ECC optimization on Sandy Bridge

The cost of cofactor $h = 1$

Daan Sprenkels
hello@dsprenkels.com

Radboud University Nijmegen

1 April 2019

Outline

Outline

Introduction

- Preliminaries
- Cofactor security

ECC implementation

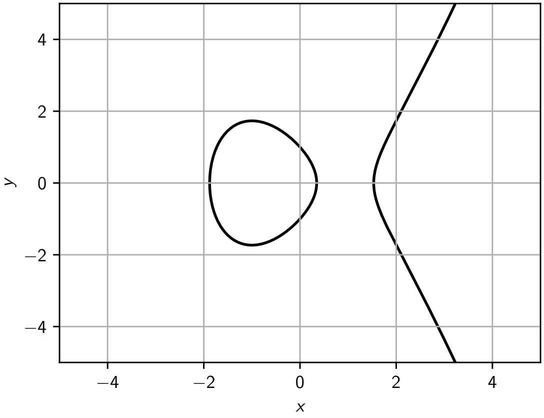
Results

Elliptic curves

$$\mathcal{E} : y^2 = x^3 + ax + b$$

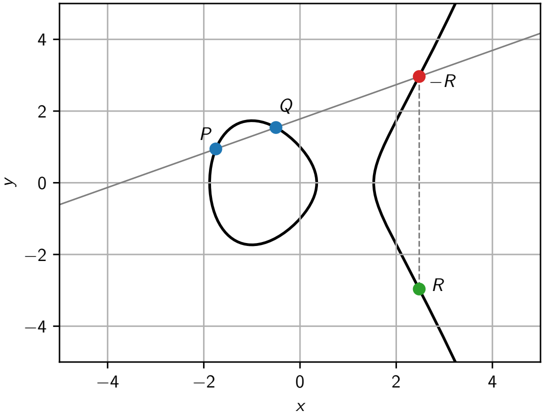
Elliptic curves

$$\mathcal{E} : y^2 = x^3 + ax + b$$



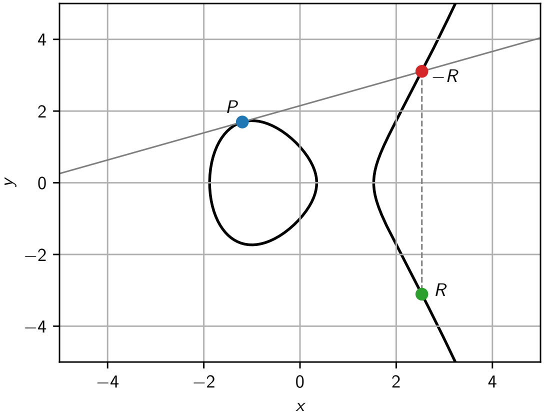
Elliptic curves: addition

$$\mathcal{E} : y^2 = x^3 + ax + b$$



Elliptic curves: doubling

$$\mathcal{E} : y^2 = x^3 + ax + b$$



Elliptic curves

- Coordinates include *the point at infinity* $\mathcal{O}$
  - Define $P + \mathcal{O} = P$

Elliptic curves

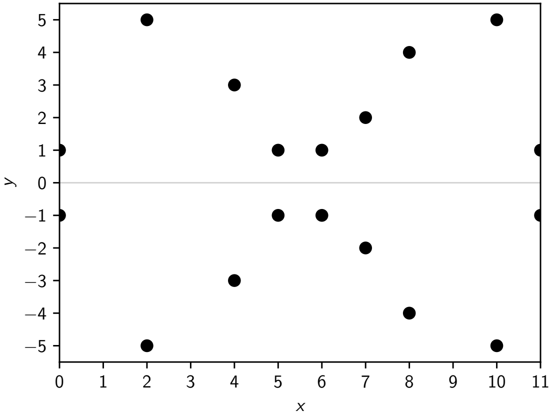
- Coordinates include *the point at infinity* $\mathcal{O}$
  - Define $P + \mathcal{O} = P$
- Curve equation: $\mathcal{E} : y^2 = x^3 + ax + b$

Elliptic curves

- Coordinates include *the point at infinity* $\mathcal{O}$
  - Define $P + \mathcal{O} = P$
- Curve equation: $\mathcal{E} : y^2 = x^3 + ax + b$
- Coordinates are defined over a field $\mathbb{F}_q$
  - I.e. integers modulo q

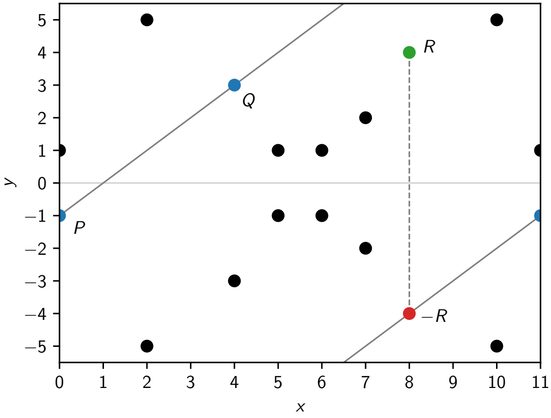
Elliptic curves: actually

$$\mathcal{E} : y^2 = x^3 - 3x + 1 \text{ defined over } \mathbb{F}_{11}$$



Elliptic curves: actual addition

$$\mathcal{E} : y^2 = x^3 - 3x + 1 \text{ defined over } \mathbb{F}_{11}$$



Group arithmetic

- We can do arithmetic with these rules! :)

- Addition: $P + Q$
- Subtraction: $P - Q$
- Neutral element: $\mathcal{O}$, i.e. "zero"

Group arithmetic

- We can do arithmetic with these rules! :)
- Addition: $P + Q$
- Subtraction: $P - Q$
- Neutral element: $\mathcal{O}$, i.e. “zero”
- Scalar multiplication: $[k]P = \underbrace{P + P + ... + P}_{k \text{ times}}$

Group arithmetic

- We can do arithmetic with these rules! :)
- Addition: $P + Q$
- Subtraction: $P - Q$
- Neutral element: $\mathcal{O}$, i.e. "zero"
- Scalar multiplication: $[k]P = \underbrace{P + P + ... + P}_{k \text{ times}}$
- Discrete log problem:
  given P, Q where $[k]P = Q$, hard to find k

Elliptic curves are cyclic

- Points form a cycle: $\mathcal{O} \xrightarrow{+P} P \xrightarrow{+P} [2]P \xrightarrow{+P} [3]P \xrightarrow{+P} \ldots \xrightarrow{+P} [n-1]P \xrightarrow{+P} \mathcal{O}$

Elliptic curves are cyclic

- Points form a cycle: $\underbrace{\mathcal{O} \xrightarrow{+P} P \xrightarrow{+P} [2]P \xrightarrow{+P} [3]P \xrightarrow{+P} \ldots \xrightarrow{+P} [n-1]P \xrightarrow{+P} \mathcal{O}}_{n \text{ steps}}$
- The **order** n should contain a large prime factor
- Only *one* cycle if n is prime

Cofactors

- If n is **not** a prime
  Then $n = h \cdot \ell$
- I.e. small loops are possible:
  E.g. if $4|n$, then there is a point T_4: $\underbrace{\mathcal{O} \xrightarrow{+T_4} T_4 \xrightarrow{+T_4} [2]\,T_4 \xrightarrow{+T_4} [3]\,T_4 \xrightarrow{+T_4} \mathcal{O}}_{\text{only 4 steps!}}$

Cofactors

- If n is **not** a prime
  Then $n = h \cdot \ell$
- I.e. small loops are possible:
  E.g. if $4|n$, then there is a point T_4: $\underbrace{\mathcal{O} \xrightarrow{+T_4} T_4 \xrightarrow{+T_4} [2]\,T_4 \xrightarrow{+T_4} [3]\,T_4 \xrightarrow{+T_4} \mathcal{O}}_{\text{only 4 steps!}}$
- h is called the **cofactor**

Cofactors

- If n is **not** a prime
  Then $n = h \cdot \ell$
- I.e. small loops are possible:
  E.g. if $4|n$, then there is a point T_4: $\underbrace{\mathcal{O} \xrightarrow{+T_4} T_4 \xrightarrow{+T_4} [2]\,T_4 \xrightarrow{+T_4} [3]\,T_4 \xrightarrow{+T_4} \mathcal{O}}_{\text{only 4 steps!}}$
- h is called the **cofactor**
- This property is often harmless

Cofactors

- If n is **not** a prime
  Then $n = h \cdot \ell$
- I.e. small loops are possible:
  E.g. if $4|n$, then there is a point T_4: $\underbrace{\mathcal{O} \xrightarrow{+T_4} T_4 \xrightarrow{+T_4} [2]\,T_4 \xrightarrow{+T_4} [3]\,T_4 \xrightarrow{+T_4} \mathcal{O}}_{\text{only 4 steps!}}$
- h is called the **cofactor**
- This property is often harmless
  - I.e. sometimes it's the opposite of harmless

A brief history...

- 1999: elliptic curves popularized

A brief history...

- 1999: elliptic curves popularized
- 2006: Curve25519 published by Bernstein
  - “Safe” for implementors

A brief history...

- 1999: elliptic curves popularized
- 2006: Curve25519 published by Bernstein
  - “Safe” for implementors
  - Super fast

A brief history...

- 1999: elliptic curves popularized
- 2006: Curve25519 published by Bernstein
  - “Safe” for implementors
  - Super fast
  - Has cofactor $h = 8$

A brief history...

- 1999: elliptic curves popularized
- 2006: Curve25519 published by Bernstein
  - “Safe” for implementors
  - Super fast
  - Has cofactor $h = 8$
- 2014: Monero cryptocurrency
  - Uses Curve25519

A brief history...

- 1999: elliptic curves popularized
- 2006: Curve25519 published by Bernstein
  - "Safe" for implementors
  - Super fast
  - Has cofactor $h = 8$
- 2014: Monero cryptocurrency
  - Uses Curve25519
- 2017: vulnerability in Monero found
  - Allowed anyone to create coins out of thin air

The Monero vulnerability

- Transaction involves a *ring signature*

The Monero vulnerability

- Transaction involves a *ring signature*
- Double-spending is prevented by a *key image* I

The Monero vulnerability

- Transaction involves a *ring signature*
- Double-spending is prevented by a *key image* I
  - I binds the transaction to signer's public key P

The Monero vulnerability

- Transaction involves a *ring signature*
- Double-spending is prevented by a *key image* I
  - I binds the transaction to signer's public key P
  - Binding is in zero-knowledge

The Monero vulnerability

- Transaction involves a *ring signature*
- Double-spending is prevented by a *key image* I
  - I binds the transaction to signer's public key P
  - Binding is in zero-knowledge
  - Key image I should be unique

Monero transactions

- Have generators G_1, G_2; private key x; public key P; key image I.
- $\textsc{Sign}_x(m)$
  - Sign m with private key x
  - Choose commitment $u \in_R h\mathbb{Z}_\ell$
  - Compute $a_2 = [u]G_2$; $c = H(m, a_1, a_2)$; $r = u + cx$
  - Output signature $s = (a_1, a_2, r)$

Monero transactions

- Have generators G_1, G_2; private key x; public key P; key image I.
- $\textsc{Sign}_x(m)$
  - Sign m with private key x
  - Choose commitment $u \in_R h\mathbb{Z}_\ell$
  - Compute $a_2 = [u]G_2$; $c = H(m, a_1, a_2)$; $r = u + cx$
  - Output signature $s = (a_1, a_2, r)$
- $\textsc{Verify}_{P,I}(m, s)$
  - $[r]G_1 \stackrel{?}{=} a_1 + [c]P$
  - $[r]G_2 \stackrel{?}{=} a_2 + [c]I$
  - I unique?

Attacking Monero signatures

- **Challenge.** Find some signature+keypair a_2, c, r, and I, s.t.

$$[r]G_2 = a_2 + [c]I = a_2 + [c]I',$$

where $I \neq I'$.

Attacking Monero signatures

- **Challenge.** Find some signature+keypair a_2, c, r, and I, s.t.

$$[r]G_2 = a_2 + [c]I = a_2 + [c]I',$$

where $I \neq I'$.

- **Solution.** Choose $I' = I + T_\alpha$, where $\alpha | c$ and $[\alpha]T_\alpha = \mathcal{O}$.

Attacking Monero signatures

- **Challenge.** Find some signature+keypair a_2, c, r, and I, s.t.

$$[r]G_2 = a_2 + [c]I = a_2 + [c]I',$$

where $I \neq I'$.

- **Solution.** Choose $I' = I + T_\alpha$, where $\alpha | c$ and $[\alpha]T_\alpha = \mathcal{O}$.
- **Correctness.**

$$a_2 + [c]I' = a_2 + [c](I + T_\alpha)$$

Attacking Monero signatures

- **Challenge.** Find some signature+keypair a_2, c, r, and I, s.t.

$$[r]G_2 = a_2 + [c]I = a_2 + [c]I',$$

where $I \neq I'$.

- **Solution.** Choose $I' = I + T_\alpha$, where $\alpha | c$ and $[\alpha]T_\alpha = \mathcal{O}$.
- **Correctness.**

$$\begin{aligned} a_2 + [c]I' &= a_2 + [c](I + T_\alpha) \\ &= a_2 + [c]I + \left[\frac{c}{\alpha}\right][\alpha]T_\alpha \end{aligned}$$

Attacking Monero signatures

- **Challenge.** Find some signature+keypair a_2, c, r, and I, s.t.

$$[r]G_2 = a_2 + [c]I = a_2 + [c]I',$$

where $I \neq I'$.

- **Solution.** Choose $I' = I + T_\alpha$, where $\alpha | c$ and $[\alpha]T_\alpha = \mathcal{O}$.
- **Correctness.**

$$\begin{aligned} a_2 + [c]I' &= a_2 + [c](I + T_\alpha) \\ &= a_2 + [c]I + \left[\frac{c}{\alpha}\right][\alpha]T_\alpha \\ &= a_2 + [c]I + \left[\frac{c}{\alpha}\right]\mathcal{O} \end{aligned}$$

Attacking Monero signatures

- **Challenge.** Find some signature+keypair a_2, c, r, and I, s.t.

$$[r]G_2 = a_2 + [c]I = a_2 + [c]I',$$

where $I \neq I'$.

- **Solution.** Choose $I' = I + T_\alpha$, where $\alpha | c$ and $[\alpha]T_\alpha = \mathcal{O}$.
- **Correctness.**

$$\begin{aligned} a_2 + [c]I' &= a_2 + [c](I + T_\alpha) \\ &= a_2 + [c]I + \left[\frac{c}{\alpha}\right][\alpha]T_\alpha \\ &= a_2 + [c]I + \cancel{\left[\frac{c}{\alpha}\right]\mathcal{O}} \end{aligned}$$

Attacking Monero signatures

- **Challenge.** Find some signature+keypair a_2, c, r, and I, s.t.

$$[r]G_2 = a_2 + [c]I = a_2 + [c]I',$$

where $I \neq I'$.

- **Solution.** Choose $I' = I + T_\alpha$, where $\alpha | c$ and $[\alpha]T_\alpha = \mathcal{O}$.
- **Correctness.**

$$\begin{aligned} a_2 + [c]I' &= a_2 + [c](I + T_\alpha) \\ &= a_2 + [c]I + \left[\frac{c}{\alpha}\right][\alpha]T_\alpha \\ &= a_2 + [c]I + \cancel{\left[\frac{c}{\alpha}\right]\mathcal{O}} \\ &= a_2 + [c]I \end{aligned}$$

Surely this could have been prevented?

Easy fix:

- Protocol assumed $[r]G_2 = a_2 + [c]I$, only for a **single** I

Surely this could have been prevented?

Easy fix:

- Protocol assumed $[r]G_2 = a_2 + [c]I$, only for a **single** I
- Fix: check if the order of I is ℓ

Surely this could have been prevented?

Easy fix:

- Protocol assumed $[r]G_2 = a_2 + [c]I$, only for a **single** I
- Fix: check if the order of I is ℓ
  - i.e. check $[\ell]I \stackrel{?}{=} \mathcal{O}$

Surely this could have been prevented?

Easy fix:

- Protocol assumed $[r]G_2 = a_2 + [c]I$, only for a **single** I
- Fix: check if the order of I is ℓ
  - i.e. check $[\ell]I \stackrel{?}{=} \mathcal{O}$
  - Fun fact: this check makes the verification $2\times$ slower

Why didn't they validate points?

Why didn't they validate points?

Look at the docs:

How do I validate Curve25519 public keys?

Don't. The Curve25519 function was carefully designed to allow all 32-byte strings as Diffie-Hellman public keys. Relevant lower-level facts: the number of points of this elliptic curve over the base field is 8 times the prime 2^252 + 27742317777372353535851937790883648493; the number of points of the twist is 4 times the prime 2^253 -

(highlight added by me)

Surely this could have been prevented?

Easy fix:

- Protocol assumed $[r]G_2 = a_2 + [c]I$, only for a **single** I
- Fix: check if the order of I is ℓ
  - i.e. check $[\ell]I \stackrel{?}{=} \mathcal{O}$
- Better fix: **use a prime order curve**

Outline

Introduction

Preliminaries

Cofactor security

ECC implementation

Results

Goal of this thesis

What is the actual performance benefit of Curve25519 over traditional (Weierstrass) curves?

Our contribution

Our research:

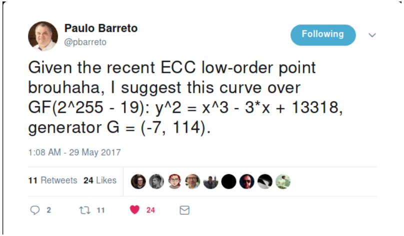
- Implement variable base-point scalar multiplication
  - That is the algorithm for computing $[k]P$,
  - for a prime-order curve,
  - that looks similar to Curve25519,
  - on Sandy Bridge microarchitecture

Our contribution

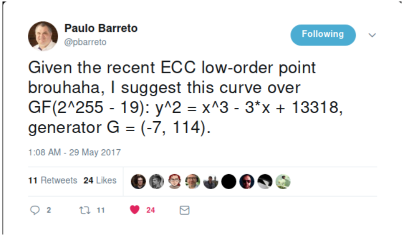
Our research:

- Implement variable base-point scalar multiplication
  - That is the algorithm for computing $[k]P$,
  - for a prime-order curve,
  - that looks similar to Curve25519,
  - on Sandy Bridge microarchitecture
- Compare performance with Curve25519 (Sandy2x)

Selecting a curve

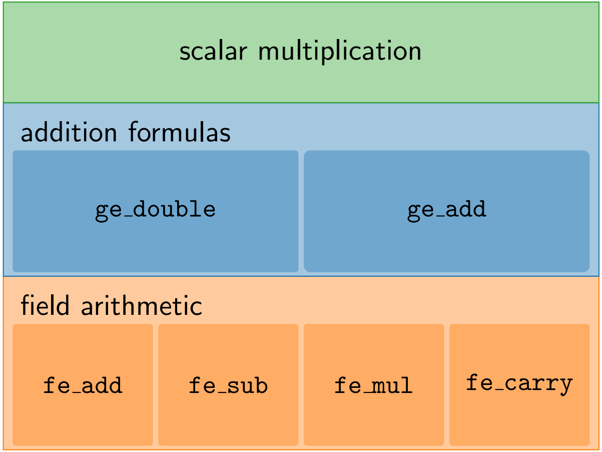


- I.e. $\mathcal{E} : y^2 = x^3 - 3x + 13318$, defined over $\mathbb{F}_{2^{255}-19}$.



- I.e. $\mathcal{E} : y^2 = x^3 - 3x + 13318$, defined over $\mathbb{F}_{2^{255}-19}$.
- Prime order curve; same field as Curve25519

Scalar multiplication overview



Field element representation

- Use double-precision floating points

Field element representation

- Use double-precision floating points
- Allows $4\times$ vectorized operations using SIMD instructions

Field element representation

- Use double-precision floating points
- Allows $4\times$ vectorized operations using SIMD instructions
- Radix-$2^{21.25}$ redundant representation

Field element representation

- Use double-precision floating points
- Allows $4\times$ vectorized operations using SIMD instructions
- Radix-$2^{21.25}$ redundant representation
- Use 12 limbs to represent 255-bit numbers

Field element representation

- Use double-precision floating points
- Allows $4\times$ vectorized operations using SIMD instructions
- Radix-$2^{21.25}$ redundant representation
- Use 12 limbs to represent 255-bit numbers
  - I.e. $f = f_0 + f_1 + ... + f_{11}$

Field arithmetic

- Carry
  - $\textsc{top}(f_i)$: force loss of precision
  - Then, move “high” bits to next limb

Field arithmetic

- Carry
  - $\textsc{top}(f_i)$: force loss of precision
  - Then, move "high" bits to next limb
- Addition
  - $(f+g)_i = f_i + g_i$
  - $(f-g)_i = f_i - g_i$

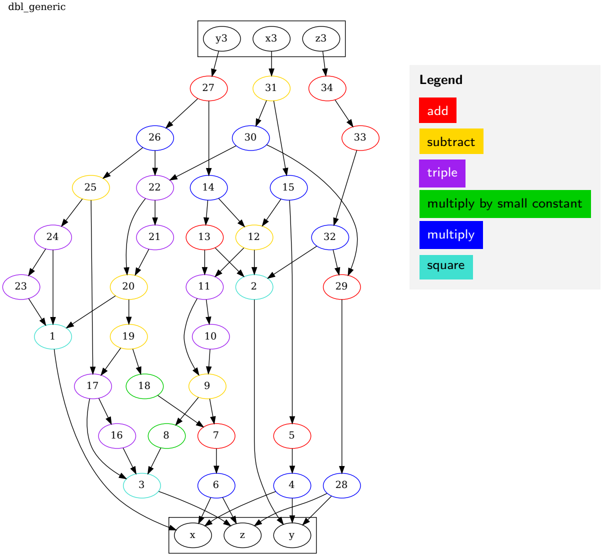
Field arithmetic

- Carry
  - $\textsc{top}(f_i)$: force loss of precision
  - Then, move "high" bits to next limb
- Addition
  - $(f+g)_i = f_i + g_i$
  - $(f-g)_i = f_i - g_i$
- Multiplication
  - $(f \cdot g)_k = \sum_{i+j=k} f_i g_i + \sum_{i+j=k+12} \left(2^{-255} \cdot 19\right) f_i g_i$
  - Optimized using Karatsuba's multiplication

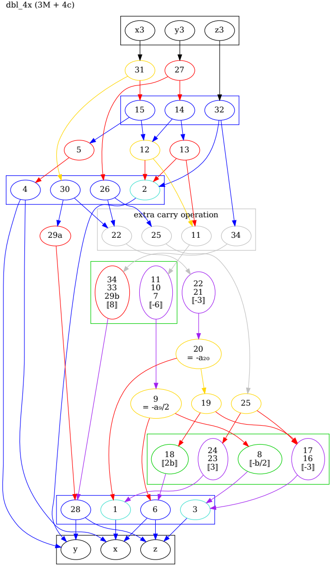
Addition formulas

- Use Renes-Costello-Batina formulas
- Rewrite using graphs into vectorized operations
- Implement using field arithmetic functions

Point doubling



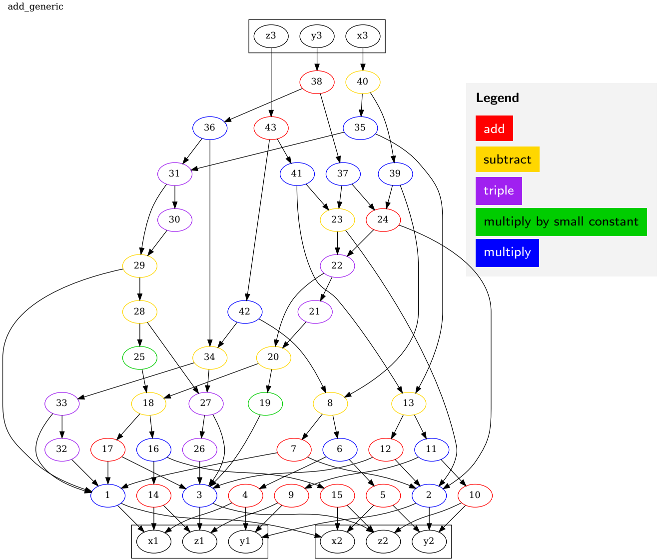
Point doubling



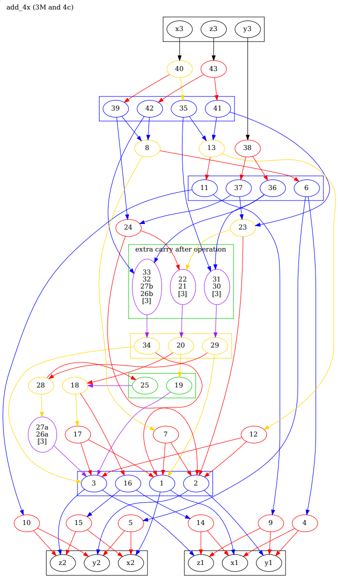
Legend

- add
- subtract
- triple
- multiply by small constant
- multiply
- square

Point addition



Point addition



Legend

- add
- subtract
- triple
- multiply by small constant
- multiply

Scalar multiplication

- Use left-to-right double-and-add

Scalar multiplication

- Use left-to-right double-and-add
  - Optimization: use signed window method ($w = 5$)

Scalar multiplication

- Use left-to-right double-and-add
  - Optimization: use signed window method ($w = 5$)
- Uses $263 \cdot \textbf{double} + 59 \cdot \textbf{add}$ operations

Outline

Compared to Curve25519

Table: Cycle counts for Sandy2x and this work.

Implementation	Sandy Bridge	Ivy Bridge	Haswell
Curve25519 (Sandy2x)	159kcc	157kcc	–
this work	390kcc	383kcc	340kcc

Compared to Curve25519

Table: Cycle counts for Sandy2x and this work.

Implementation	**Sandy Bridge**	**Ivy Bridge**	**Haswell**
Curve25519 (Sandy2x)	159kcc	157kcc	–
this work	390kcc	383kcc	340kcc

Conclusion: about **2.5**$\times$ slower

Thank you! I

Acknowledgements <3:

- Peter, (+the department, Marrit, Judith, Gerdriaan)
- The LLVM project (especially for `llvm-mca`)
- Olivier (from SNT; for lending their Sandy Bridge machine)

Thank you! I

Acknowledgements <3:

- Peter, (+the department, Marrit, Judith, Gerdriaan)
- The LLVM project (especially for `llvm-mca`)
- Olivier (from SNT; for lending their Sandy Bridge machine)

Stuff I left out:

- Ristretto
- Politics
- *Many* implementation details

Thank you! II

The code is at `https://github.com/dsprenkels/curve13318`

Extra reading:

- My thesis: `https://dsprenkels.com/files/thesis-20190311.pdf`
- Monero vulnerability (1): `https://nickler.ninja/blog/2017/05/23/exploiting-low-order-generators-in-one-time-ring-signatures/`
- Monero vulnerability (2): `https://moderncrypto.org/mail-archive/curves/2017/000898.html`

Find me through:

- Email: hello@dsprenkels.com
- PGP key: `951D 6F6E C19E 5D87 1A61  A7F4 1445 C075 FFD5 68CD`

References I

Barreto, P.S.L.M.: on Twitter (May 2017), https://twitter.com/pbarreto/status/869103226276134912

Bernstein, D.J.: Curve25519: New Diffie-Hellman speed records. pp. 207–228 (2006), https://cr.yp.to/ecdh/curve25519-20060209.pdf

Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. pp. 124–142 (2011), https://ed25519.cr.yp.to/ed25519-20110926.pdf

Chou, T.: Sandy2x: New Curve25519 speed records. pp. 145–160 (2016), https://www.win.tue.nl/~tchou/papers/sandy2x.pdf

Genkin, D., Valenta, L., Yarom, Y.: May the Fourth Be With You: A microarchitectural side channel attack on several real-world applications of Curve25519. pp. 845–858 (2017), https://eprint.iacr.org/2017/806.pdf

Karatsuba, A., Ofman, Y.: Multiplication of many-digital numbers by automatic computers. Dokl. Akad. Nauk SSSR 145(2), 293–294 (1962), http://www.mathnet.ru/php/getFT.phtml?jrnid=dan&paperid=26729&what=fullt&option_lang=eng

References II

Kaufmann, T., Pelletier, H., Vaudenay, S., Villegas, K.: When constant-time source yields variable-time binary: Exploiting Curve25519-donna built with MSVC 2015. pp. 573–582 (2016), https://infoscience.epfl.ch/record/223794/files/32_1.pdf

Koblitz, N.: Elliptic curve cryptosystems. Math. Comp. 48, 209–209 (1987), https://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf

luigi1111, "fluffypony" Spagni, R.: Disclosure of a major bug in cryptonote based currencies (May 2017), https://src.getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html

Miller, V.S.: Use of elliptic curves in cryptography. pp. 417–426 (1986), https://www.researchgate.net/profile/Victor_Miller/publication/227128293_Use_of_Elliptic_Curves_in_Cryptography/links/0c96052e065c94b47c000000/Use-of-Elliptic-Curves-in-Cryptography.pdf

Perrin, T.: Subject: [curves] CryptoNote and equivalent points (May 2017), https://moderncrypto.org/mail-archive/curves/2017/000898.html

Renes, J., Costello, C., Batina, L.: Complete addition formulas for prime order elliptic curves. pp. 403–428 (2016), http://eprint.iacr.org/2015/1060

References III

Schnorr, C.P.: Efficient signature generation by smart cards 4(3), 161–174 (Jan 1991), https://www.researchgate.net/profile/Claus_Schnorr/publication/227088517_Efficient_signature_generation_by_smart_cards/links/0046353849579ce09c000000/Efficient-signature-generation-by-smart-cards.pdf

Double-and-add algorithm

```
function DoubleAndAdd(k, P)                 ▷ Compute [k]P
    R ← 𝒪
    for i from n − 1 down to 0 do
        R ← [2]R                            ▷ Doubling
        if k_i = 1 then
            R ← R + P                       ▷ Addition
        else
            R ← R + 𝒪                       ▷ Addition
        end if
    end for
    return R
end function
```

Fixed-window double-and-add

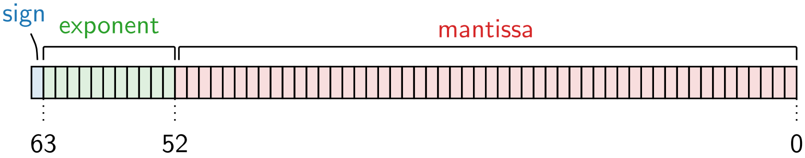
```
function FixedWindow(k, P)                          ▷ Compute [k]P
    k' ← Windows_w(k)
    Precompute ([2]P, ..., [2^w − 1]P)
    R ← O
    for i from n/w − 1 down to 0 do
        for j from 0 to w − 1 do
            R ← [2]R                                ▷ w doublings
        end for
        if k'_i ≠ 0 then
            R ← R + [k'_i]P                         ▷ Addition
        else
            R ← R + O                               ▷ Addition
        end if
    end for
    return R
end function
```

Signed double-and-add

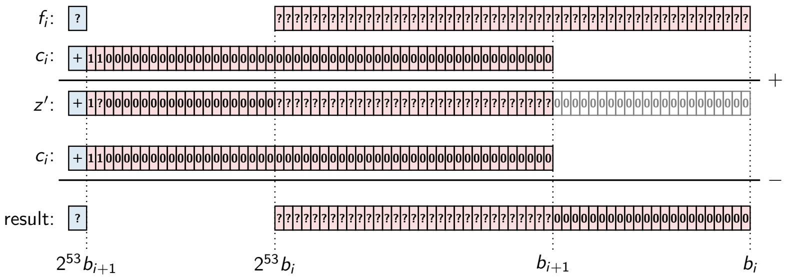
```
function SignedFixedWindow(k, P)                        ▷ Compute [k]P
    k' ← RecodeSigned(Windows_w(k))
    Precompute ([2]P, ..., [2^{w−1}]P)
    R ← O
    for i from n/w − 1 down to 0 do
        for j from 0 to w − 1 do
            R ← [2]R                                    ▷ w doublings
        end for
        if k'_i > 0 then
            R ← R + [k'_i]P                             ▷ Addition
        else if k'_i < 0 then
            R ← R − [−k'_i]P                            ▷ Addition
        else
            R ← R + O                                   ▷ Addition
        end if
    end for
    return R
end function
```

Implemented signed double-and-add

function $\textsc{ScalarMultiplication}(k, P)$ ▷ Compute $[k]P$
  $\mathbf{T} \leftarrow (\mathcal{O}, P, \ldots, [16]P)$ ▷ Precompute $([2]P, \ldots, [16]P)$
  $k' \leftarrow \textsc{RecodeSigned}(\textsc{Windows}_5(k))$
  $R \leftarrow \mathcal{O}$
  for i **from** 50 **down to** 0 **do**
    for j **from** 0 **to** 4 **do**
      $R \leftarrow [2]R$ ▷ 5 doublings
    end for
    if $k'_i < 0$ **then**
      $R \leftarrow R - \mathbf{T}_{-k'_i}$ ▷ Addition
    else
      $R \leftarrow R + \mathbf{T}_{k'_i}$ ▷ Addition
    end if
  end for
  return R ▷ $R = (X_R : Y_R : Z_R)$
end function


sign
exponent
mantissa
63
52
0

Depiction of TOP(f)



Signed windows

$$k = \underbrace{1011}_{k'_3}\,\underbrace{0010}_{k'_2}\,\underbrace{0110}_{k'_1}\,\underbrace{1110}_{k'_0}$$

Signed window recoding

$$\begin{array}{cccccc} k = & & 1011 & 0010 & 0110 & 1110 \\ & & \downarrow & \downarrow & \downarrow & \downarrow \\ & \underbrace{1}_{k''_4} & \underbrace{-101}_{k''_3} & \underbrace{010}_{k''_2} & \underbrace{111}_{k''_1} & \underbrace{-010}_{k''_0} \end{array}$$