High-speed elliptic-curve cryptography

D. J. Bernstein

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

Define $p = 2^{255} - 19$; prime.
Define $A = 358990$. Define
$\text{Curve} : \mathbf{Z} \to \{0, 1, \ldots, p-1, \infty\}$ by
$n \mapsto x$ coordinate of nth multiple
of $(2, \ldots)$ on the elliptic curve
$y^2 = x^3 + Ax^2 + x$ over $\mathbf{F}_p$.

Main topic of this talk: Compute
$U, \text{Curve}(V) \mapsto \text{Curve}(UV)$
in very few CPU cycles.
In particular, use floating point
for fast arithmetic mod p.

tography

is at Chicago

0

oundation

Define $p = 2^{255} - 19$; prime.

Define $A = 358990$. Define $\text{Curve} : \mathbf{Z} \to \{0, 1, \ldots, p-1, \infty\}$ by $n \mapsto x$ coordinate of nth multiple of $(2, \ldots)$ on the elliptic curve $y^2 = x^3 + Ax^2 + x$ over $\mathbf{F}_p$.

Main topic of this talk: Compute $U, \text{Curve}(V) \mapsto \text{Curve}(UV)$ in very few CPU cycles.

In particular, use floating point for fast arithmetic mod p.

Why cryptographe

Each user has secr
public key Curve(U

Users with secret
exchange Curve(U
through an authen
compute Curve(UV
use hash as shared
encrypt and authe

Curve speed is imp
when number of m

Define $p = 2^{255} - 19$; prime.
Define $A = 358990$. Define
Curve : $\mathbf{Z} \to \{0, 1, \ldots, p-1, \infty\}$ by
$n \mapsto x$ coordinate of nth multiple
of $(2, \ldots)$ on the elliptic curve
$y^2 = x^3 + Ax^2 + x$ over $\mathbf{F}_p$.

Main topic of this talk: Compute
U, Curve(V) $\mapsto$ Curve(UV)
in very few CPU cycles.
In particular, use floating point
for fast arithmetic mod p.

Why cryptographers care

Each user has secret key U,
public key Curve(U).

Users with secret keys U, V
exchange Curve(U), Curve(V)
through an authenticated channel;
compute Curve(UV); hash it;
use hash as shared secret to
encrypt and authenticate messages.

Curve speed is important
when number of messages is small.

19; prime.
0. Define
$\ldots, p-1, \infty\}$ by
of nth multiple
elliptic curve
x over $\mathbf{F}_p$.

talk: Compute
rve(UV)
ycles.
loating point
mod p.

<u>Why cryptographers care</u>

Each user has secret key U,
public key Curve(U).

Users with secret keys U, V
exchange Curve(U), Curve(V)
through an authenticated channel;
compute Curve(UV); hash it;
use hash as shared secret to
encrypt and authenticate messages.

Curve speed is important
when number of messages is small.

Analogous system
1976 Diffie Hellma

Using elliptic curve
to avoid index-calc
1986 Miller, 1987

Using $x^3 + Ax^2 +$
1987 Montgomery

High precision fro
1968 Veltkamp, 19
Speedups: 1999–2

Why cryptographers care

Each user has secret key U,
public key Curve(U).

Users with secret keys U, V
exchange Curve(U), Curve(V)
through an authenticated channel;
compute Curve(UV); hash it;
use hash as shared secret to
encrypt and authenticate messages.

Curve speed is important
when number of messages is small.

Analogous system using 2^U mod p:
1976 Diffie Hellman.

Using elliptic curves
to avoid index-calculus attacks:
1986 Miller, 1987 Koblitz.

Using $x^3 + Ax^2 + x$ for speed:
1987 Montgomery (for ECM).

High precision from fp sums:
1968 Veltkamp, 1971 Dekker.
Speedups: 1999–2005 Bernstein.

ers care

ret key U,
U).

keys U, V
), Curve(V)
nticated channel;
V); hash it;
l secret to
nticate messages.

portant
nessages is small.

Analogous system using 2^U mod p:
1976 Diffie Hellman.

Using elliptic curves
to avoid index-calculus attacks:
1986 Miller, 1987 Koblitz.

Using $x^3 + Ax^2 + x$ for speed:
1987 Montgomery (for ECM).

High precision from fp sums:
1968 Veltkamp, 1971 Dekker.
Speedups: 1999–2005 Bernstein.

Understanding CP

Computers are des
music, movies, Ph
etc. Heavy use of
i.e., approximate r

Example: Athlon,
does one add and
of high-precision fp

Programmer payin
to these CPU feat
can use them for c

Analogous system using $2^U \bmod p$:
1976 Diffie Hellman.

Using elliptic curves
to avoid index-calculus attacks:
1986 Miller, 1987 Koblitz.

Using $x^3 + Ax^2 + x$ for speed:
1987 Montgomery (for ECM).

High precision from fp sums:
1968 Veltkamp, 1971 Dekker.
Speedups: 1999–2005 Bernstein.

Understanding CPU design

Computers are designed for
music, movies, Photoshop, Doom 3,
etc. Heavy use of fp arithmetic,
i.e., approximate real arithmetic.

Example: Athlon, every cycle,
does one add and one multiply
of high-precision fp numbers.

Programmer paying attention
to these CPU features
can use them for cryptography.

using 2^U mod p:
an.

es
culus attacks:
Koblitz.

x for speed:
(for ECM).

m fp sums:
971 Dekker.
2005 Bernstein.

Understanding CPU design

Computers are designed for music, movies, Photoshop, Doom 3, etc. Heavy use of fp arithmetic, i.e., approximate real arithmetic.

Example: Athlon, every cycle, does one add and one multiply of high-precision fp numbers.

Programmer paying attention to these CPU features can use them for cryptography.

A **53-bit fp numb**
is a real number 2
with $e, f \in$ **Z** and

Round each real n
closest 53-bit fp n
Round halves to e

Examples:
$\mathrm{fp}_{53}(8675309) = 8$
$\mathrm{fp}_{53}(2^{127} + 867530$
$\mathrm{fp}_{53}(2^{127} - 867530$

Understanding CPU design

Computers are designed for music, movies, Photoshop, Doom 3, etc. Heavy use of fp arithmetic, i.e., approximate real arithmetic.

Example: Athlon, every cycle, does one add and one multiply of high-precision fp numbers.

Programmer paying attention to these CPU features can use them for cryptography.

A **53-bit fp number** is a real number $2^e f$ with $e, f \in \mathbf{Z}$ and $|f| \le 2^{53}$.

Round each real number z to closest 53-bit fp number, $\text{fp}_{53}\, z$. Round halves to even.

Examples:
$\text{fp}_{53}(8675309) = 8675309$;
$\text{fp}_{53}(2^{127} + 8675309) = 2^{127}$;
$\text{fp}_{53}(2^{127} - 8675309) = 2^{127}$.

U design

signed for
otoshop, Doom 3,
fp arithmetic,
eal arithmetic.

every cycle,
one multiply
p numbers.

g attention
ures
cryptography.

A **53-bit fp number**
is a real number $2^e f$
with $e, f \in \mathbf{Z}$ and $|f| \le 2^{53}$.

Round each real number z to
closest 53-bit fp number, $\text{fp}_{53}\, z$.
Round halves to even.

Examples:
$\text{fp}_{53}(8675309) = 8675309$;
$\text{fp}_{53}(2^{127} + 8675309) = 2^{127}$;
$\text{fp}_{53}(2^{127} - 8675309) = 2^{127}$.

Typical CPU: Ultr

Every cycle, UltraS
one fp multiplicati
$r, s \mapsto \text{fp}_{53}(rs)$
and one fp additio
$r, s \mapsto \text{fp}_{53}(r + s)$
subject to limits o

"4-cycle fp-operati
Results available a

Can substitute sub
for addition. I'll co
subtractions as ad

A **53-bit fp number** is a real number $2^e f$ with $e, f \in \mathbf{Z}$ and $|f| \le 2^{53}$.

Round each real number z to closest 53-bit fp number, $\mathrm{fp}_{53}\, z$. Round halves to even.

Examples:
$\mathrm{fp}_{53}(8675309) = 8675309$;
$\mathrm{fp}_{53}(2^{127} + 8675309) = 2^{127}$;
$\mathrm{fp}_{53}(2^{127} - 8675309) = 2^{127}$.

Typical CPU: UltraSPARC III.

Every cycle, UltraSPARC III can do one fp multiplication $r, s \mapsto \mathrm{fp}_{53}(rs)$ and one fp addition $r, s \mapsto \mathrm{fp}_{53}(r + s)$, subject to limits on e.

"4-cycle fp-operation latency": Results available after 4 cycles.

Can substitute subtraction for addition. I'll count subtractions as additions.

ber

$^{e} f$

$|f| \le 2^{53}$.

umber z to

umber, $\mathrm{fp}_{53}\, z$.

ven.

8675309;

$09) = 2^{127}$;

$09) = 2^{127}$.

Typical CPU: UltraSPARC III.

Every cycle, UltraSPARC III can do one fp multiplication $r, s \mapsto \mathrm{fp}_{53}(rs)$ and one fp addition $r, s \mapsto \mathrm{fp}_{53}(r + s)$, subject to limits on e.

"4-cycle fp-operation latency": Results available after 4 cycles.

Can substitute subtraction for addition. I'll count subtractions as additions.

Some variation am

PowerPC RS64 IV
or one multiplicati
"fused" $r, s, t \mapsto$ f
Results available a

Athlon: fp_{64} inste
one multiplication
Results available a

I'll focus on UltraS
Not the most imp
but it's a good wa

Typical CPU: UltraSPARC III.

Every cycle, UltraSPARC III can do one fp multiplication $r, s \mapsto \mathrm{fp}_{53}(rs)$ and one fp addition $r, s \mapsto \mathrm{fp}_{53}(r + s)$, subject to limits on e.

"4-cycle fp-operation latency": Results available after 4 cycles.

Can substitute subtraction for addition. I'll count subtractions as additions.

Some variation among CPUs.

PowerPC RS64 IV: One addition or one multiplication or one "fused" $r, s, t \mapsto \mathrm{fp}_{53}(rs + t)$. Results available after 4 cycles.

Athlon: fp_{64} instead of fp_{53}; one multiplication and one addition. Results available after 4 cycles.

I'll focus on UltraSPARC III. Not the most important CPU, but it's a good warmup.

aSPARC III.

SPARC III can do
on

n

'
n e.

ion latency":
fter 4 cycles.

otraction
ount
ditions.

Some variation among CPUs.

PowerPC RS64 IV: One addition or one multiplication or one "fused" $r, s, t \mapsto \text{fp}_{53}(rs + t)$. Results available after 4 cycles.

Athlon: fp_{64} instead of fp_{53}; one multiplication and one addition. Results available after 4 cycles.

I'll focus on UltraSPARC III. Not the most important CPU, but it's a good warmup.

Exact dot product

If $a, b \in \{-2^{20}, \ldots$
then ab is a 53-bit
so $ab = \text{fp}_{53}(ab)$.

If $a, b, c, d \in \{-2^{2}$
then $ab, cd, ab + c$
53-bit fp numbers
$ab = \text{fp}_{53}(ab)$, cd
$ab + cd = \text{fp}_{53}(ab$

UltraSPARC III co
$a, b, c, d \mapsto ab + c$
two fp mults, one

Some variation among CPUs.

PowerPC RS64 IV: One addition or one multiplication or one "fused" $r, s, t \mapsto \text{fp}_{53}(rs + t)$. Results available after 4 cycles.

Athlon: fp_{64} instead of fp_{53}; one multiplication and one addition. Results available after 4 cycles.

I'll focus on UltraSPARC III. Not the most important CPU, but it's a good warmup.

Exact dot products

If $a, b \in \{-2^{20}, \ldots, 0, 1, \ldots, 2^{20}\}$ then ab is a 53-bit fp number so $ab = \text{fp}_{53}(ab)$.

If $a, b, c, d \in \{-2^{20}, \ldots, 2^{20}\}$ then $ab, cd, ab + cd$ are 53-bit fp numbers so $ab = \text{fp}_{53}(ab)$, $cd = \text{fp}_{53}(cd)$, $ab + cd = \text{fp}_{53}(ab + cd)$.

UltraSPARC III computes $a, b, c, d \mapsto ab + cd$ with two fp mults, one fp add.

nong CPUs.

: One addition
on or one
$\text{p}_{53}(rs+t)$.
after 4 cycles.

ad of fp_{53};
and one addition.
after 4 cycles.

SPARC III.
ortant CPU,
rmup.

Exact dot products

If $a, b \in \{-2^{20}, \ldots, 0, 1, \ldots, 2^{20}\}$ then ab is a 53-bit fp number so $ab = \text{fp}_{53}(ab)$.

If $a, b, c, d \in \{-2^{20}, \ldots, 2^{20}\}$ then $ab, cd, ab+cd$ are 53-bit fp numbers so $ab = \text{fp}_{53}(ab)$, $cd = \text{fp}_{53}(cd)$, $ab + cd = \text{fp}_{53}(ab+cd)$.

UltraSPARC III computes $a, b, c, d \mapsto ab + cd$ with two fp mults, one fp add.

Bit extraction

Define $\alpha_i = 3 \cdot 2^{i-}$
$\text{top}_i\, r = \text{fp}_{53}(\text{fp}_{53}($
$\text{bottom}_i\, r = \text{fp}_{53}($

If r is a 53-bit fp
and $|r| \le 2^{i+51}$ th
$\text{top}_i\, r \in 2^i\mathbf{Z}$;
$|\text{bottom}_i\, r| \le 2^{i-1}$
$r = \text{top}_i\, r + \text{botto}$

Exact dot products

If $a, b \in \{-2^{20}, \ldots, 0, 1, \ldots, 2^{20}\}$ then ab is a 53-bit fp number so $ab = \text{fp}_{53}(ab)$.

If $a, b, c, d \in \{-2^{20}, \ldots, 2^{20}\}$ then ab, cd, $ab + cd$ are 53-bit fp numbers so $ab = \text{fp}_{53}(ab)$, $cd = \text{fp}_{53}(cd)$, $ab + cd = \text{fp}_{53}(ab + cd)$.

UltraSPARC III computes $a, b, c, d \mapsto ab + cd$ with two fp mults, one fp add.

Bit extraction

Define $\alpha_i = 3 \cdot 2^{i+51}$, $\text{top}_i\, r = \text{fp}_{53}(\text{fp}_{53}(r + \alpha_i) - \alpha_i)$, $\text{bottom}_i\, r = \text{fp}_{53}(r - \text{top}_i\, r)$.

If r is a 53-bit fp number and $|r| \le 2^{i+51}$ then $\text{top}_i\, r \in 2^i\mathbf{Z}$; $|\text{bottom}_i\, r| \le 2^{i-1}$; and $r = \text{top}_i\, r + \text{bottom}_i\, r$.

s

$\ldots, 0, 1, \ldots, 2^{20}\}$
t fp number

$^{20}, \ldots, 2^{20}\}$
d are
so
$= \text{fp}_{53}(cd)$,
$+ cd)$.

omputes
d with
fp add.

Bit extraction

Define $\alpha_i = 3 \cdot 2^{i+51}$,
$\text{top}_i\, r = \text{fp}_{53}(\text{fp}_{53}(r + \alpha_i) - \alpha_i)$,
$\text{bottom}_i\, r = \text{fp}_{53}(r - \text{top}_i\, r)$.

If r is a 53-bit fp number
and $|r| \le 2^{i+51}$ then
$\text{top}_i\, r \in 2^i\mathbf{Z}$;
$|\text{bottom}_i\, r| \le 2^{i-1}$; and
$r = \text{top}_i\, r + \text{bottom}_i\, r$.

Big integers as fp

Every integer mod
can be written as
$u_0 + u_{22} + u_{43} +$
$u_{85} + u_{107} + u_{128}$
$u_{170} + u_{192} + u_{21}$
where $u_i/2^i \in \{-$

Indices i are $\lceil 255j$
for $j \in \{0, 1, \ldots, 1$

Representation is
it's not the input/
Uniqueness would

Bit extraction

Define $\alpha_i = 3 \cdot 2^{i+51}$,
$\text{top}_i\, r = \text{fp}_{53}(\text{fp}_{53}(r + \alpha_i) - \alpha_i)$,
$\text{bottom}_i\, r = \text{fp}_{53}(r - \text{top}_i\, r)$.

If r is a 53-bit fp number
and $|r| \le 2^{i+51}$ then
$\text{top}_i\, r \in 2^i\mathbf{Z}$;
$|\text{bottom}_i\, r| \le 2^{i-1}$; and
$r = \text{top}_i\, r + \text{bottom}_i\, r$.

Big integers as fp sums

Every integer mod $2^{255} - 19$
can be written as a sum
$u_0 + u_{22} + u_{43} + u_{64} +$
$u_{85} + u_{107} + u_{128} + u_{149} +$
$u_{170} + u_{192} + u_{213} + u_{234}$
where $u_i/2^i \in \left\{-2^{22}, \ldots, 2^{22}\right\}$.

Indices i are $\lceil 255j/12 \rceil$
for $j \in \{0, 1, \ldots, 11\}$.

Representation is not unique;
it's not the input/output format.
Uniqueness would cost cycles!

$^{+51}$,

$(r + \alpha_i) - \alpha_i)$,

$r - \mathrm{top}_i\, r)$.

number

hen

^{1}; and

$\mathrm{om}_i\, r$.

Big integers as fp sums

Every integer mod $2^{255} - 19$ can be written as a sum

$u_0 + u_{22} + u_{43} + u_{64} +$
$u_{85} + u_{107} + u_{128} + u_{149} +$
$u_{170} + u_{192} + u_{213} + u_{234}$
where $u_i/2^i \in \{-2^{22}, \ldots, 2^{22}\}$.

Indices i are $\lceil 255j/12 \rceil$ for $j \in \{0, 1, \ldots, 11\}$.

Representation is not unique; it's not the input/output format. Uniqueness would cost cycles!

Assume $u = \sum u_i$

and similarly $v = \sum$

$uv = w_0 + w_{22} +$

where $w_0 = u_0 v_0$,

$w_{22} = u_0 v_{22} + u_2$

$w_{43} = u_0 v_{43} + u_2$

etc.

Each w_i is a 53-bi

Given u_i's and v_i'

can compute w_i's

144 fp mults, 121

Big integers as fp sums

Every integer mod $2^{255} - 19$ can be written as a sum

$u_0 + u_{22} + u_{43} + u_{64} +$
$u_{85} + u_{107} + u_{128} + u_{149} +$
$u_{170} + u_{192} + u_{213} + u_{234}$
where $u_i/2^i \in \{-2^{22}, \ldots, 2^{22}\}$.

Indices i are $\lceil 255j/12 \rceil$ for $j \in \{0, 1, \ldots, 11\}$.

Representation is not unique; it's not the input/output format. Uniqueness would cost cycles!

Assume $u = \sum u_i$ as above, and similarly $v = \sum v_i$. Then
$uv = w_0 + w_{22} + \cdots + w_{468}$
where $w_0 = u_0 v_0$,
$w_{22} = u_0 v_{22} + u_{22} v_0$,
$w_{43} = u_0 v_{43} + u_{22} v_{22} + u_{43} v_0$,
etc.

Each w_i is a 53-bit fp number. Given u_i's and v_i's, can compute w_i's using 144 fp mults, 121 fp adds.

sums

$2^{255} - 19$

a sum

$u_{64} +$

$+ u_{149} +$

$_3 + u_{234}$

$2^{22}, \ldots, 2^{22}\}$.

$j/12\rceil$

$11\}$.

not unique;

output format.

cost cycles!

Assume $u = \sum u_i$ as above, and similarly $v = \sum v_i$. Then

$uv = w_0 + w_{22} + \cdots + w_{468}$

where $w_0 = u_0 v_0$,

$w_{22} = u_0 v_{22} + u_{22} v_0$,

$w_{43} = u_0 v_{43} + u_{22} v_{22} + u_{43} v_0$,

etc.

Each w_i is a 53-bit fp number. Given u_i's and v_i's, can compute w_i's using 144 fp mults, 121 fp adds.

Furthermore, mod

$uv \equiv r_0 + r_{22} + \cdot$

where $r_0 = w_0 +$

$r_{22} = w_{22} + 19 \cdot 2$

Each r_i is a 53-bit

Example: r_0 is an

$|r_0| \leq 381 \cdot 2^{44}$.

Computing r_i's fro

11 fp mults, 11 fp

Structure: $(\mathbf{Z}[t] \cap$

$/(2^{255} t^{12} - 19) \rightarrow$

Assume $u = \sum u_i$ as above, and similarly $v = \sum v_i$. Then $uv = w_0 + w_{22} + \cdots + w_{468}$ where $w_0 = u_0 v_0$, $w_{22} = u_0 v_{22} + u_{22} v_0$, $w_{43} = u_0 v_{43} + u_{22} v_{22} + u_{43} v_0$, etc.

Each w_i is a 53-bit fp number. Given u_i's and v_i's, can compute w_i's using 144 fp mults, 121 fp adds.

Furthermore, modulo $2^{255} - 19$, $uv \equiv r_0 + r_{22} + \cdots + r_{234}$ where $r_0 = w_0 + 19 \cdot 2^{-255} w_{255}$, $r_{22} = w_{22} + 19 \cdot 2^{-255} w_{277}$, etc.

Each r_i is a 53-bit fp number. Example: r_0 is an integer; $|r_0| \le 381 \cdot 2^{44}$.

Computing r_i's from w_i's takes 11 fp mults, 11 fp adds.

Structure: $(\mathbf{Z}[t] \cap \overline{\mathbf{Z}}[2^{255/12} t]) / (2^{255} t^{12} - 19) \to \mathbf{Z}/(2^{255} - 19)$.

as above,

$\sum v_i$. Then

$\cdots + w_{468}$

$_2 v_0$,

$_2 v_{22} + u_{43} v_0$,

t fp number.

s,

using

fp adds.

Furthermore, modulo $2^{255} - 19$,
$uv \equiv r_0 + r_{22} + \cdots + r_{234}$
where $r_0 = w_0 + 19 \cdot 2^{-255} w_{255}$,
$r_{22} = w_{22} + 19 \cdot 2^{-255} w_{277}$, etc.

Each r_i is a 53-bit fp number.
Example: r_0 is an integer;
$|r_0| \leq 381 \cdot 2^{44}$.

Computing r_i's from w_i's takes
11 fp mults, 11 fp adds.

Structure: $(\mathbf{Z}[t] \cap \overline{\mathbf{Z}}[2^{255/12} t])$
$/(2^{255} t^{12} - 19) \to \mathbf{Z}/(2^{255} - 19)$.

Carries

"Carry from r_0 to

replace r_0 and r_{22}

bottom$_{22}\, r_0$ and r

This takes 4 fp ad

and guarantees $|r_0$

Series of 13 carries

in range for subsec

from r_{192} to r_{213}

then from r_0 to r_2

to r_{192} to r_{213}.

This takes 52 fp a

Furthermore, modulo $2^{255} - 19$,
$uv \equiv r_0 + r_{22} + \cdots + r_{234}$
where $r_0 = w_0 + 19 \cdot 2^{-255} w_{255}$,
$r_{22} = w_{22} + 19 \cdot 2^{-255} w_{277}$, etc.

Each r_i is a 53-bit fp number.
Example: r_0 is an integer;
$|r_0| \le 381 \cdot 2^{44}$.

Computing r_i's from w_i's takes
11 fp mults, 11 fp adds.

Structure: $(\mathbf{Z}[t] \cap \overline{\mathbf{Z}}[2^{255/12}t])$
$/(2^{255}t^{12} - 19) \to \mathbf{Z}/(2^{255} - 19)$.

Carries

"Carry from r_0 to r_{22}":
replace r_0 and r_{22} by
$\text{bottom}_{22}\, r_0$ and $r_{22} + \text{top}_{22}\, r_0$.
This takes 4 fp adds,
and guarantees $|r_0| \le 2^{21}$.

Series of 13 carries puts all r_i's
in range for subsequent products:
from r_{192} to r_{213} to r_{234} to w_{255};
then from r_0 to r_{22} to r_{43} to ...
to r_{192} to r_{213}.
This takes 52 fp adds.

ulo $2^{255} - 19$,

$\cdots + r_{234}$

$19 \cdot 2^{-255} w_{255}$,

$2^{-255} w_{277}$, etc.

t fp number.

integer;

om w_i's takes

adds.

$\overline{\mathbf{Z}}[2^{255/12} t])$

$\to \mathbf{Z}/(2^{255} - 19)$.

Carries

"Carry from r_0 to r_{22}":
replace r_0 and r_{22} by
$\text{bottom}_{22}\, r_0$ and $r_{22} + \text{top}_{22}\, r_0$.
This takes 4 fp adds,
and guarantees $|r_0| \le 2^{21}$.

Series of 13 carries puts all r_i's
in range for subsequent products:
from r_{192} to r_{213} to r_{234} to w_{255};
then from r_0 to r_{22} to r_{43} to ...
to r_{192} to r_{213}.
This takes 52 fp adds.

Total 155 mults, 1
to multiply modul
in this representati

$\ge$ 184 UltraSPAR

$=$ 184 cycles? Tw
fp-operation latenc
"load/store" laten
limited number of

Schedule instructi
to bring cycles do

Carries

"Carry from r_0 to r_{22}": replace r_0 and r_{22} by $\text{bottom}_{22}\, r_0$ and $r_{22} + \text{top}_{22}\, r_0$. This takes 4 fp adds, and guarantees $|r_0| \le 2^{21}$.

Series of 13 carries puts all r_i's in range for subsequent products: from r_{192} to r_{213} to r_{234} to w_{255}; then from r_0 to r_{22} to r_{43} to $\ldots$ to r_{192} to r_{213}. This takes 52 fp adds.

Total 155 mults, 184 adds to multiply modulo $2^{255} - 19$ in this representation.

$\ge$ 184 UltraSPARC III cycles.

$=$ 184 cycles? Two obstacles: fp-operation latency; "load/store" latency imposed by limited number of "registers."

Schedule instructions carefully to bring cycles down to $\approx$ 184.

r_{22}":
by

$_{22}$ + top$_{22}$ r_0.
ds,
$| \leq 2^{21}$.

s puts all r_i's
quent products:
to r_{234} to w_{255};
$_{22}$ to r_{43} to ...

dds.

Total 155 mults, 184 adds
to multiply modulo $2^{255} - 19$
in this representation.

$\geq$ 184 UltraSPARC III cycles.

= 184 cycles? Two obstacles:
fp-operation latency;
"load/store" latency imposed by
limited number of "registers."

Schedule instructions carefully
to bring cycles down to $\approx$ 184.

Have developed q
new programming
for high-speed com

Includes range ver
guided register all

Lets me write des
with much less hu
traditional asm, C
Have also used fo
fast Poly1305, fas
see, e.g., `http://`
`/mac/poly1305_a`

Total 155 mults, 184 adds to multiply modulo $2^{255} - 19$ in this representation.

≥ 184 UltraSPARC III cycles.

$= 184$ cycles? Two obstacles: fp-operation latency; "load/store" latency imposed by limited number of "registers."

Schedule instructions carefully to bring cycles down to ≈ 184.

Have developed `qhasm`, new programming language for high-speed computations.

Includes range verification, guided register allocation, et al.

Lets me write desired code with much less human time than traditional asm, C compiler, etc. Have also used for fast AES, fast Poly1305, fast Salsa20, etc.; see, e.g., `http://cr.yp.to/mac/poly1305_athlon.s`.

84 adds
o $2^{255} - 19$
ion.

C III cycles.

o obstacles:
cy;
cy imposed by
"registers."

ons carefully
wn to ≈ 184.

Have developed qhasm,
new programming language
for high-speed computations.

Includes range verification,
guided register allocation, et al.

Lets me write desired code
with much less human time than
traditional asm, C compiler, etc.
Have also used for fast AES,
fast Poly1305, fast Salsa20, etc.;
see, e.g., `http://cr.yp.to/mac/poly1305_athlon.s`.

Speedup: Squarin

Often know in adv

$u_0 u_{64} + u_{22} u_{43} +$
is more efficiently
$2(u_0 u_{64} + u_{22} u_{43}$

Even better: First
$2u_0, 2u_{22}, \ldots, 2u_2$
and then compute
$(2u_0)u_{64} + (2u_{22}$

130 fp adds instea
Makes carry time

Have developed `qhasm`,
new programming language
for high-speed computations.

Includes range verification,
guided register allocation, et al.

Lets me write desired code
with much less human time than
traditional asm, C compiler, etc.
Have also used for fast AES,
fast Poly1305, fast Salsa20, etc.;
see, e.g., `http://cr.yp.to/mac/poly1305_athlon.s`.

Speedup: Squarings

Often know in advance that $u = v$.

$u_0 u_{64} + u_{22} u_{43} + u_{43} u_{22} + u_{64} u_0$
is more efficiently computed as
$2(u_0 u_{64} + u_{22} u_{43})$.

Even better: First compute
$2u_0, 2u_{22}, \ldots, 2u_{234}$
and then compute
$(2u_0)u_{64} + (2u_{22})u_{43}$ etc.

130 fp adds instead of 184.
Makes carry time even more visible.

nasm,
language
nputations.

ification,
ocation, et al.

red code
man time than
compiler, etc.
fast AES,
t Salsa20, etc.;
cr.yp.to
athlon.s.

Speedup: Squarings

Often know in advance that $u = v$.

$u_0 u_{64} + u_{22} u_{43} + u_{43} u_{22} + u_{64} u_0$
is more efficiently computed as
$2(u_0 u_{64} + u_{22} u_{43})$.

Even better: First compute
$2u_0, 2u_{22}, \ldots, 2u_{234}$
and then compute
$(2u_0)u_{64} + (2u_{22})u_{43}$ etc.

130 fp adds instead of 184.
Makes carry time even more visible.

Speedup: Karatsu

Say $A_0 = u_0 + u_2$
$A_1 = u_{128} + u_{149}t$
$B_0 = v_0 + \cdots$, B_1

Original, 184 adds
$A_0 B_0 + (A_0 B_1 + A$

Karatsuba, 182 ad
$((A_0 + A_1)(B_0 + B_1$
$+ A_0 B_0 + A_1 B_1 t^1$

Improved Karatsub
$(A_0 + A_1)(B_0 + B$
$+ (A_0 B_0 - A_1 B_1 t$

Speedup: Squarings

Often know in advance that $u = v$.

$u_0 u_{64} + u_{22} u_{43} + u_{43} u_{22} + u_{64} u_0$
is more efficiently computed as
$2(u_0 u_{64} + u_{22} u_{43})$.

Even better: First compute
$2u_0, 2u_{22}, \ldots, 2u_{234}$
and then compute
$(2u_0)u_{64} + (2u_{22})u_{43}$ etc.

130 fp adds instead of 184.
Makes carry time even more visible.

Speedup: Karatsuba's method

Say $A_0 = u_0 + u_{22}t + \cdots + u_{107}t^5$,
$A_1 = u_{128} + u_{149}t + \cdots + u_{234}t^5$,
$B_0 = v_0 + \cdots$, $B_1 = v_{128} + \cdots$.

Original, 184 adds: Product is
$A_0 B_0 + (A_0 B_1 + A_1 B_0)t^6 + A_1 B_1 t^{12}$.

Karatsuba, 182 adds:
$((A_0+A_1)(B_0+B_1) - A_0 B_0 - A_1 B_1)t^6$
$+ A_0 B_0 + A_1 B_1 t^{12}$.

Improved Karatsuba, 177 adds:
$(A_0 + A_1)(B_0 + B_1)t^6$
$+ (A_0 B_0 - A_1 B_1 t^6)(1 - t^6)$.

gs

vance that $u = v$.

$u_{43}u_{22} + u_{64}u_0$

computed as

).

compute

234

)u_{43} etc.

d of 184.

even more visible.

Speedup: Karatsuba's method

Say $A_0 = u_0 + u_{22}t + \cdots + u_{107}t^5$, $A_1 = u_{128} + u_{149}t + \cdots + u_{234}t^5$, $B_0 = v_0 + \cdots$, $B_1 = v_{128} + \cdots$.

Original, 184 adds: Product is $A_0B_0 + (A_0B_1 + A_1B_0)t^6 + A_1B_1t^{12}$.

Karatsuba, 182 adds:
$((A_0+A_1)(B_0+B_1) - A_0B_0 - A_1B_1)t^6 + A_0B_0 + A_1B_1t^{12}$.

Improved Karatsuba, 177 adds:
$(A_0 + A_1)(B_0 + B_1)t^6 + (A_0B_0 - A_1B_1t^6)(1 - t^6)$.

The Curve functio

Overall strategy to

U, Curve(V) $\mapsto$ Cu

using arithmetic m

For various integer

find x_n, z_n such th

Curve(nV) $\equiv x_n/$

i.e., z_n Curve(nV)

e.g. $x_1 =$ Curve(V

assuming Curve(V

Can easily restrict

to ensure that ∞

Speedup: Karatsuba's method

Say $A_0 = u_0 + u_{22}t + \cdots + u_{107}t^5$,
$A_1 = u_{128} + u_{149}t + \cdots + u_{234}t^5$,
$B_0 = v_0 + \cdots$, $B_1 = v_{128} + \cdots$.

Original, 184 adds: Product is
$A_0 B_0 + (A_0 B_1 + A_1 B_0)t^6 + A_1 B_1 t^{12}$.

Karatsuba, 182 adds:
$((A_0+A_1)(B_0+B_1)-A_0 B_0-A_1 B_1)t^6$
$+ A_0 B_0 + A_1 B_1 t^{12}$.

Improved Karatsuba, 177 adds:
$(A_0 + A_1)(B_0 + B_1)t^6$
$+ (A_0 B_0 - A_1 B_1 t^6)(1 - t^6)$.

The Curve function

Overall strategy to compute
U, Curve(V) $\mapsto$ Curve(UV),
using arithmetic mod $p = 2^{255} - 19$:

For various integers n,
find x_n, z_n such that
$\text{Curve}(nV) \equiv x_n/z_n \pmod{p}$,
i.e., $z_n \, \text{Curve}(nV) \equiv x_n \pmod{p}$.

e.g. $x_1 = \text{Curve}(V)$, $z_1 = 1$,
assuming Curve(V) $\neq \infty$.

Can easily restrict U, Curve(V)
to ensure that ∞ never appears.

ba's method

$_2t + \cdots + u_{107}t^5$,

$t + \cdots + u_{234}t^5$,

$= v_{128} + \cdots$.

: Product is

$_1B_0)t^6 + A_1B_1t^{12}$.

ds:

$) - A_0B_0 - A_1B_1)t^6$

^{2}.

ba, 177 adds:

$B_1)t^6$

$^6)(1 - t^6)$.

The Curve function

Overall strategy to compute
$U, \text{Curve}(V) \mapsto \text{Curve}(UV)$,
using arithmetic mod $p = 2^{255} - 19$:

For various integers n,
find x_n, z_n such that
$\text{Curve}(nV) \equiv x_n/z_n \pmod{p}$,
i.e., $z_n \text{Curve}(nV) \equiv x_n \pmod{p}$.

e.g. $x_1 = \text{Curve}(V)$, $z_1 = 1$,
assuming $\text{Curve}(V) \neq \infty$.

Can easily restrict $U, \text{Curve}(V)$
to ensure that ∞ never appears.

We'll see how to c

$x_m, z_m \mapsto x_{2m}, z_2$

x_m, z_m, x_{m+1}, z_m

$\mapsto x_{2m+1}, z_{2m+1}$.

Combine to compu

x_m, z_m, x_{m+1}, z_m

$\mapsto x_n, z_n, x_{n+1}, z$

where $m = \lfloor n/2 \rfloor$

Conditional branch
input-dependent lo
can leak b via timi
Replace with arith
e.g., $(1 - b)x_m +$

The Curve function

Overall strategy to compute
$U, \text{Curve}(V) \mapsto \text{Curve}(UV)$,
using arithmetic mod $p = 2^{255} - 19$:

For various integers n,
find x_n, z_n such that
$\text{Curve}(nV) \equiv x_n/z_n \pmod{p}$,
i.e., $z_n \text{Curve}(nV) \equiv x_n \pmod{p}$.

e.g. $x_1 = \text{Curve}(V)$, $z_1 = 1$,
assuming $\text{Curve}(V) \neq \infty$.

Can easily restrict $U, \text{Curve}(V)$
to ensure that ∞ never appears.

We'll see how to compute
$x_m, z_m \mapsto x_{2m}, z_{2m}$; and
$x_m, z_m, x_{m+1}, z_{m+1}, \text{Curve}(V)$
$\mapsto x_{2m+1}, z_{2m+1}$.

Combine to compute
$x_m, z_m, x_{m+1}, z_{m+1}, b, \text{Curve}(V)$
$\mapsto \; x_n, z_n, x_{n+1}, z_{n+1}$
where $m = \lfloor n/2 \rfloor$, $b = n \bmod 2$.

Conditional branches and
input-dependent load addresses
can leak b via timing.
Replace with arithmetic:
e.g., $(1 - b)x_m + (b)x_{m+1}$.

n

o compute
rve(UV),
nod $p = 2^{255} - 19$:

rs n,
hat
$z_n \pmod p$),
$\equiv x_n \pmod p$).

), $z_1 = 1$,
) $\neq \infty$.

U, Curve(V)
never appears.

We'll see how to compute
$x_m, z_m \mapsto x_{2m}, z_{2m}$; and
$x_m, z_m, x_{m+1}, z_{m+1}$, Curve($V$)
$\mapsto x_{2m+1}, z_{2m+1}$.

Combine to compute
$x_m, z_m, x_{m+1}, z_{m+1}, b$, Curve($V$)
$\mapsto x_n, z_n, x_{n+1}, z_{n+1}$
where $m = \lfloor n/2 \rfloor$, $b = n \bmod 2$.

Conditional branches and input-dependent load addresses can leak b via timing.
Replace with arithmetic:
e.g., $(1-b)x_m + (b)x_{m+1}$.

Eventually reach n

Divide x_U by z_U n
to obtain Curve(U

Simple division me
$x_U/z_U \equiv x_U z_U^{p-2}$.
Euclid-type divisio
are faster but have
input-dependent ti

Finally convert fro
floating-point repr
to byte-string outp

We'll see how to compute

$x_m, z_m \mapsto x_{2m}, z_{2m}$; and

$x_m, z_m, x_{m+1}, z_{m+1}$, Curve($V$)

$\mapsto x_{2m+1}, z_{2m+1}$.

Combine to compute

$x_m, z_m, x_{m+1}, z_{m+1}, b$, Curve($V$)

$\mapsto \; x_n, z_n, x_{n+1}, z_{n+1}$

where $m = \lfloor n/2 \rfloor$, $b = n$ mod 2.

Conditional branches and
input-dependent load addresses
can leak b via timing.
Replace with arithmetic:
e.g., $(1-b)x_m + (b)x_{m+1}$.

Eventually reach $n = U$.

Divide x_U by z_U modulo p
to obtain Curve(UV).

Simple division method: Fermat!
$x_U/z_U \equiv x_U z_U^{p-2}$.
Euclid-type division methods
are faster but have
input-dependent timings.

Finally convert from
floating-point representation
to byte-string output format.

compute

$_{2m}$; and

$_{+1}$, Curve(V)

ute

$_{+1}$, b, Curve(V)

$_{n+1}$

, $b = n$ mod 2.

nes and

oad addresses

ng.

metic:

$(b)x_{m+1}$.

Eventually reach $n = U$.

Divide x_U by z_U modulo p to obtain Curve(UV).

Simple division method: Fermat! $x_U/z_U \equiv x_U z_U^{p-2}$.
Euclid-type division methods are faster but have input-dependent timings.

Finally convert from floating-point representation to byte-string output format.

<u>From n to $2n$</u>

In $\mathbf{Z}/p$:
$x_{2n} = (x_n^2 - z_n^2)^2$
$z_{2n} = 4x_n z_n(x_n^2 -$

Compute as follow
$(x_n - z_n)^2$; $(x_n +$
$x_{2n} = (x_n - z_n)^2$
$4x_n z_n = (x_n + z_n$
$(A-2)x_n z_n = 89$
$z_{2n} =$
$4x_n z_n((x_n + z_n)^2$

Eventually reach $n = U$.

Divide x_U by z_U modulo p to obtain Curve(UV).

Simple division method: Fermat! $x_U/z_U \equiv x_U z_U^{p-2}$.
Euclid-type division methods are faster but have input-dependent timings.

Finally convert from floating-point representation to byte-string output format.

From n to $2n$

In $\mathbf{Z}/p$:
$x_{2n} = (x_n^2 - z_n^2)^2$,
$z_{2n} = 4x_n z_n(x_n^2 + Ax_n z_n + z_n^2)$.

Compute as follows:
$(x_n - z_n)^2$; $(x_n + z_n)^2$;
$x_{2n} = (x_n - z_n)^2(x_n + z_n)^2$;
$4x_n z_n = (x_n + z_n)^2 - (x_n - z_n)^2$;
$(A-2)x_n z_n = 89747 \cdot 4x_n z_n$;
$z_{2n} =$
$4x_n z_n((x_n + z_n)^2 + (A-2)x_n z_n)$.

$n = U$.

modulo p

V).

ethod: Fermat!

n methods

e

imings.

m

esentation

out format.

From n to $2n$

In $\mathbf{Z}/p$:

$x_{2n} = (x_n^2 - z_n^2)^2$,

$z_{2n} = 4x_n z_n(x_n^2 + Ax_n z_n + z_n^2)$.

Compute as follows:

$(x_n - z_n)^2$; $(x_n + z_n)^2$;

$x_{2n} = (x_n - z_n)^2(x_n + z_n)^2$;

$4x_n z_n = (x_n + z_n)^2 - (x_n - z_n)^2$;

$(A-2)x_n z_n = 89747 \cdot 4x_n z_n$;

$z_{2n} =$

$4x_n z_n((x_n + z_n)^2 + (A-2)x_n z_n)$.

From $n, n+1$ to 2

$x_{2n+1} = 4(x_n x_{n+}$

$z_{2n+1} =$

$4(x_n z_{n+1} - z_n x_{n}$

Compute as follow

$(x_n - z_n)(x_{n+1} +$

$(x_n + z_n)(x_{n+1} -$

$2(x_n x_{n+1} - z_n z_n$

$2(x_n z_{n+1} - z_n x_n$

$x_{2n+1} = (2(x_n x_n$

$(2(x_n z_{n+1} - z_n x_n$

$z_{2n+1} = (\cdots)$ Curv

From n to $2n$

In $\mathbf{Z}/p$:

$x_{2n} = (x_n^2 - z_n^2)^2$,

$z_{2n} = 4x_n z_n(x_n^2 + Ax_n z_n + z_n^2)$.

Compute as follows:

$(x_n - z_n)^2$; $(x_n + z_n)^2$;

$x_{2n} = (x_n - z_n)^2(x_n + z_n)^2$;

$4x_n z_n = (x_n + z_n)^2 - (x_n - z_n)^2$;

$(A - 2)x_n z_n = 89747 \cdot 4x_n z_n$;

$z_{2n} =$

$4x_n z_n((x_n + z_n)^2 + (A - 2)x_n z_n)$.

From $n, n+1$ to $2n+1$

$x_{2n+1} = 4(x_n x_{n+1} - z_n z_{n+1})^2$,

$z_{2n+1} =$

$4(x_n z_{n+1} - z_n x_{n+1})^2$ Curve(V).

Compute as follows:

$(x_n - z_n)(x_{n+1} + z_{n+1})$;

$(x_n + z_n)(x_{n+1} - z_{n+1})$;

$2(x_n x_{n+1} - z_n z_{n+1}) =$ sum;

$2(x_n z_{n+1} - z_n x_{n+1}) =$ difference;

$x_{2n+1} = (2(x_n x_{n+1} - z_n z_{n+1}))^2$;

$(2(x_n z_{n+1} - z_n x_{n+1}))^2$;

$z_{2n+1} = (\cdots)$ Curve(V).

,

$+ Ax_n z_n + z_n^2)$.

vs:

$- z_n)^2$;

$(x_n + z_n)^2$;

$)^2 - (x_n - z_n)^2$;

$747 \cdot 4x_n z_n$;

$^2 + (A - 2)x_n z_n)$.

From $n, n+1$ to $2n+1$

$x_{2n+1} = 4(x_n x_{n+1} - z_n z_{n+1})^2$,

$z_{2n+1} =$

$4(x_n z_{n+1} - z_n x_{n+1})^2 \operatorname{Curve}(V)$.

Compute as follows:

$(x_n - z_n)(x_{n+1} + z_{n+1})$;

$(x_n + z_n)(x_{n+1} - z_{n+1})$;

$2(x_n x_{n+1} - z_n z_{n+1}) =$ sum;

$2(x_n z_{n+1} - z_n x_{n+1}) =$ difference;

$x_{2n+1} = (2(x_n x_{n+1} - z_n z_{n+1}))^2$;

$(2(x_n z_{n+1} - z_n x_{n+1}))^2$;

$z_{2n+1} = (\cdots) \operatorname{Curve}(V)$.

Total time

Slightly over 1600

(520 from carries)

for each bit of U.

Total for 256-bit U

≈ 413000 fp adds;

≈ 50000 fp adds f

Aiming for 500000

Still have to finish

Should end up eve

my NIST P-224 sc

despite 14% more

From $n, n+1$ to $2n+1$

$x_{2n+1} = 4(x_n x_{n+1} - z_n z_{n+1})^2$,
$z_{2n+1} =$
$4(x_n z_{n+1} - z_n x_{n+1})^2 \operatorname{Curve}(V)$.

Compute as follows:
$(x_n - z_n)(x_{n+1} + z_{n+1})$;
$(x_n + z_n)(x_{n+1} - z_{n+1})$;
$2(x_n x_{n+1} - z_n z_{n+1}) = \text{sum}$;
$2(x_n z_{n+1} - z_n x_{n+1}) = \text{difference}$;
$x_{2n+1} = (2(x_n x_{n+1} - z_n z_{n+1}))^2$;
$(2(x_n z_{n+1} - z_n x_{n+1}))^2$;
$z_{2n+1} = (\cdots) \operatorname{Curve}(V)$.

Total time

Slightly over 1600 fp adds
(520 from carries)
for each bit of U.

Total for 256-bit U:
$\approx$ 413000 fp adds; plus
$\approx$ 50000 fp adds for final division.

Aiming for 500000 cycles.
Still have to finish software.
Should end up even faster than
my NIST P-224 software,
despite 14% more bits!