c^2$ is faster than (a^2, b^2, c^2).

<u>How was the key range chosen?</u>

Public key for secret key n is x-coordinate of nth multiple of standard base point $(9, \ldots)$.

Base-point order is $p_1 \approx 2^{252}$, so uniform random n in $2^{251} + \{0, 1, 2, \ldots, 2^{251} - 1\}$ produces almost exactly uniform random public key from among $\approx 2^{251}$ possibilities.

The addition of 2^{251} avoids ∞ and avoids timing attacks.

g curves
ordinate systems:
field ops!
ke bad choices.

mall constant
several adds.

-multiply ratio
d, not $4/5$.

c^2 is
$, c^2)$.

How was the key range chosen?

Public key for secret key n is x-coordinate of nth multiple of standard base point $(9, \ldots)$.

Base-point order is $p_1 \approx 2^{252}$, so uniform random n in $2^{251} + \{0, 1, 2, \ldots, 2^{251} - 1\}$ produces almost exactly uniform random public key from among $\approx 2^{251}$ possibilities.

The addition of 2^{251} avoids ∞ and avoids timing attacks.

Miller, CRYPTO '

"For the key excha
only the x-coordin
transmitted. The
multiples of a poin
first section make
x-coordinate of a
only on the x-coor
original point."

This is the compre
use. No trouble fr
compression" pate
1994.07.29.

How was the key range chosen?

Public key for secret key n is x-coordinate of nth multiple of standard base point $(9, \ldots)$.

Base-point order is $p_1 \approx 2^{252}$, so uniform random n in $2^{251} + \{0, 1, 2, \ldots, 2^{251} - 1\}$ produces almost exactly uniform random public key from among $\approx 2^{251}$ possibilities.

The addition of 2^{251} avoids ∞ and avoids timing attacks.

Miller, CRYPTO '85:

"For the key exchange . . . only the x-coordinate needs to be transmitted. The formulas for multiples of a point cited in the first section make it clear that the x-coordinate of a multiple depends only on the x-coordinate of the original point."

This is the compression method I use. No trouble from "point compression" patent 6141420 filed 1994.07.29.

range chosen?

ret key n
nth multiple
point $(9, \ldots)$.

s $p_1 \approx 2^{252}$,
n n in
$, 2^{251} - 1\}$
xactly uniform
r from
ssibilities.

^{251} avoids ∞
attacks.

Miller, CRYPTO '85:

"For the key exchange ...
only the x-coordinate needs to be transmitted. The formulas for multiples of a point cited in the first section make it clear that the x-coordinate of a multiple depends only on the x-coordinate of the original point."

This is the compression method I use. No trouble from "point compression" patent 6141420 filed 1994.07.29.

Insert factor of 8 i
in case $(K, \ldots)$ is
in this group of or

Three possibilities
∞, output as 0;
or a nontrivial poi
in the desired prim
or a nontrivial poi
in the twist prime

Don't spend time
"validating" K, i.e
checking it's in de

Miller, CRYPTO '85:

"For the key exchange . . . only the x-coordinate needs to be transmitted. The formulas for multiples of a point cited in the first section make it clear that the x-coordinate of a multiple depends only on the x-coordinate of the original point."

This is the compression method I use. No trouble from "point compression" patent 6141420 filed 1994.07.29.

Insert factor of 8 into n in case $(K, \ldots)$ is not actually in this group of order p_1.

Three possibilities for $8(K, \ldots)$:
∞, output as 0;
or a nontrivial point in the desired prime group;
or a nontrivial point in the twist prime group.

Don't spend time "validating" K, i.e., checking it's in desired group.

85:

ange . . .
ate needs to be
formulas for
nt cited in the
it clear that the
multiple depends
rdinate of the

ession method I
om "point
ent 6141420 filed

Insert factor of 8 into n in case $(K, \ldots)$ is not actually in this group of order p_1.

Three possibilities for $8(K, \ldots)$: ∞, output as 0; or a nontrivial point in the desired prime group; or a nontrivial point in the twist prime group.

Don't spend time "validating" K, i.e., checking it's in desired group.

Even if attacker w
same n times poir
would still need to
hash-Diffie-Hellma
of these two prime

For uniform rando
provably requires b
at least one of the
Curve and twist bo

No known way to
limited exponent r
Often used in Diffi
for multiplicative g

Insert factor of 8 into n
in case $(K, \ldots)$ is not actually
in this group of order p_1.

Three possibilities for $8(K, \ldots)$:
∞, output as 0;
or a nontrivial point
in the desired prime group;
or a nontrivial point
in the twist prime group.

Don't spend time
"validating" K, i.e.,
checking it's in desired group.

Even if attacker were given
same n times point on twist,
would still need to break
hash-Diffie-Hellman for product
of these two prime groups.

For uniform random exponent,
provably requires breaking
at least one of the prime groups.
Curve and twist both seem secure.

No known way to exploit
limited exponent range.
Often used in Diffie-Hellman
for multiplicative group.

nto n
not actually
der p_1.

for $8(K, \ldots)$:

nt
ne group;
nt
group.

e.,
sired group.

Even if attacker were given
same n times point on twist,
would still need to break
hash-Diffie-Hellman for product
of these two prime groups.

For uniform random exponent,
provably requires breaking
at least one of the prime groups.
Curve and twist both seem secure.

No known way to exploit
limited exponent range.
Often used in Diffie-Hellman
for multiplicative group.

Bernstein, sci.cryp

"You can happily s
transmission and t
In fact, if both the
twist have nearly p
you can even skip

I use a curve of th
No trouble from ru
"public-key validat
filed 2003.

Even if attacker were given same n times point on twist, would still need to break hash-Diffie-Hellman for product of these two prime groups.

For uniform random exponent, provably requires breaking at least one of the prime groups. Curve and twist both seem secure.

No known way to exploit limited exponent range. Often used in Diffie-Hellman for multiplicative group.

Bernstein, sci.crypt, 2001.11.09:

"You can happily skip both the y transmission and the square root. In fact, if both the curve and its twist have nearly prime order, then you can even skip square testing."

I use a curve of this type. No trouble from rumored new "public-key validation" patent filed 2003.

ere given
nt on twist,
break
n for product
e groups.

m exponent,
breaking
e prime groups.
oth seem secure.

exploit
ange.
ie-Hellman
group.

Bernstein, sci.crypt, 2001.11.09:

"You can happily skip both the y transmission and the square root. In fact, if both the curve and its twist have nearly prime order, then you can even skip square testing."

I use a curve of this type. No trouble from rumored new "public-key validation" patent filed 2003.

How was the softw

Common phenome
Write fp op seque
Feed it to C comp
to produce machi
Observe that cycl
is much larger tha
sometimes 5 or m

Have faith. Don't
Understand and el
non-fp-op cycles.

(I have more work
Athlon et al. Exp

Bernstein, sci.crypt, 2001.11.09:

"You can happily skip both the y transmission and the square root. In fact, if both the curve and its twist have nearly prime order, then you can even skip square testing."

I use a curve of this type.
No trouble from rumored new "public-key validation" patent filed 2003.

How was the software built?

Common phenomenon:
Write fp op sequence in C.
Feed it to C compiler to produce machine language.
Observe that cycles/fp ops is much larger than 1: sometimes 5 or more!

Have faith. Don't accept > 1.1.
Understand and eliminate non-fp-op cycles.

(I have more work to do here for Athlon et al. Expect speedups.)

t, 2001.11.09:

skip both the y
the square root.
e curve and its
prime order, then
square testing."

is type.
umored new
tion" patent

How was the software built?

Common phenomenon:
Write fp op sequence in C.
Feed it to C compiler
to produce machine language.
Observe that cycles/fp ops
is much larger than 1:
sometimes 5 or more!

Have faith. Don't accept > 1.1.
Understand and eliminate
non-fp-op cycles.

(I have more work to do here for
Athlon et al. Expect speedups.)

Some important d

- 3-cycle "load" la
  copying data fro
  "register" for ari
  Only 8 registers.
- 3-cycle fp add la
- 5-cycle fp mult l

An op waits if its i
aren't ready. CPU
ability to reorder o
uses greedy algorit

How was the software built?

Common phenomenon:
Write fp op sequence in C.
Feed it to C compiler
to produce machine language.
Observe that cycles/fp ops
is much larger than 1:
sometimes 5 or more!

Have faith. Don't accept > 1.1.
Understand and eliminate
non-fp-op cycles.

(I have more work to do here for
Athlon et al. Expect speedups.)

Some important delays:

- 3-cycle "load" latency, copying data from "cache" to "register" for arithmetic. Only 8 registers.
- 3-cycle fp add latency.
- 5-cycle fp mult latency.

An op waits if its inputs
aren't ready. CPU has some
ability to reorder ops, but
uses greedy algorithm; suboptimal.

vare built?

enon:

nce in C.

iler

ne language.

es/fp ops

n 1:

ore!

accept > 1.1.

iminate

to do here for

ect speedups.)

Some important delays:

- 3-cycle "load" latency, copying data from "cache" to "register" for arithmetic. Only 8 registers.
- 3-cycle fp add latency.
- 5-cycle fp mult latency.

An op waits if its inputs aren't ready. CPU has some ability to reorder ops, but uses greedy algorithm; suboptimal.

Can't rely on C co

to sensibly permut

Sometimes $r \leftarrow a$

$r \leftarrow r + c$; $r \leftarrow r$

a sequence of exac

best done as, e.g.,

$s \leftarrow a + c$; $r \leftarrow r$

But sometimes $r \leftarrow$

is a non-associativ

deliberately round

The C language ha

to express this dist

Some important delays:

- 3-cycle "load" latency, copying data from "cache" to "register" for arithmetic. Only 8 registers.
- 3-cycle fp add latency.
- 5-cycle fp mult latency.

An op waits if its inputs aren't ready. CPU has some ability to reorder ops, but uses greedy algorithm; suboptimal.

Can't rely on C compiler to sensibly permute fp ops.

Sometimes $r \leftarrow a + b$; $r \leftarrow r + c$; $r \leftarrow r + d$ is a sequence of exact fp adds best done as, e.g., $r \leftarrow b + d$; $s \leftarrow a + c$; $r \leftarrow r + s$.

But sometimes $r \leftarrow r + c$ is a non-associative deliberately rounded fp add!

The C language has no way to express this distinction.

elays:

atency,
m "cache" to
ithmetic.

tency.
atency.

inputs
has some
ps, but
thm; suboptimal.

Can't rely on C compiler to sensibly permute fp ops.

Sometimes $r \leftarrow a + b$; $r \leftarrow r + c$; $r \leftarrow r + d$ is a sequence of exact fp adds best done as, e.g., $r \leftarrow b + d$; $s \leftarrow a + c$; $r \leftarrow r + s$.

But sometimes $r \leftarrow r + c$ is a non-associative deliberately rounded fp add!

The C language has no way to express this distinction.

Curve25519 imple
is actually in qhas
new programming
for high-speed com

Language allows d
and propagation o
guided register all

Lets me write desi
with much less hu
traditional asm an
Have also used fo
fast Poly1305, fas

Can't rely on C compiler to sensibly permute fp ops.

Sometimes $r \leftarrow a + b$; $r \leftarrow r + c$; $r \leftarrow r + d$ is a sequence of exact fp adds best done as, e.g., $r \leftarrow b + d$; $s \leftarrow a + c$; $r \leftarrow r + s$.

But sometimes $r \leftarrow r + c$ is a non-associative deliberately rounded fp add!

The C language has no way to express this distinction.

Curve25519 implementation is actually in `qhasm`, new programming language for high-speed computations.

Language allows declaration and propagation of fp ranges; guided register allocation; et al.

Lets me write desired code with much less human time than traditional asm and C compiler. Have also used for fast AES, fast Poly1305, fast Salsa20, etc.

ompiler
te fp ops.

$+ b$;
$+ d$ is
ct fp adds
$r \leftarrow b + d$;
$+ s$.

$\leftarrow r + c$
e
ed fp add!

as no way
tinction.

Curve25519 implementation is actually in `qhasm`, new programming language for high-speed computations.

Language allows declaration and propagation of fp ranges; guided register allocation; et al.

Lets me write desired code with much less human time than traditional asm and C compiler. Have also used for fast AES, fast Poly1305, fast Salsa20, etc.

What's next?

Culmination of ext
on eliminating fiel
genus-2 hyperellip
25 mults per bit.
`eprint.iacr.org`

Half-size prime: e.
Select curve to ma
mults easier, like c

Should count fp op
Prediction: this wi

Curve25519 implementation is actually in `qhasm`, new programming language for high-speed computations.

Language allows declaration and propagation of fp ranges; guided register allocation; et al.

Lets me write desired code with much less human time than traditional asm and C compiler. Have also used for fast AES, fast Poly1305, fast Salsa20, etc.

What's next?

Culmination of extensive work on eliminating field mults for genus-2 hyperelliptic curves: 25 mults per bit. Gaudry, `eprint.iacr.org/2005/314`

Half-size prime: e.g., $2^{127} - 1$. Select curve to make some mults easier, like choosing A.

Should count fp ops instead. Prediction: this will beat genus 1.