ECC optimization on Sandy Bridge

The cost of cofactor $h = 1$

Daan Sprenkels
hello@dsprekels.com

Radboud University Nijmegen

1 April 2019
Outline

Introduction
  Preliminaries
  Cofactor security

ECC implementation

Results
Outline

Introduction

  Preliminaries

  Cofactor security

ECC implementation

Results
Elliptic curves

\[ \mathcal{E} : y^2 = x^3 + ax + b \]
Elliptic curves

\[ E : y^2 = x^3 + ax + b \]
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: $\mathcal{E} : y^2 = x^3 + ax + b$
Coordinates include \textit{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

\[ E : y^2 = x^3 - 3x + 1 \text{ defined over } \mathbb{F}_{11} \]
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: \( O \), 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: $[k]P = \underbrace{P + P + \ldots + P}_{k \text{ times}}$
Group arithmetic

- We can do arithmetic with these rules! :)

- Addition: \( P + Q \)

- Subtraction: \( P - Q \)

- Neutral element: \( \mathcal{O} \), i.e. “zero”

- Scalar multiplication: \([k]P = P + P + \ldots + P\)
  \( k \) 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: \[ O \xrightarrow{+P} P \xrightarrow{+P} [2]P \xrightarrow{+P} [3]P \xrightarrow{+P} \ldots \xrightarrow{+P} [n-1]P \xrightarrow{+P} O \]

- 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 \( 4 \mid n \), then there is a point \( T_4: \)
\[
\begin{align*}
\mathcal{O} & \xrightarrow{+T_4} T_4 \xrightarrow{+T_4} [2] T_4 \xrightarrow{+T_4} [3] T_4 \xrightarrow{+T_4} \mathcal{O} \\
\end{align*}
\]
only 4 steps!
Cofactors

▶ If \( n \) is **not** a prime
   Then \( n = h \cdot \ell \)

▶ I.e. small loops are possible:

   E.g. if \( 4 \mid n \), then there is a point \( T_4 \):
   
   \[ \mathcal{O} \xrightarrow{+T_4} T_4 \xrightarrow{+T_4} [2] T_4 \xrightarrow{+T_4} [3] T_4 \xrightarrow{+T_4} \mathcal{O} \]

   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 \( 4 \mid n \), then there is a point \( T_4: \)

\[
\begin{align*}
O & \xrightarrow{T_4} T_4 \xrightarrow{T_4} [2] T_4 \xrightarrow{T_4} [3] T_4 \xrightarrow{T_4} O \\
\text{only 4 steps!}
\end{align*}
\]

- \( h \) 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 $4 \mid n$, then there is a point $T_4$: $\mathcal{O} \xrightarrow{+T_4} T_4 \xrightarrow{+T_4} [2]T_4 \xrightarrow{+T_4} [3]T_4 \xrightarrow{+T_4} \mathcal{O}$
  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

- 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
A brief history...

- 1999: elliptic curves popularized
- 2006: Curve25519 published by Bernstein
  - “Safe” for implementors
- 2014: Monero cryptocurrency
  - Uses Curve25519
- 2017: vulnerability in Monero found
  - Allowed anyone to create coins out of thin air
A brief history...

- 1999: elliptic curves popularized
- 2006: Curve25519 published by Bernstein
  - “Safe” for implementors
  - Super fast
- 2014: Monero cryptocurrency
  - Uses Curve25519
- 2017: vulnerability in Monero found
  - Allowed anyone to create coins out of thin air
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
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 $I$.
- $\text{SIGN}_x(m)$
  - 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 \( I \).

- \( \text{SIGN}_x(m) \)
  - 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 \( s = (a_1, a_2, r) \)

- \( \text{VERIFY}_{P,I}(m, s) \)
  - \( [r]G_1 = a_1 + [c]P \)
  - \( [r]G_2 = a_2 + [c]I \)
  - \( I \) unique?
Attacking Monero signatures

**Challenge.** Find some signature+keypair $a_2, c, r$, and $l$, s.t.

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

where $l \neq l'$. 

Solution. Choose $l' = l + \alpha \cdot T$, where $\alpha | c$ and $[\alpha \cdot T] = O$. 

Correctness. 

$$a_2 + [c]l = a_2 + [c]l'$$

Daan Sprenkels  
ECC optimization on Sandy Bridge  
1 April 2019  
12 / 30
Attacking Monero signatures

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

  \[ [r]G_2 = a_2 + [c]l = a_2 + [c]l', \]

  where $l \neq l'$.

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

▶ **Challenge.** Find some signature+keypair $a_2, c, r, \text{ and } l$, s.t.

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

where $l \neq l'$.

▶ **Solution.** Choose $l' = l + T_\alpha$, where $\alpha | c$ and $[\alpha]T_\alpha = O$.

▶ **Correctness.**

$$a_2 + [c]l' = a_2 + [c](l + T_\alpha)$$
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 $[\alpha]T_\alpha = O$.

- **Correctness.**

\[
a_2 + [c]I' = a_2 + [c](I + T_\alpha) \\
= a_2 + [c]I + \left[ \frac{c}{\alpha} \right] [\alpha]T_\alpha
\]
Attacking Monero signatures

▶ **Challenge.** Find some signature+keypair $a_2, c, r$, and $l$, s.t.

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

where $l \neq l'$.

▶ **Solution.** Choose $l' = l + T_\alpha$, where $\alpha | c$ and $[\alpha] T_\alpha = O$.

▶ **Correctness.**

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

$$= a_2 + [c]l + \left[\frac{c}{\alpha}\right] [\alpha] T_\alpha$$

$$= a_2 + [c]l + \left[\frac{c}{\alpha}\right] O$$
Attacking Monero signatures

▶ **Challenge.** Find some signature-keypair $a_2, c, r$, and $l$, s.t.

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

where $l \neq l'$.

▶ **Solution.** Choose $l' = l + T_\alpha$, where $\alpha | c$ and $[\alpha]T_\alpha = O$.

▶ **Correctness.**

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

$$= a_2 + [c]l + \left[\frac{c}{\alpha}\right] [\alpha] T_\alpha$$

$$= a_2 + [c]l + \left[\frac{c}{\alpha}\right] O$$
Attacking Monero signatures

▶ Challenge. Find some signature-keypair $a_2, c, r$, and $l$, s.t.

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

where $l \neq l'$.

▶ Solution. Choose $l' = l + T_\alpha$, where $\alpha | c$ and $[\alpha]T_\alpha = O$.

▶ Correctness.

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

$$= a_2 + [c]l + \left[\frac{c}{\alpha}\right] [\alpha]T_\alpha$$

$$= a_2 + [c]l + \left[\frac{c}{\alpha}\right] O$$

$$= a_2 + [c]l$$
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 \([r]G_2 = a_2 + [c]l\), only for a **single** \(l\)
- Fix: check if the order of \(l\) is \(\ell\)
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 \(\ell\)
  - i.e. check \([\ell]I \equiv \mathcal{O}\)
Surely this could have been prevented?

Easy fix:

- Protocol assumed $[r]G_2 = a_2 + [c]l$, only for a single $l$
- Fix: check if the order of $l$ is $\ell$
  - i.e. check $[\ell]l \overset{?}{=} O$
  - Fun fact: this check makes the verification $2 \times$ slower
Why didn’t they validate points?

Look at the docs:
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 $2^{253} - 3$.  

(highlight added by me)
Surely this could have been prevented?

Easy fix:

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

Preliminaries

Cofactor security

ECC implementation

Results
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. \( E : y^2 = x^3 - 3x + 13318 \), defined over \( \mathbb{F}_{2^{255} - 19} \).
Selecting a curve

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

- Scalar multiplication
  - Addition formulas
    - ge_double
    - ge_add
  - Field arithmetic
    - fe_add
    - fe_sub
    - fe_mul
    - fe_carry
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. $f = f_0 + f_1 + \ldots + f_{11}$
Field arithmetic

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

Addition

$$f_i + g_i = f_i + g_i$$

Multiplication

$$f_i \cdot g_i = \sum_{i+j=k} f_i g_i + \sum_{i+j=k+1} (2-255 \cdot 19) f_i g_i$$

Optimized using Karatsuba’s multiplication
Field arithmetic

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

- **Carry**
  - 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**
  - $(f \cdot g)_k = \sum_{i+j=k} f_i g_i + \sum_{i+j=k+12} (2^{-255} \cdot 19) f_i g_i$
  - 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

dbl\_generic

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

Daan Sprenkels
ECC optimization on Sandy Bridge
1 April 2019 24 / 30
Point doubling

dbl_generic

x y z

x^3

y^3

z^3

1

2

3 4

5

6

7

8

9

10

11

12

13

14 15

16

17 18

19

20

21

22

23

24

25

26

28

29

30

32

33

Legend

add
subtract
triple
multiply by small constant
multiply
square

daan Sprenkels
ECC optimization on Sandy Bridge
1 April 2019 24 / 30
Point addition

add_generic

x1 y1 z1
x2 y2 z2
x3
40
y3
38
z3
43
1 2 3 4 5
6
7
8
9 10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

Legend
add
subtract
triple
multiply by small constant
multiply
Point addition

add_4x (3M and 4c)

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

ECC optimization on Sandy Bridge
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 \(263 \cdot \text{double} + 59 \cdot \text{add}\) operations
Outline

Introduction

Preliminaries

Cofactor security

ECC implementation

Results
Compared to Curve25519

Table: Cycle counts for Sandy2x and this work.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Sandy Bridge</th>
<th>Ivy Bridge</th>
<th>Haswell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve25519 (Sandy2x)</td>
<td>159kcc</td>
<td>157kcc</td>
<td>–</td>
</tr>
<tr>
<td>this work</td>
<td>390kcc</td>
<td>383kcc</td>
<td>340kcc</td>
</tr>
</tbody>
</table>


**Compared to Curve25519**

Table: Cycle counts for Sandy2x and this work.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Sandy Bridge</th>
<th>Ivy Bridge</th>
<th>Haswell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve25519 (Sandy2x)</td>
<td>159kcc</td>
<td>157kcc</td>
<td>–</td>
</tr>
<tr>
<td>this work</td>
<td>390kcc</td>
<td>383kcc</td>
<td>340kcc</td>
</tr>
</tbody>
</table>

**Conclusion:** about 2.5× 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)

Stuff I left out:

- Ristretto
- Politics
- Many implementation details
Thank you! 

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:


Find me through:

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


Double-and-add algorithm

\begin{verbatim}
function DoubleAndAdd(k, P)
    R ← O
    for i from n − 1 down to 0 do
        R ← [2]R
        if kᵢ = 1 then
            R ← R + P
        else
            R ← R + O
        end if
    end for
    return R
end function
\end{verbatim}

\begin{itemize}
    \item Compute \([k]P\)
    \item Doubling
    \item Addition
    \item Addition
\end{itemize}
Fixed-window double-and-add

function \textsc{FixedWindow}(k, P)

\begin{align*}
& k' \leftarrow \textsc{Windows}_w(k) \\
& \text{Precompute ([2]P, \ldots, [2^w - 1]P)} \\
& R \leftarrow \emptyset \\
& \text{for } i \text{ from } \frac{n}{w} - 1 \text{ down to } 0 \text{ do} \\
& \hspace{1em} \text{for } j \text{ from } 0 \text{ to } w - 1 \text{ do} \\
& \hspace{2em} R \leftarrow [2]R \\
& \hspace{3em} \text{end for} \\
& \hspace{1em} \text{if } k'_i \neq 0 \text{ then} \\
& \hspace{2em} R \leftarrow R + [k'_i]P \\
& \hspace{1em} \text{else} \\
& \hspace{2em} R \leftarrow R + \emptyset \\
& \hspace{1em} \text{end if} \\
& \text{end for} \\
& \text{return } R
\end{align*}

\begin{align*}
\triangleright & \text{ Compute } [k]P \\
\triangleright & \text{ } w \text{ doublings} \\
\triangleright & \text{ Addition} \\
\triangleright & \text{ Addition}
\end{align*}
Signed double-and-add

```plaintext
function SIGNED_FIXED_WINDOW(k, P)
    ⌷ Compute \([k]P\)
    \(k’ \leftarrow \text{RECODE}\_\text{SIGNED}(\text{WINDOWS}_w(k))\)
    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
            \(R \leftarrow [2]R\)
        end for
        if \(k'_i > 0\) then
            \(R \leftarrow R + [k'_i]P\)
        else if \(k'_i < 0\) then
            \(R \leftarrow R - [-k'_i]P\)
        else
            \(R \leftarrow R + \mathcal{O}\)
        end if
    end for
    return \(R\)
end function
```
function \textsc{ScalarMultiplication}(k, P)
\hspace{1em} T \leftarrow (O, P, \ldots, [16]P)
\hspace{1em} k' \leftarrow \text{RecodeSigned}(\text{Windows}_5(k))
\hspace{1em} R \leftarrow O
\hspace{1em} for \ i \text{ from } 50 \ \text{down to } 0 \ \text{do}
\hspace{2em} for \ j \text{ from } 0 \ \text{to } 4 \ \text{do}
\hspace{3em} R \leftarrow [2]R
\hspace{3em} end for
\hspace{2em} if \ k'_i < 0 \ then
\hspace{3em} R \leftarrow R - T_{-k'_i}
\hspace{2em} else
\hspace{3em} R \leftarrow R + T_{k'_i}
\hspace{3em} end if
\hspace{1em} end for
\hspace{1em} return \ R
end function

▷ Compute \([k]P\)
▷ Precompute \(([2]P, \ldots, [16]P)\)

▷ 5 doublings
▷ Addition
▷ Addition

▷ \(R = (X_R : Y_R : Z_R)\)
Depiction of $\text{TOP}(f)$

\[ f_i : \begin{array}{c}
\vdots \\
? \\
\vdots 
\end{array} \]

\[ c_i : \begin{array}{c}
+ 1 \underbrace{000000000000000000000000000000000000000000000000000000000} \\
\vdots \\
\vdots 
\end{array} \]

\[ z_i' : \begin{array}{c}
+ 1 \underbrace{000000000000000000000000000000000000000000000000000000000} \\
\vdots \\
\vdots 
\end{array} \]

\[ c_i : \begin{array}{c}
+ 1 \underbrace{000000000000000000000000000000000000000000000000000000000} \\
\vdots \\
\vdots 
\end{array} \]

result: 
\[ ? \]

\[ 2^{53} b_{i+1} \quad 2^{53} b_i \quad b_{i+1} \quad b_i \]
Signed windows

\[ k = \begin{array}{c}
0111 \\
k'_3 \\
0010 \\
k'_2 \\
0110 \\
k'_1 \\
1110 \\
k'_0 
\end{array} \]
Signed window recoding

\[ k = \begin{array}{cccccc}
1 & 101 & 0010 & 0110 & 1110 \\
& \downarrow & \downarrow & \downarrow & \downarrow \\
1 & -101 & 010 & 111 & -010 \\
\end{array} \]

\[ \begin{array}{cccccc}
k_0'' & k_1'' & k_2'' & k_3'' & k_4'' \\
- & - & 0 & 1 & 1 \\
\end{array} \]