Improved Classical Cryptanalysis of SIKE in Practice

Craig Costello Patrick Longa Michael Naehrig
Joost Renes Fernando Virdia

PKC 2020

Isogeny-based cryptography

- Supersingular Isogeny Key Encapsulation (SIKE)
  $\Longrightarrow$ `https://sike.org/`
- Round-2 candidate in NIST standardization

Isogeny-based cryptography

- Supersingular Isogeny Key Encapsulation (SIKE)
  $\Longrightarrow$ `https://sike.org/`
- Round-2 candidate in NIST standardization
- Cryptanalysis important, recent works include:

Isogeny-based cryptography

- Supersingular Isogeny Key Encapsulation (SIKE)

  $\Longrightarrow$ `https://sike.org/`
- Round-2 candidate in NIST standardization
- Cryptanalysis important, recent works include:
  (1) Classical cryptanalysis by Adj et al.[1]
  (2) Quantum cryptanalysis by Jaques, Schanck and Schrottenloher[2][3]

[1] **canadians**.
[2] **JS19**.
[3] **JS20**.

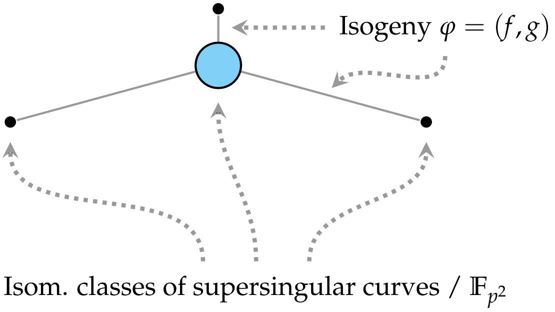
Isogeny-based cryptography

- Supersingular Isogeny Key Encapsulation (SIKE)

  $\Longrightarrow$ `https://sike.org/`
- Round-2 candidate in NIST standardization
- Cryptanalysis important, recent works include:
  (1) Classical cryptanalysis by Adj et al.[1]
  (2) Quantum cryptanalysis by Jaques, Schanck and Schrottenloher[2][3]
- **Today:** Further analysis of *classical* attacks on *SIKE*

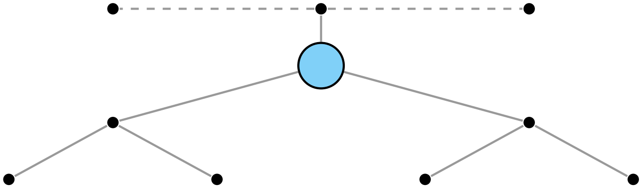
[1] **canadians**.
[2] **JS19**.
[3] **JS20**.

SIDH ($\ell = 2$)

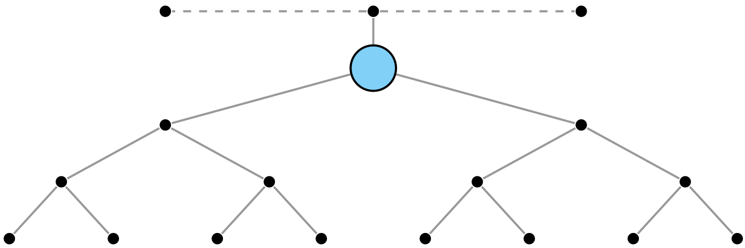


(1) All classes of curves form a connected graph
(2) $\approx p/12$ nodes, each has $\ell + 1$ outgoing isogenies for prime ℓ

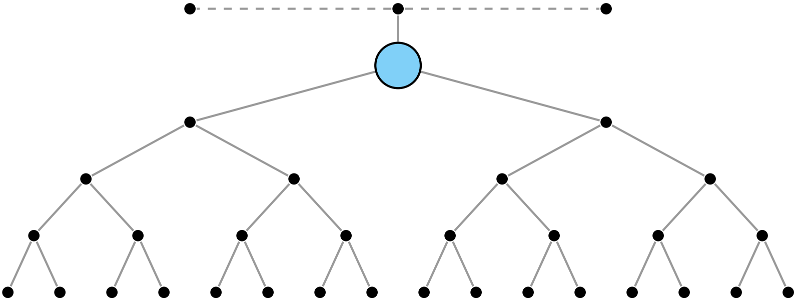
SIDH ($\ell = 2$)



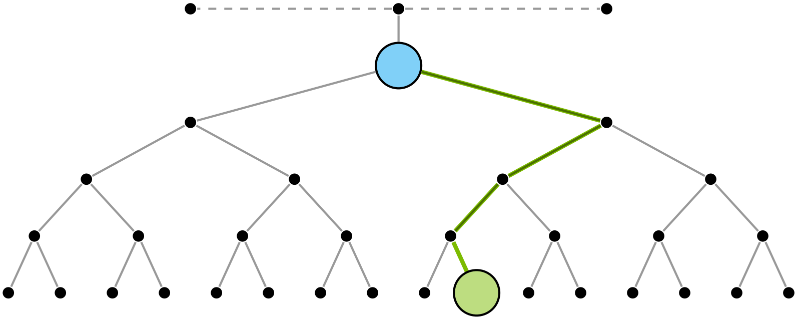
SIDH ($\ell = 2$)



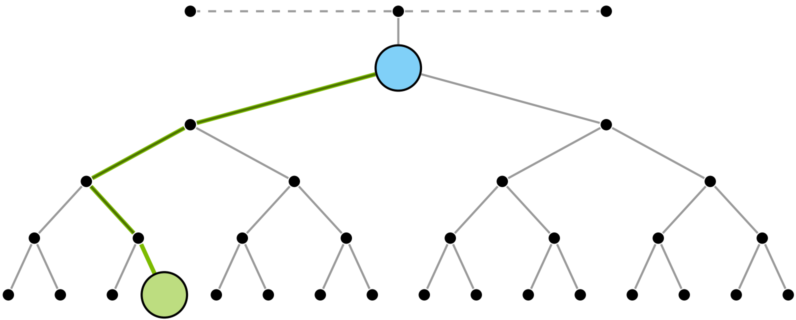
SIDH ($\ell = 2$)



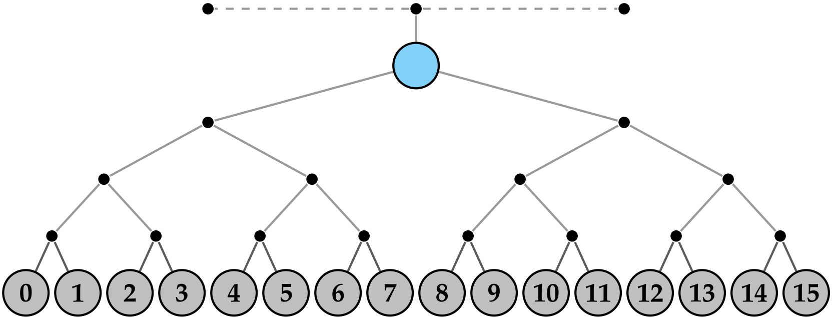
SIDH ($\ell = 2$)



SIDH ($\ell = 2$)



SIDH ($\ell = 2$)

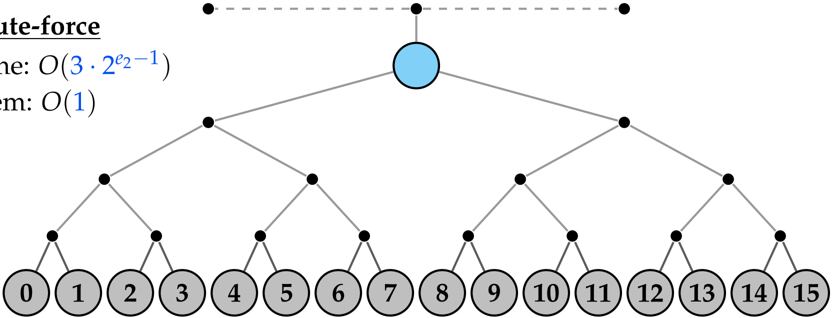


SIDH ($\ell = 2$)

Brute-force

Time: $O(3 \cdot 2^{e_2 - 1})$

Mem: $O(1)$

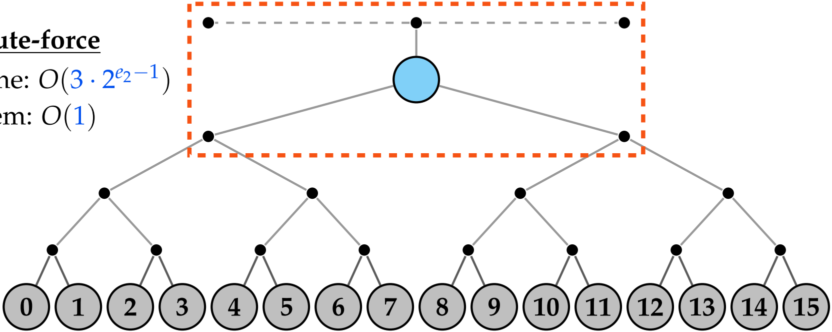


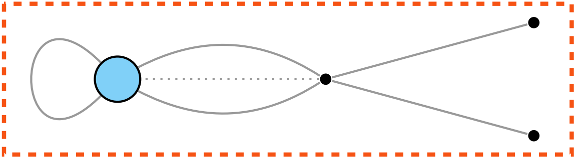
SIKE ($\ell = 2$)

Brute-force

Time: $O(3 \cdot 2^{e_2 - 1})$

Mem: $O(1)$



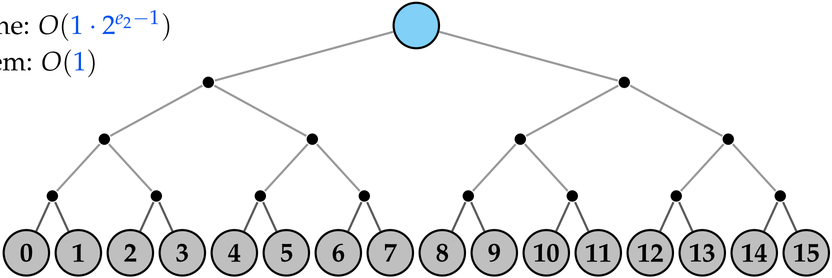


SIKE ($\ell = 2$)

Brute-force

Time: $O(1 \cdot 2^{e_2 - 1})$

Mem: $O(1)$

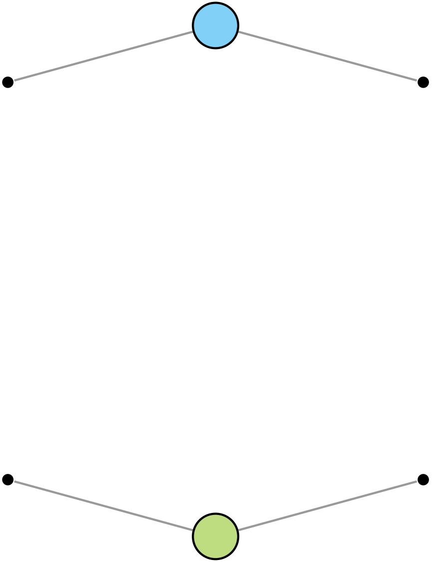


SIKE ($\ell = 2$)

Brute-force

Time: $O(1 \cdot 2^{e_2 - 1})$

Mem: $O(1)$

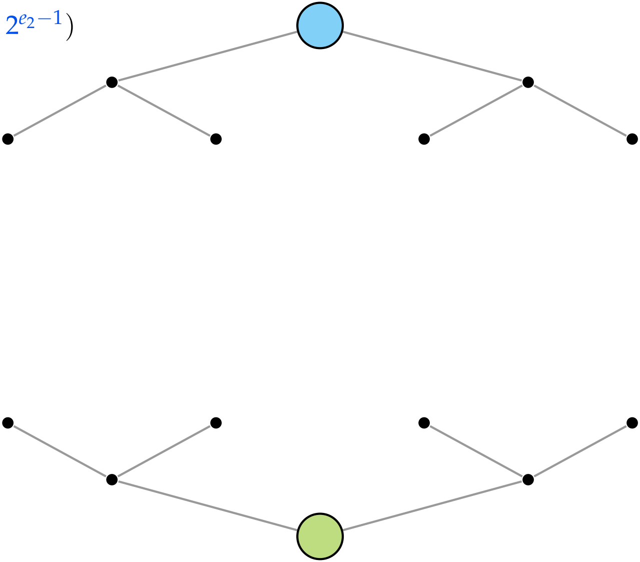


SIKE ($\ell = 2$)

Brute-force

Time: $O(1 \cdot 2^{e_2 - 1})$

Mem: $O(1)$

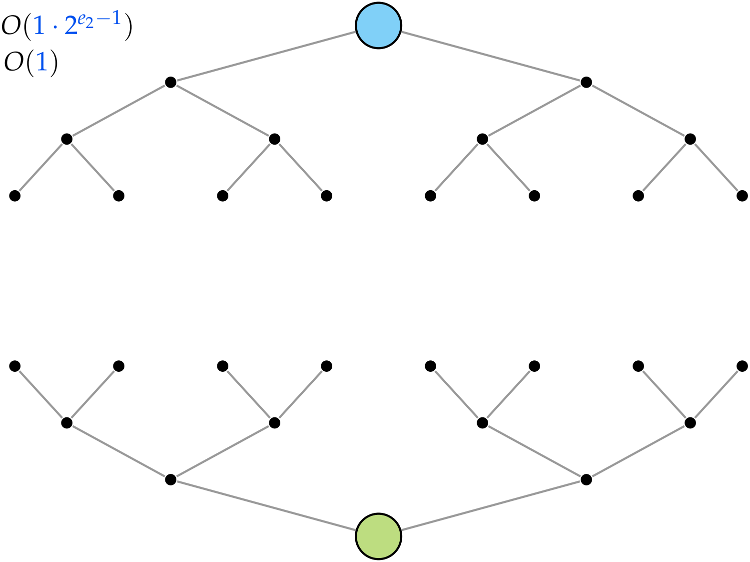


SIKE ($\ell = 2$)

Brute-force

Time: $O(1 \cdot 2^{e_2 - 1})$

Mem: $O(1)$

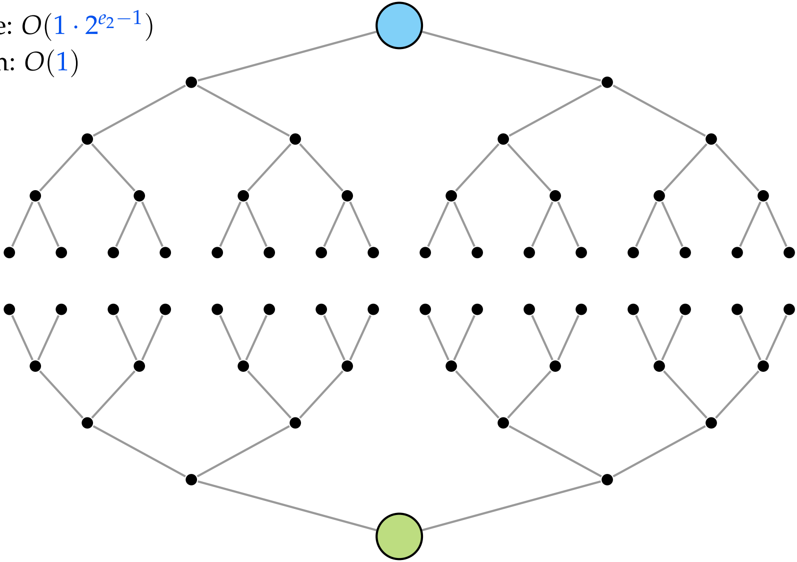


SIKE ($\ell = 2$)

Brute-force

Time: $O(1 \cdot 2^{e_2 - 1})$

Mem: $O(1)$

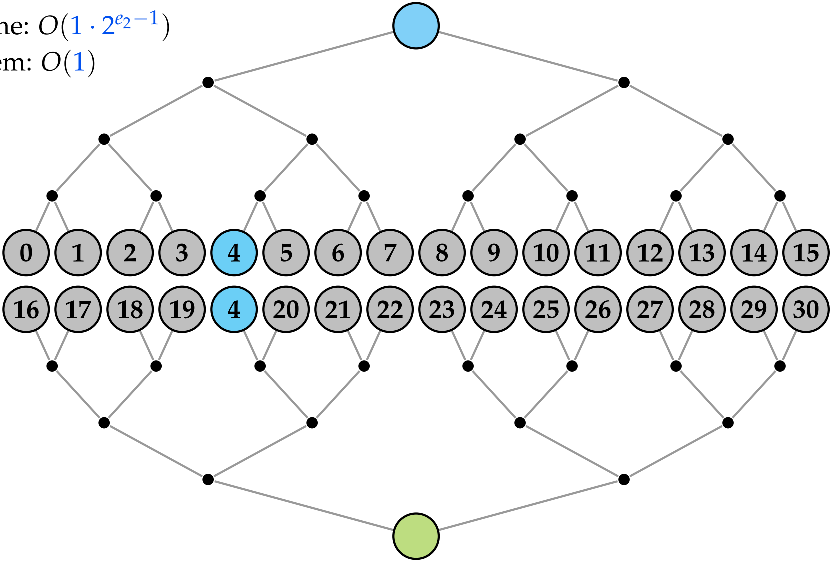


SIKE ($\ell = 2$)

Brute-force

Time: $O(1 \cdot 2^{e_2 - 1})$

Mem: $O(1)$

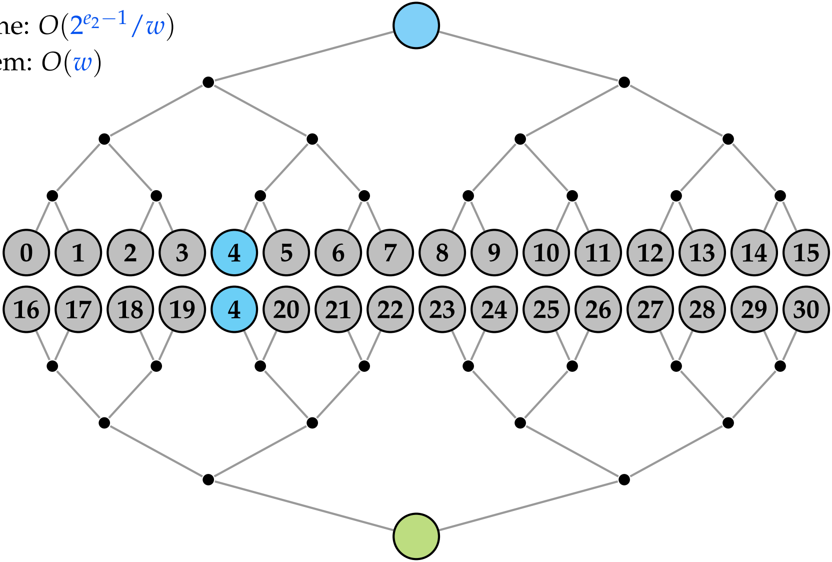


SIKE ($\ell = 2$)

MitM

Time: $O(2^{e_2 - 1}/w)$

Mem: $O(w)$

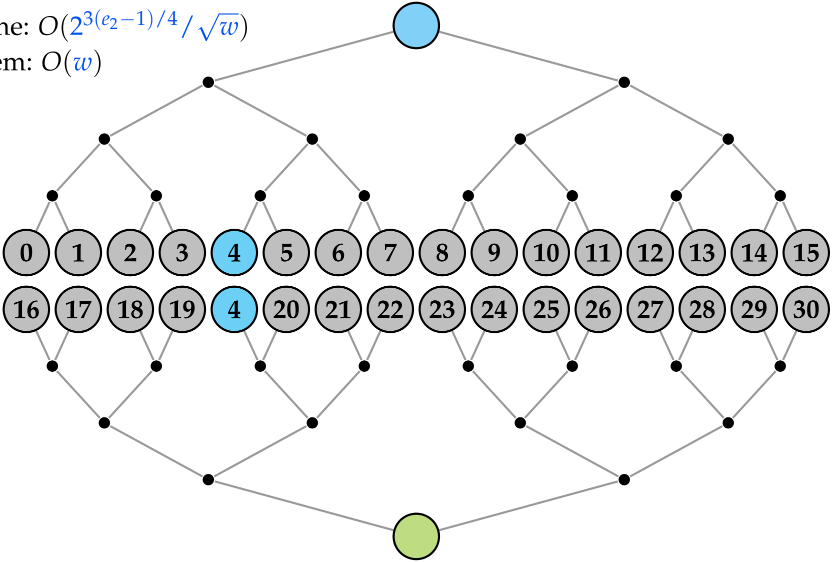


SIKE ($\ell = 2$)

vOW

Time: $O(2^{3(e_2-1)/4}/\sqrt{w})$

Mem: $O(w)$

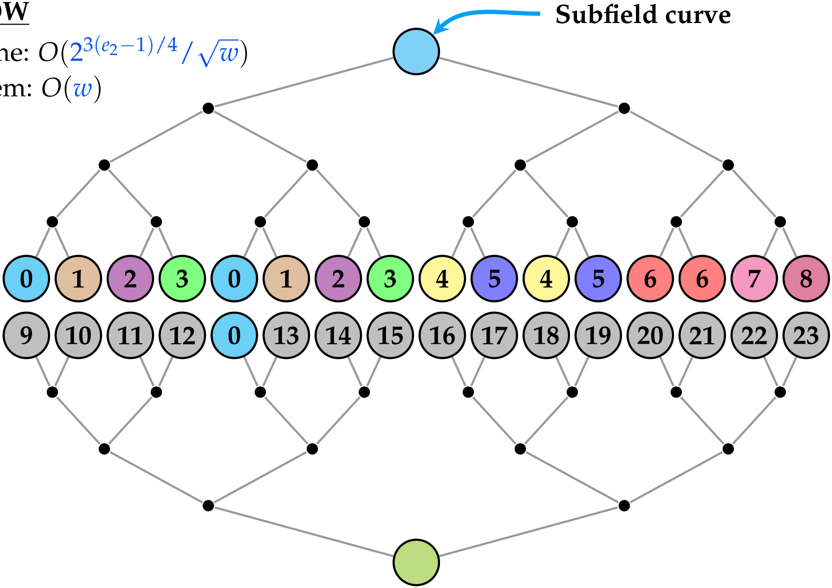


SIKE ($\ell = 2$)

vOW

Time: $O(2^{3(e_2-1)/4}/\sqrt{w})$

Mem: $O(w)$

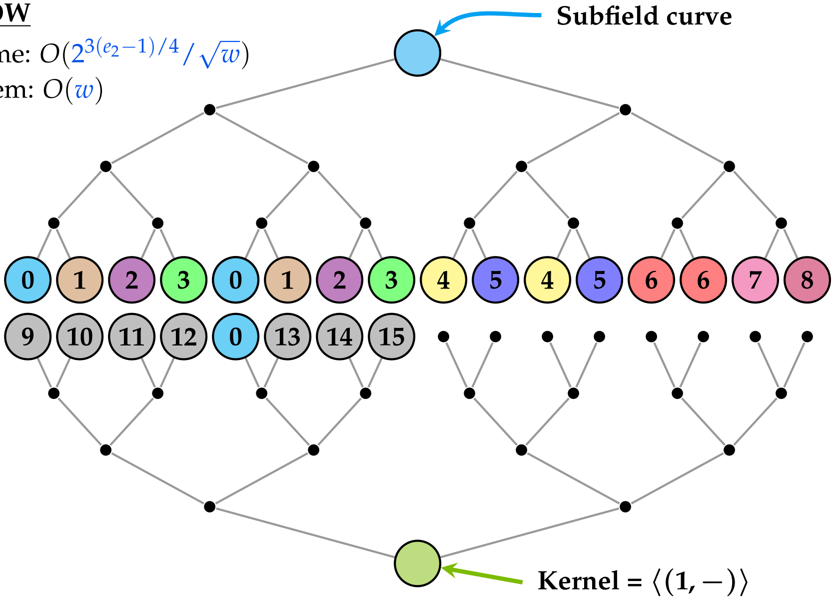


SIKE ($\ell = 2$)

vOW

Time: $O(2^{3(e_2-1)/4}/\sqrt{w})$

Mem: $O(w)$

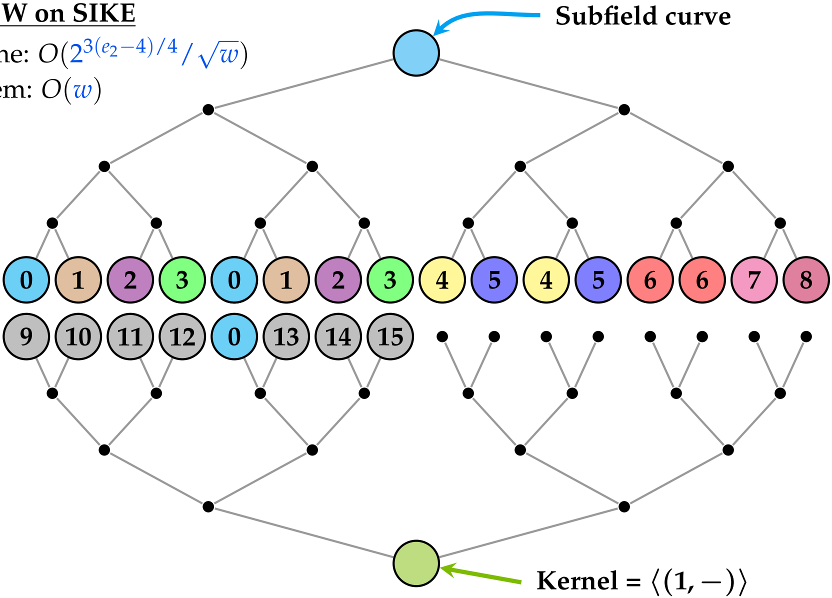


SIKE ($\ell = 2$)

vOW on SIKE

Time: $O(2^{3(e_2-4)/4}/\sqrt{w})$

Mem: $O(w)$



van Oorschot-Wiener

A set $S = \{0, \ldots, N-1\}$ and functions $h_0, h_1 : S \to T$

MitM problem

Find $x, y \in S$ such that $h_0(x) = h_1(y)$

[4] **vOW99**.

van Oorschot-Wiener

A set $S = \{0, \ldots, N-1\}$ and functions $h_0, h_1 : S \to T$

MitM problem

Find $x, y \in S$ such that $h_0(x) = h_1(y)$

- Define a family $\{f_n\}_{n \in \mathbb{N}}$ of functions

$$f_n : S^* \to S^*, \quad S^* = S \times \{0,1\}$$
$$(z, b) \mapsto g_n(h_b(z))$$

- The $\{g_n\}_{n \in \mathbb{N}}$ are "random" (e. g. SHA-3 domain sep. n)

[4] **vOW99**.

van Oorschot-Wiener

A set $S = \{0, \ldots, N-1\}$ and functions $h_0, h_1 : S \to T$

MitM problem

Find $x, y \in S$ such that $h_0(x) = h_1(y)$

- Define a family $\{f_n\}_{n \in \mathbb{N}}$ of functions

$$f_n : S^* \to S^*, \quad S^* = S \times \{0, 1\}$$
$$(z, b) \mapsto g_n(h_b(z))$$

- The $\{g_n\}_{n \in \mathbb{N}}$ are "random" (e. g. SHA-3 domain sep. n)
- For every n have a golden collision $f_n(x, 0) = f_n(y, 1)$

[4] **vOW99**.

van Oorschot-Wiener

A set $S = \{0, \ldots, N-1\}$ and functions $h_0, h_1 : S \to T$

MitM problem

Find $x, y \in S$ such that $h_0(x) = h_1(y)$

- Define a family $\{f_n\}_{n \in \mathbb{N}}$ of functions

$$f_n : S^* \to S^*, \quad S^* = S \times \{0, 1\}$$
$$(z, b) \mapsto g_n(h_b(z))$$

- The $\{g_n\}_{n \in \mathbb{N}}$ are "random" (e. g. SHA-3 domain sep. n)
- For every n have a golden collision $f_n(x, 0) = f_n(y, 1)$
- But for every n have many other collisions ($\approx N/2$)

[4] **vOW99**.

van Oorschot-Wiener

A set $S = \{0, \ldots, N-1\}$ and functions $h_0, h_1 : S \to T$

MitM problem

Find $x, y \in S$ such that $h_0(x) = h_1(y)$

- Define a family $\{f_n\}_{n \in \mathbb{N}}$ of functions

$$f_n : S^* \to S^*, \quad S^* = S \times \{0, 1\}$$
$$(z, b) \mapsto g_n(h_b(z))$$

- The $\{g_n\}_{n \in \mathbb{N}}$ are "random" (e. g. SHA-3 domain sep. n)
- For every n have a golden collision $f_n(x, 0) = f_n(y, 1)$
- But for every n have many other collisions ($\approx N/2$)

$\implies$ MitM reduces to golden collision search in any of the f_n

[4] **vOW99**.

van Oorschot-Wiener

Golden collision search

Given f_n (for fixed n) find the golden collision

van Oorschot-Wiener

Golden collision search

Given f_n (for fixed n) find the golden collision

The algorithm on a set S^* of size N and memory size w;

van Oorschot-Wiener

Golden collision search

Given f_n (for fixed n) find the golden collision

The algorithm on a set S^* of size N and memory size w;

1. Define distinguishedness property for $\theta = \sqrt{w/N} \in (0, 1]$

van Oorschot-Wiener

Golden collision search

Given f_n (for fixed n) find the golden collision

The algorithm on a set S^* of size N and memory size w;

1. Define distinguishedness property for $\theta = \sqrt{w/N} \in (0, 1]$
2. Randomly sample $z \in S^*$

van Oorschot-Wiener

Golden collision search

Given f_n (for fixed n) find the golden collision

The algorithm on a set S^* of size N and memory size w;

1. Define distinguishedness property for $\theta = \sqrt{w/N} \in (0, 1]$
2. Randomly sample $z \in S^*$
3. Compute $f_n(z), f_n(f_n(z)), \ldots$ until distinguished

van Oorschot-Wiener

Golden collision search

Given f_n (for fixed n) find the golden collision

The algorithm on a set S^* of size N and memory size w;

1. Define distinguishedness property for $\theta = \sqrt{w/N} \in (0, 1]$
2. Randomly sample $z \in S^*$
3. Compute $f_n(z), f_n(f_n(z)), \ldots$ until distinguished
4. If new, store in memory and goto 2

van Oorschot-Wiener

Golden collision search

Given f_n (for fixed n) find the golden collision

The algorithm on a set S^* of size N and memory size w;

1. Define distinguishedness property for $\theta = \sqrt{w/N} \in (0, 1]$
2. Randomly sample $z \in S^*$
3. Compute $f_n(z), f_n(f_n(z)), \ldots$ until distinguished
4. If new, store in memory and goto 2
5. If not new, check for golden collision. If not, store and goto 2

van Oorschot-Wiener

Golden collision search

Given f_n (for fixed n) find the golden collision

The algorithm on a set S^* of size N and memory size w;

1. Define distinguishedness property for $\theta = \sqrt{w/N} \in (0, 1]$
2. Randomly sample $z \in S^*$
3. Compute $f_n(z), f_n(f_n(z)), \ldots$ until distinguished
4. If new, store in memory and goto 2
5. If not new, check for golden collision. If not, store and goto 2
6. If found $10w$ distinguished points, try next n

van Oorschot-Wiener applied to SIDH/SIKE

The set $S = \{0, \ldots, \sqrt{2^{e_2}} - 1\}$ and (family of) random functions

$$f_n : z \in S^* \mapsto \text{AES-CBC}_n(j(E_i / \langle P_i + [z]Q_i \rangle))$$

van Oorschot-Wiener applied to SIDH/SIKE

The set $S = \{0, \ldots, \sqrt{2^{e_2}} - 1\}$ and (family of) random functions

$$f_n : z \in S^* \mapsto \text{AES-CBC}_n(j(E_i / \langle P_i + [z]Q_i \rangle))$$
$$=$$

van Oorschot-Wiener applied to SIDH/SIKE

The set $S = \{0, \ldots, \sqrt{2^{e_2}} - 1\}$ and (family of) random functions

$$\begin{aligned} f_n : z \in S^* &\mapsto \text{AES-CBC}_n(j(E_i / \langle P_i + [z]Q_i \rangle)) \\ &= \text{“Start from } E_i\text{,} \end{aligned}$$

van Oorschot-Wiener applied to SIDH/SIKE

The set $S = \{0, \dots, \sqrt{2^{e_2}} - 1\}$ and (family of) random functions

$$\begin{aligned} f_n : z \in S^* \mapsto & \text{ AES-CBC}_n(j(E_i / \langle P_i + [z]Q_i \rangle)) \\ = & \text{ “Start from } E_i \text{,} \\ & \text{ compute isogeny walk corresponding to } z \text{,} \end{aligned}$$

van Oorschot-Wiener applied to SIDH/SIKE

The set $S = \{0, \ldots, \sqrt{2^{e_2}} - 1\}$ and (family of) random functions

$$\begin{aligned} f_n : z \in S^* \mapsto\ & \text{AES-CBC}_n(j(E_i / \langle P_i + [z]Q_i \rangle)) \\ =\ & \text{“Start from } E_i\text{,} \\ & \text{compute isogeny walk corresponding to } z\text{,} \\ & \text{apply AES with key } n \text{ to } j\text{-invariant.”} \end{aligned}$$

van Oorschot-Wiener applied to SIDH/SIKE

The set $S = \{0, \ldots, \sqrt{2^{e_2}} - 1\}$ and (family of) random functions

$$\begin{aligned} f_n : z \in S^* \mapsto \; & \text{AES-CBC}_n(j(E_i / \langle P_i + [z]Q_i \rangle)) \\ = \; & \text{“Start from } E_i \text{,} \\ & \text{compute isogeny walk corresponding to } z \text{,} \\ & \text{apply AES with key } n \text{ to } j\text{-invariant.”} \end{aligned}$$

(Here E_0 is the *starting curve* and E_1 the *public key*.)

Application to SIDH / SIKE

e_2	$\log w$	$\log$ #**Queries** to f_n
		Exp. SIDH
32	9	23.20
36	10	25.70
40	11	28.20
44	13	30.20

[5] **canadians**.

Application to SIDH / SIKE

		log #**Queries** to f_n	
e_2	$\log w$	Exp.	**[canadians]**
		SIDH	SIDH
32	9	23.20	24.38
36	10	25.70	27.25
40	11	28.20	29.01
44	13	30.20	30.91

[5] **canadians**.

Application to SIDH / SIKE

e_2	$\log w$	log #**Queries** to f_n Exp. SIDH	**[canadians]** SIDH	**Ours** SIDH
32	9	23.20	24.38	23.29
36	10	25.70	27.25	25.74
40	11	28.20	29.01	28.33
44	13	30.20	30.91	30.37

[5] **canadians**.

Application to SIDH / SIKE

SIKE parameter choices + Equivalence classes

$\Longrightarrow$ N decreases by factor 6

$\Longrightarrow$ $N^{3/2}/w$ decreases by factor $\approx$ 15

		log #**Queries** to f_n		
e_2	$\log w$	Exp.	**[canadians]**	**Ours**
		SIDH	SIDH	SIDH
32	9	23.20	24.38	23.29
36	10	25.70	27.25	25.74
40	11	28.20	29.01	28.33
44	13	30.20	30.91	30.37

[5] **canadians**.

Application to SIDH / SIKE

SIKE parameter choices + Equivalence classes

$\implies N$ decreases by factor 6

$\implies N^{3/2}/w$ decreases by factor ≈ 15

e_2	$\log w$	log #**Queries** to f_n			
		Exp.		**[canadians]**	**Ours**
		SIDH	SIKE	SIDH	SIDH
32	9	23.20	19.32	24.38	23.29
36	10	25.70	21.82	27.25	25.74
40	11	28.20	24.32	29.01	28.33
44	13	30.20	26.32	30.91	30.37

[5] **canadians**.

Application to SIDH / SIKE

SIKE parameter choices + Equivalence classes

$\implies$ N decreases by factor 6

$\implies$ $N^{3/2}/w$ decreases by factor $\approx$ 15

		log #**Queries** to f_n				
		Exp.		**[canadians]**	**Ours**	
e_2	$\log w$	SIDH	SIKE	SIDH	SIDH	SIKE
32	9	23.20	19.32	24.38	23.29	19.58
36	10	25.70	21.82	27.25	25.74	21.89
40	11	28.20	24.32	29.01	28.33	24.40
44	13	30.20	26.32	30.91	30.37	26.42

[5]**canadians**.

Application to SIDH / SIKE

SIKE parameter choices + Equivalence classes

$\Longrightarrow$ N decreases by factor 6

$\Longrightarrow$ $N^{3/2}/w$ decreases by factor $\approx$ 15

e_2	$\log w$	log #**Queries** to f_n				
		Exp.		**[canadians]**	**Ours**	
		SIDH	SIKE	SIDH	SIDH	SIKE
32	9	23.20	19.32	24.38	23.29	19.58
36	10	25.70	21.82	27.25	25.74	21.89
40	11	28.20	24.32	29.01	28.33	24.40
44	13	30.20	26.32	30.91	30.37	26.42
56	17	37.20	33.32	–	–	33.38

[5] **canadians**.

From theory to practice

Appearing soon: `https://github.com/microsoft/vOW4SIKE`

- C library building on SIKE submission

$\Longrightarrow$ Fast arithmetic

$\Longrightarrow$ Speed-up oracle queries f_n

From theory to practice

Appearing soon: `https://github.com/microsoft/vOW4SIKE`

- C library building on SIKE submission
  - $\Longrightarrow$ Fast arithmetic
  - $\Longrightarrow$ Speed-up oracle queries f_n
- Naïve setup: single instance with access to memory of size w
  - $\Longrightarrow$ No significant overhead

From theory to practice

Appearing soon: `https://github.com/microsoft/vOW4SIKE`

- C library building on SIKE submission
  - $\Longrightarrow$ Fast arithmetic
  - $\Longrightarrow$ Speed-up oracle queries f_n
- Naïve setup: single instance with access to memory of size w
  - $\Longrightarrow$ No significant overhead
- Real setup: m instances with shared memory of size w
  - $\Longrightarrow$ Complexity $O(2^{3(e_2-4)/4}/(m\sqrt{w}))$

From theory to practice

Appearing soon: `https://github.com/microsoft/vOW4SIKE`

- C library building on SIKE submission
  - $\Longrightarrow$ Fast arithmetic
  - $\Longrightarrow$ Speed-up oracle queries f_n
- Naïve setup: single instance with access to memory of size w
  - $\Longrightarrow$ No significant overhead
- Real setup: m instances with shared memory of size w
  - $\Longrightarrow$ Complexity $O(2^{3(e_2-4)/4}/(m\sqrt{w}))$
  - $\Longrightarrow$ Number of memory accesses becomes bottleneck

From theory to practice

Appearing soon: `https://github.com/microsoft/vOW4SIKE`

- C library building on SIKE submission
  - $\Longrightarrow$ Fast arithmetic
  - $\Longrightarrow$ Speed-up oracle queries f_n
- Naïve setup: single instance with access to memory of size w
  - $\Longrightarrow$ No significant overhead
- Real setup: m instances with shared memory of size w
  - $\Longrightarrow$ Complexity $O(2^{3(e_2-4)/4}/(m\sqrt{w}))$
  - $\Longrightarrow$ Number of memory accesses becomes bottleneck
  - $\Longrightarrow$ Have to synchronize n across instances

From theory to practice

Appearing soon: `https://github.com/microsoft/vOW4SIKE`

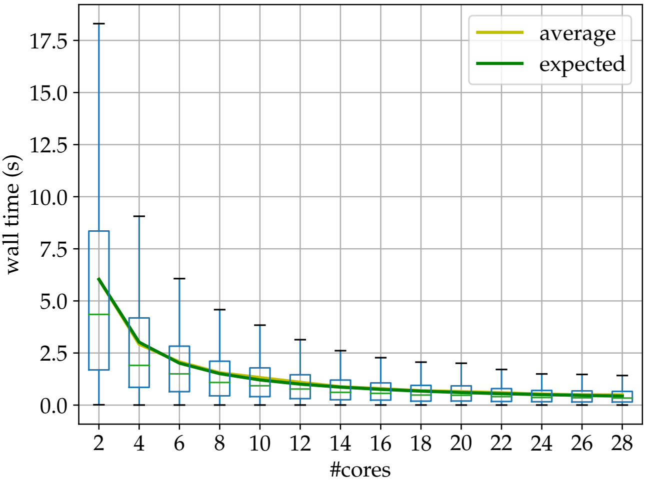
- C library building on SIKE submission
  - $\Longrightarrow$ Fast arithmetic
  - $\Longrightarrow$ Speed-up oracle queries f_n
- Naïve setup: single instance with access to memory of size w
  - $\Longrightarrow$ No significant overhead
- Real setup: m instances with shared memory of size w
  - $\Longrightarrow$ Complexity $O(2^{3(e_2-4)/4}/(m\sqrt{w}))$
  - $\Longrightarrow$ Number of memory accesses becomes bottleneck
  - $\Longrightarrow$ Have to synchronize n across instances
  - $\Longrightarrow$ We built only *multi-core*, steps towards true *distributed*

From theory to practice

Appearing soon: `https://github.com/microsoft/vOW4SIKE`

- C library building on SIKE submission
  - $\Longrightarrow$ Fast arithmetic
  - $\Longrightarrow$ Speed-up oracle queries f_n
- Naïve setup: single instance with access to memory of size w
  - $\Longrightarrow$ No significant overhead
- Real setup: m instances with shared memory of size w
  - $\Longrightarrow$ Complexity $O(2^{3(e_2-4)/4}/(m\sqrt{w}))$
  - $\Longrightarrow$ Number of memory accesses becomes bottleneck
  - $\Longrightarrow$ Have to synchronize n across instances
  - $\Longrightarrow$ We built only *multi-core*, steps towards true *distributed*
  - $\Longrightarrow$ Instances have (small) *local memory*, use this

Parallelized van Oorschot-Wiener



[6]Experiments run on 2x Intel(R) Xeon(R) E5-2690 v4 at 2.60 GHz with 14 cores each

Local memory for checking collisions

Collision checking

$(z, \bar{z}, d)$ z ● ● $\bar{z}$

$(y, \bar{z}, e)$ y ● ● $\bar{z}$

Local memory for checking collisions

Collision checking

$(z, \bar{z}, d)$

$(y, \bar{z}, e)$

Local memory for checking collisions

Collision checking

$(z, \bar{z}, d)$

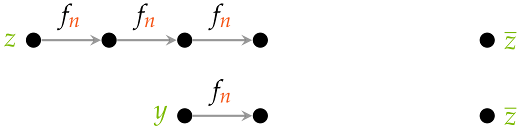
$(y, \bar{z}, e)$

Local memory for checking collisions

Collision checking

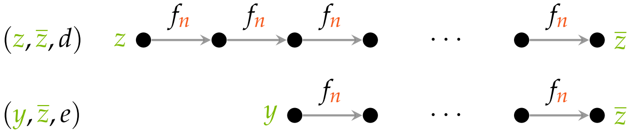
$(z, \bar{z}, d)$

$(y, \bar{z}, e)$



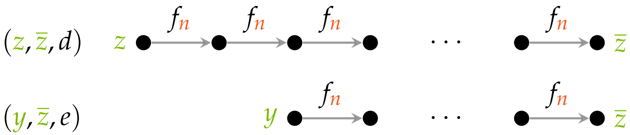
Local memory for checking collisions

Collision checking

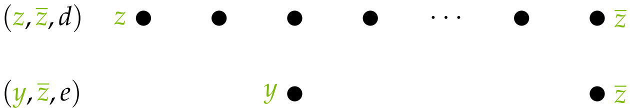


Local memory for checking collisions

Collision checking

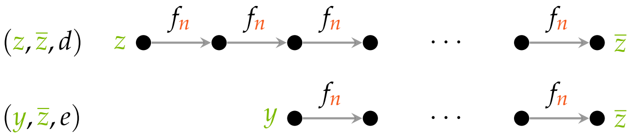


Additional assumption: can store *all* intermediate points locally

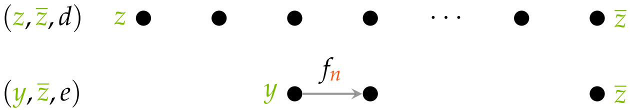


Local memory for checking collisions

Collision checking

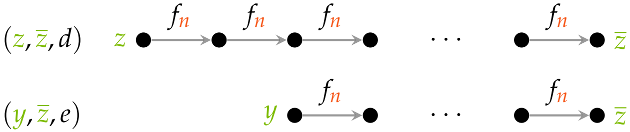


Additional assumption: can store *all* intermediate points locally

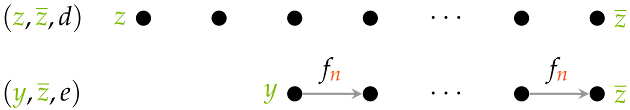


Local memory for checking collisions

Collision checking

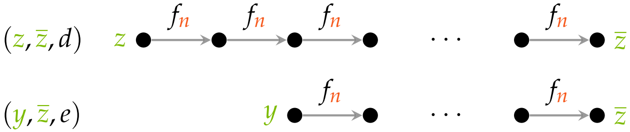


Additional assumption: can store *all* intermediate points locally

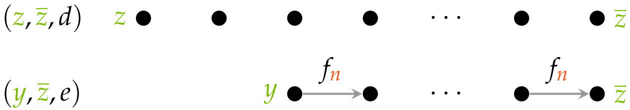


Local memory for checking collisions

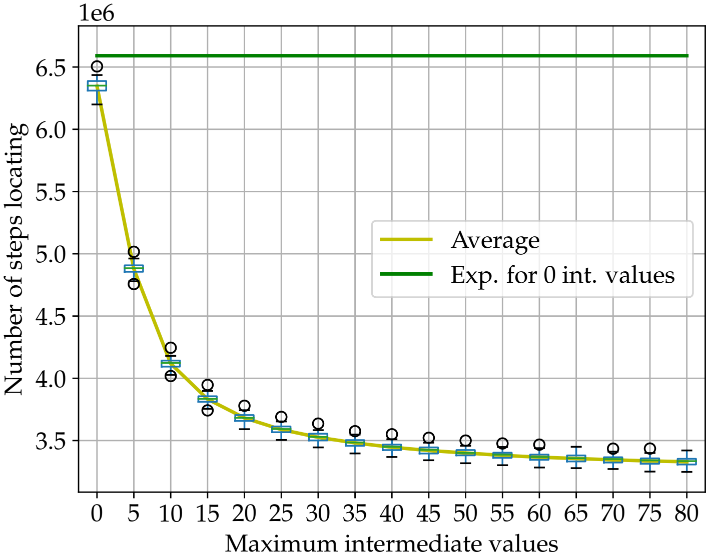
Collision checking



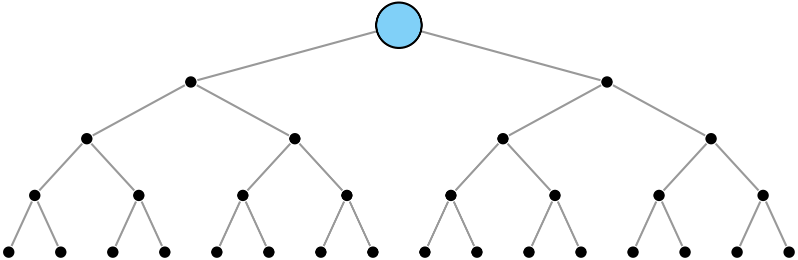
Additional assumption: can store t intermediate points locally



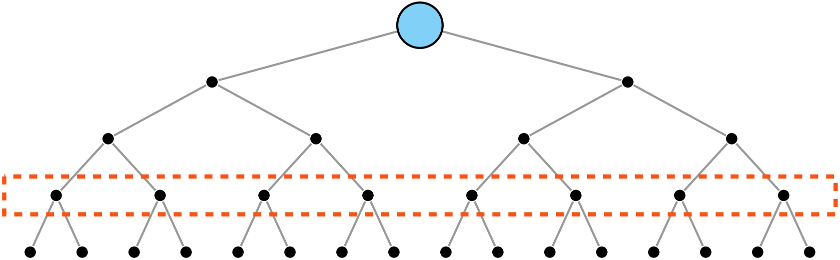
Local memory for checking collisions



Local memory for precomputation

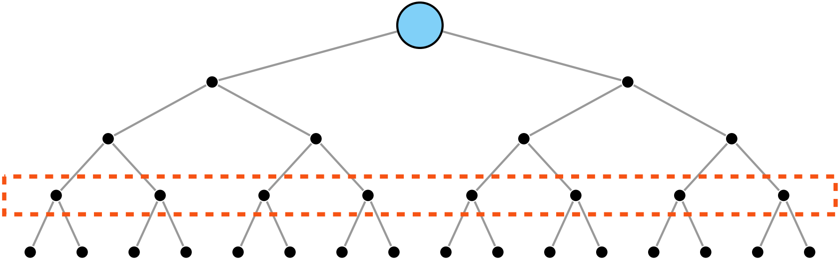


Local memory for precomputation



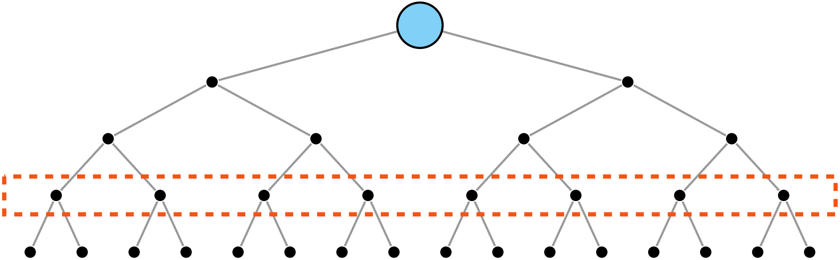
- Fix precomputation depth Δ (could vary per device)

Local memory for precomputation



- Fix precomputation depth Δ (could vary per device)
- Isogenies of degree $2^{2(e_2-1)-\Delta}$
- Table size of $2 \cdot 3 \cdot 2^{\Delta}$ elements in $\mathbb{F}_{p^2}$

Local memory for precomputation



- Fix precomputation depth Δ (could vary per device)
- Isogenies of degree $2^{2(e_2-1)-\Delta}$
- Table size of $2 \cdot 3 \cdot 2^{\Delta}$ elements in $\mathbb{F}_{p^2}$
- Make sure to store the right basis of $2^{e_2-\Delta}$-torsion..

Other optimizations

- Multi-target attacks
- Compressing distinguished points — “leading bits are zero”

Thanks!

$$\begin{array}{ccc} E_0 & \dashrightarrow & E_A \\ \downarrow & & \downarrow \\ E_B & \dashrightarrow & E_{AB} \end{array}$$