Radbound University

The complete cost of cofactor $h = 1$

Implementing Weierstrass curves with complete formulas

Peter Schwabe Daan Sprenkels

18 December 2019

Radboud University,
peter@cryptojedi.org, daan@dsprenkels.com

Introduction

Some history

- Traditionally, we use various different Weierstraß curves
- Considered unsafe because of incomplete formulas
- 2006: Curve25519 [Ber06] proposed as better alternative

Interesting cases of cofactor insecurity in protocols (mis)using Curve25519:

- 2017: [IfS17] reported major vulnerability in Monero

Cofactor (in)security

Interesting cases of cofactor insecurity in protocols (mis)using Curve25519:

- 2017: [IfS17] reported major vulnerability in Monero
- 2019: [CJ19] found three other vulnerabilities caused by cofactor insecurity

The Monero vulnerability

- Transaction involves a *ring signature*
- Trivial case: ring size is 1

The Monero vulnerability

- Transaction involves a *ring signature*
- Trivial case: ring size is 1
- Double-spending is prevented by a *key image* I

The Monero vulnerability

- Transaction involves a *ring signature*
- Trivial case: ring size is 1
- 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*
- Trivial case: ring size is 1
- 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*
- Trivial case: ring size is 1
- 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 (simplified)

- 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 random $u \in_R h\mathbb{Z}_\ell$
  - Compute commitment $a_2 = [u]G_2$; $c = H(m, a_1, a_2)$; $r = u + cx$
  - Output signature $s = (a_1, a_2, r)$

Monero transactions (simplified)

- 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 random $u \in_R h\mathbb{Z}_\ell$
  - Compute commitment $a_2 = [u]G_2$; $c = H(m, a_1, a_2)$; $r = u + cx$
  - 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 $I \neq I'$.

- **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.**

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

- **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.**

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

- **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.**

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

- **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
- Not the case for Curve25519

Surely this could have been prevented?

Easy fix:

- Protocol assumed $[r]G_2 = a_2 + [c]I$, only for a **single** I
- Not the case for Curve25519
- 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
- Not the case for Curve25519
- 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
- Not the case for Curve25519
- 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?

My *guess*:

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**

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**
- Best fix: **use Ristretto** [Ham15, dVGT+19]

Research question

- Curve25519: nontrivial cofactor
- Weierstraß: slow or incomplete formulas

Research question

- Curve25519: nontrivial cofactor
- Weierstraß: slow or incomplete formulas
- But how much slower *exactly*?

Research question

What is the actual performance benefit of Curve25519 over traditional (Weierstrass) curves when using complete formulas?

Our contribution

Our research:

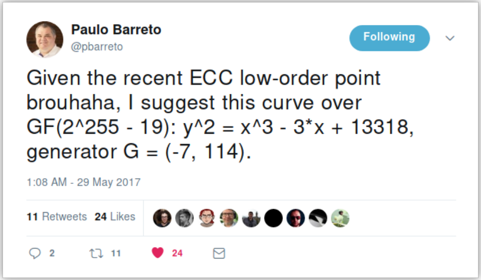
- Implement variable base-point scalar multiplication
  - for a prime-order curve,
  - that looks similar to Curve25519,
  - using complete formulas,
  - on Sandy Bridge, Haswell, and Cortex M4.

Our research:

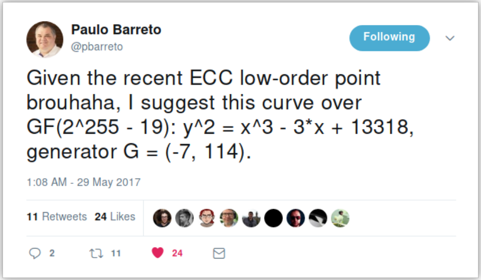
- Implement variable base-point scalar multiplication
  - for a prime-order curve,
  - that looks similar to Curve25519,
  - using complete formulas,
  - on Sandy Bridge, Haswell, and Cortex M4.
- Compare performance with Curve25519

Selecting a curve

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

Implementation

- Use left-to-right fixed-window method ($w = 5$)

Scalar multiplication

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

Addition formulas

Use the Renes-Costello-Batina addition formulas [RCB16]

- Complete formulas (no exceptions)
- No optimized software implementations published

Field arithmetic

Sandy Bridge

- AVX: has 2-way parallel 64-bit integer arithmetic
- AVX: has 4-way parallel floating-point arithmetic
- $\rightarrow$ use radix-$2^{21.25}$ representation based on [Ber04]

Field arithmetic

Sandy Bridge

- AVX: has 2-way parallel 64-bit integer arithmetic
- AVX: has 4-way parallel floating-point arithmetic
- $\rightarrow$ use radix-$2^{21.25}$ representation based on [Ber04]

Haswell

- AVX2: has 4-way parallel 64-bit integer arithmetic
- $\rightarrow$ use radix-$2^{25.5}$ representation based on [BS12]

Field arithmetic

Sandy Bridge

- AVX: has 2-way parallel 64-bit integer arithmetic
- AVX: has 4-way parallel floating-point arithmetic
- $\rightarrow$ use radix-$2^{21.25}$ representation based on [Ber04]

Haswell

- AVX2: has 4-way parallel 64-bit integer arithmetic
- $\rightarrow$ use radix-$2^{25.5}$ representation based on [BS12]

Cortex-M4

- Has powerful `umlal` and `umaal` instructions
- $\rightarrow$ use packed representation from [HL19]

Application of formulas

Sandy Bridge + Haswell

- Vectorize all multiplications and some other ops
- Shuffles etc. all implemented by hand
- Inline all the calls to field arithmetic

Application of formulas

Sandy Bridge + Haswell

- Vectorize all multiplications and some other ops
- Shuffles etc. all implemented by hand
- Inline all the calls to field arithmetic

Cortex-M4

- Size-constrained device
- One-to-one implementation of formulas
- No function inlining

Results

Benchmarks

Figure: cycle counts in kcc

Implementation	**SB**	**H**	**M4**
Chou16 [Cho16]	159[a]	156[b]	–
Faz-Hernández-López15 [FL15]	–	156[a]	–
OLHF18 [OLH+18]	–	139[a]	–
Fujii-Aranha19 [FA19]	–	–	907[a]
Haase-Labrique19 [HL19]	–	–	625[a]
Curve13318 (**this work**)	390[b]	205[b]	1 797[b]
slowdown	2.45×	1.47×	2.87×

[a] As reported in the respective publication.

[b] From own measurements.

Future work

- Use formulas from [SM17]
- Benchmark with ristretto255

Thank you!

The code is at `https://github.com/dsprenkels/curve13318-all` (public domain)

Extra reading:

- Paper: `https://dsprenkels.com/files/curve13318.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`

Paulo S. L. M. Barreto.

Tweet, 2017.

https://twitter.com/pbarreto/status/869103226276134912.

Daniel J. Bernstein.

Floating-point arithmetic and message authentication, 2004.

http://cr.yp.to/papers.html#hash127.

Daniel J. Bernstein.

Curve25519: new Diffie-Hellman speed records.

In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography – PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, 2006.

http://cr.yp.to/papers.html#curve25519.

Daniel J. Bernstein and Tanja Lange.

eBACS: ECRYPT Benchmarking of Cryptographic Systems.

https://bench.cr.yp.to/results-sign.html (accessed 2019-10-03).

Daniel J. Bernstein and Peter Schwabe.

NEON crypto.

In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *LNCS*, pages 320–339. Springer, 2012.

http://cryptojedi.org/papers/#neoncrypto.

Tung Chou.

Sandy2x: New Curve25519 speed records.

In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography – SAC 2015*, volume 9566 of *LNCS*, pages 145–160. Springer, 2016.

https://www.win.tue.nl/~tchou/papers/sandy2x.pdf.

Cas Cremers and Dennis Jackson.

Prime, order please! revisiting small subgroup and invalid curve attacks on protocols using Diffie-Hellman.

In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 78–93, 2019.

https://eprint.iacr.org/2019/526.

Henry de Valence, Jack Grigg, George Tankersley, Filippo Valsorda, and Isis Lovecruft.

The ristretto255 group.

IETF CFRG Internet Draft, 2019.

`https://tools.ietf.org/html/draft-hdevalence-cfrg-ristretto-01` (accessed 2019-07-31).

Hayato Fujii and Diego F. Aranha.

Curve25519 for the Cortex-M4 and Beyond.

In Tanja Lange and Orr Dunkelman, editors, *Progress in Cryptology – LATINCRYPT 2017*, volume 11368 of *LNCS*, pages 109–127. Springer, 2019.

http://www.cs.haifa.ac.il/~orrd/LC17/paper39.pdf.

Armando Faz-Hernández and Julio López.

Fast implementation of Curve25519 using AVX2.

In Kristin Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology – LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 329–345. Springer, 2015.

Mike Hamburg.

Decaf: Eliminating cofactors through point compression.

In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9215 of *LNCS*, pages 705–723. Springer, 2015.

https://www.shiftleft.org/papers/decaf/.

Björn Haase and Benoît Labrique.

AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT.

IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 1–48, 2019.

https://tches.iacr.org/index.php/TCHES/article/view/7384.

luigi1111 and Riccardo “fluffypony” Spagni.

Disclosure of a major bug in CryptoNote based currencies.

Post on the Monero website, 2017.

`https://www.getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html` (accessed 2019-07-31).

Thomaz Oliveira, Julio López, Hüseyin Hışıl, Armando Faz-Hernández, and Francisco Rodríguez-Henríquez.

How to (Pre-)Compute a Ladder.

In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography – SAC 2017*, volume 10719 of *LNCS*, pages 172–191. Springer, 2018.

https://eprint.iacr.org/2017/264.pdf.

Joost Renes, Craig Costello, and Lejla Batina.

Complete addition formulas for prime order elliptic curves.

In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – Eurocrypt 2016*, volume 9230 of *LNCS*, pages 403–428. Springer, 2016.

http://eprint.iacr.org/2015/1060.

Ruggero Susella and Sofia Montrasio.

A compact and exception-free ladder for all short Weierstrass elliptic curves.

In Kerstin Lemke-Rust and Michael Tunstall, editors, *Smart Card Research and Advanced Applications*, volume 10146 of *LNCS*, pages 156–173. Springer, 2017.

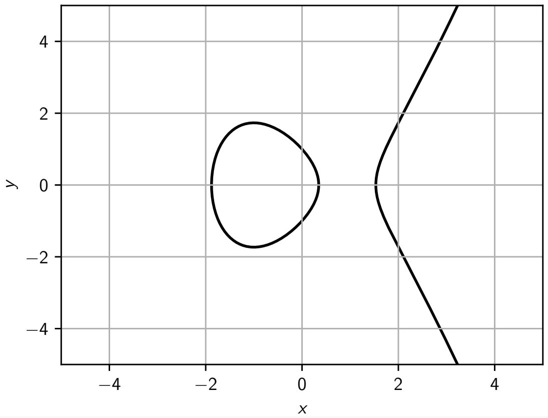
Preliminaries

Elliptic curves

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

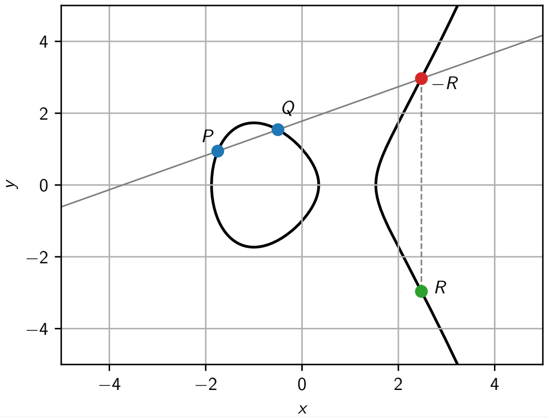
Elliptic curves

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



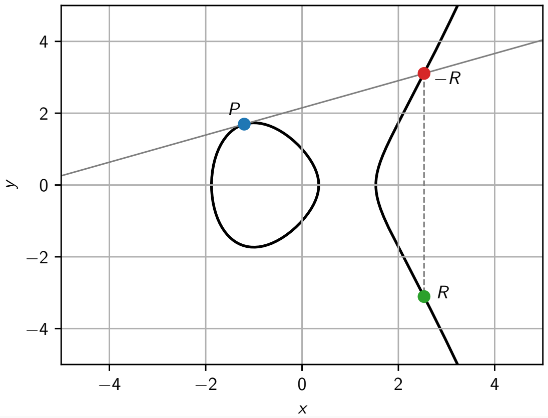
Elliptic curves: addition

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



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

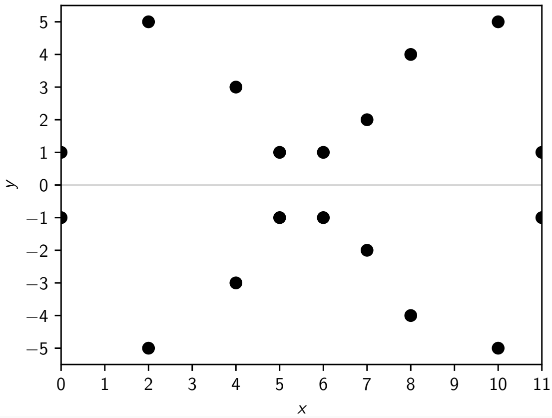
- Coordinates include *the point at infinity* $\mathcal{O}$
  - Define $P + \mathcal{O} = P$
- Curve equation: $\mathcal{E} : y^2 = x^3 + ax + b$

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

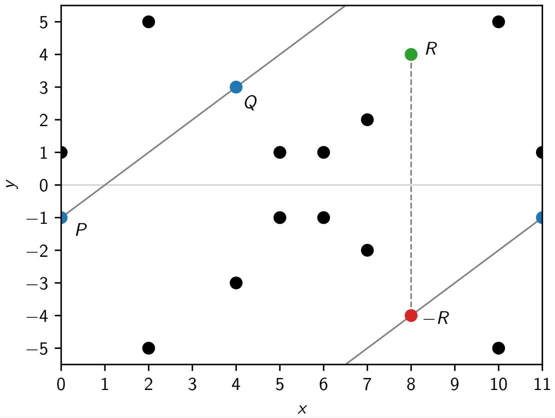
Elliptic curves: actually

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



Elliptic curves: actual addition

$\mathcal{E} : y^2 = x^3 - 3x + 1$ 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: $[k]P = \underbrace{P + P + \ldots + P}_{k \text{ times}}$

- 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

Double-and-add

Double-and-add algorithm

function $\textsc{DoubleAndAdd}(k, P)$ ▷ Compute $[k]P$
    $R \leftarrow \mathcal{O}$
    for i **from** $n-1$ **down to** 0 **do**
        $R \leftarrow [2]R$ ▷ Doubling
        if $k_i = 1$ **then**
            $R \leftarrow R + P$ ▷ Addition
        else
            $R \leftarrow R + \mathcal{O}$ ▷ Addition
        end if
    end for
    return R
end function

Fixed-window double-and-add

function $\textsc{FixedWindow}(k, P)$ ▷ Compute $[k]P$

  $k' \leftarrow \textsc{Windows}_w(k)$

  Precompute $([2]P, \ldots, [2^w - 1]P)$

  $R \leftarrow \mathcal{O}$

  for i **from** $\frac{n}{w} - 1$ **down to** 0 **do**

    for j **from** 0 **to** $w - 1$ **do**

      $R \leftarrow [2]R$ ▷ w doublings

    end for

    if $k'_i \neq 0$ **then**

      $R \leftarrow R + [k'_i]P$ ▷ Addition

    else

      $R \leftarrow R + \mathcal{O}$ ▷ Addition

    end if

  end for

  return R

end function

Signed double-and-add

function $\textsc{SignedFixedWindow}(k, P)$ ▷ Compute $[k]P$

  $k' \leftarrow \textsc{RecodeSigned}(\textsc{Windows}_w(k))$

  Precompute $([2]P, \ldots, [2^{w-1}]P)$

  $R \leftarrow \mathcal{O}$

  for i **from** $\frac{n}{w} - 1$ **down to** 0 **do**

    for j **from** 0 **to** $w - 1$ **do**

      $R \leftarrow [2]R$ ▷ w doublings

    end for

    if $k'_i > 0$ **then**

      $R \leftarrow R + [k'_i]P$ ▷ Addition

    else if $k'_i < 0$ **then**

      $R \leftarrow R - [-k'_i]P$ ▷ Addition

    else

      $R \leftarrow R + \mathcal{O}$ ▷ Addition

    end if

  end for

  return R

end function

Implemented signed double-and-add

function $\textsc{ScalarMultiplication}(k, P)$ ▷ Compute $[k]P$

  $\mathbf{T} \leftarrow (\mathcal{O}, P, \dots, [16]P)$ ▷ Precompute $([2]P, \dots, [16]P)$

  $k' \leftarrow \textsc{RecodeSigned}(\textsc{Windows}_5(k))$

  $R \leftarrow \mathcal{O}$

  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

Signed windows

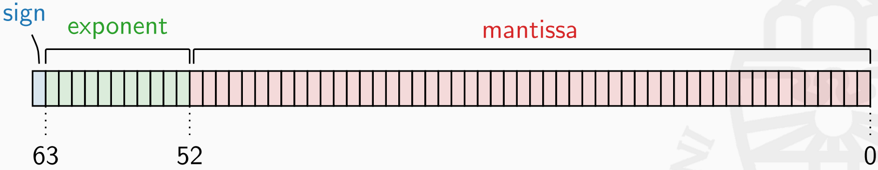
$$k = \underbrace{1011}_{k'_3}\ \underbrace{0010}_{k'_2}\ \underbrace{0110}_{k'_1}\ \underbrace{1110}_{k'_0}$$

Signed window recoding

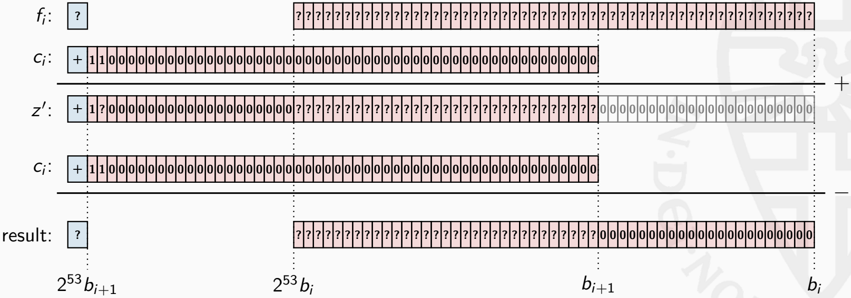
$$k = 1011\ \ 0010\ \ 0110\ \ 1110$$

$$\underbrace{1}_{k''_4}\ \underbrace{-101}_{k''_3}\ \underbrace{010}_{k''_2}\ \underbrace{111}_{k''_1}\ \underbrace{-010}_{k''_0}$$

Sandy Bridge details


sign
exponent
mantissa
63
52
0

Depiction of top(f)



Sandy Bridge: field element representation

- Use double-precision floating points

Sandy Bridge: field element representation

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

Sandy Bridge: field element representation

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

Sandy Bridge: 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

Sandy Bridge: 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. $f = f_0 + f_1 + ... + f_{11}$

Sandy Bridge: field element representation

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

Sandy Bridge: field element representation

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

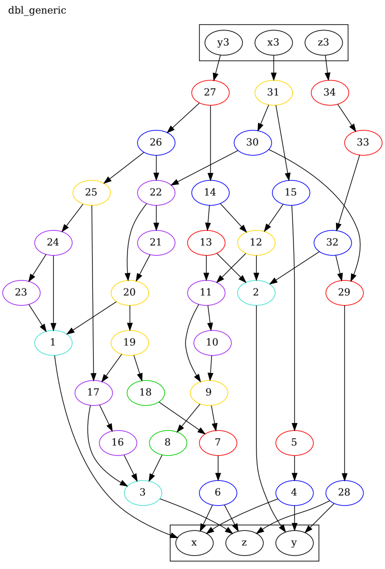
Sandy Bridge: field element representation

- 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

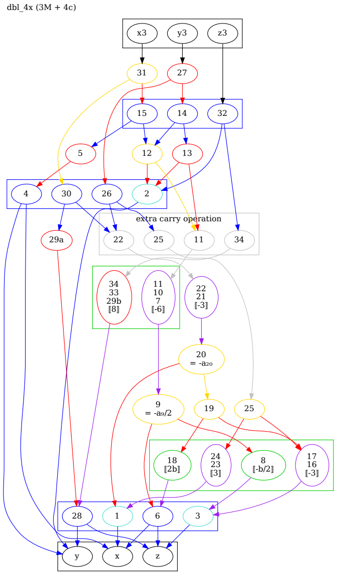
Point doubling



Legend

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

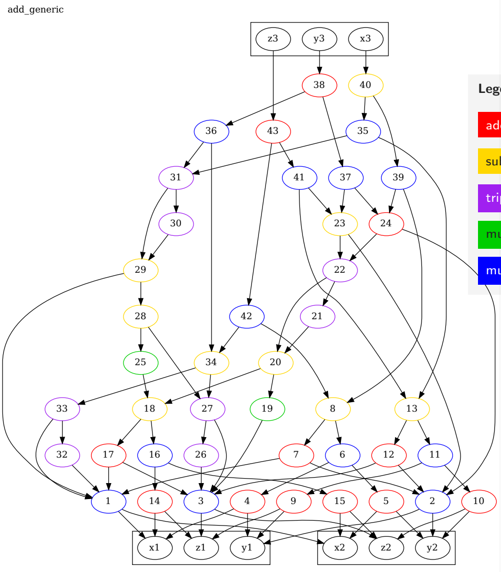
Point doubling



Legend

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

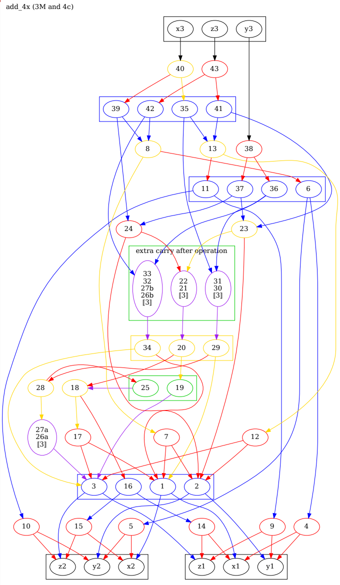
Point addition



Legend

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

Point addition



Legend

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

Figure: Measured cycle counts

Implementation	SB	IB	H	
Chou16 [Cho16]	159 128[a]	156 995[a]	155 823[b]	
Faz-Hernández-Lopez15 [FL15]	–	–	≈ 156 500[c]	
OLHF18 [OLH$^+$18]	–	–	138 963[a]	
Fujii-Aranha19 [FA19]	–	–	–	
Haase-Labrique19 [HL19]	–	–	–	
Curve13318 (**this work**)	389 546[b]	382 966[b]	204 643[b]	1
Ed25519 verify	221 988[d]	206 080[d]	184 052[d]	
slowdown	2.45×	2.44×	1.47×	

[a] As reported in the respective publication.

[b] From own measurements.

[c] As reported in [FL15]. This publication expressed their benchmarks in kcc. As such,
has been padded with zeros.