Faster Isogeny-Based Compressed Key Agreement

Gustavo H. M. Zanon, Marcos A. Simplicio Jr, Geovandro C. C. F. Pereira, Javad Doliskani, and Paulo S. L. M. Barreto.

REVIEW: SIDH AND COMPRESSED KEYS

Isogeny-based Crypto

- SIDH: proposed replacement for DH-based elliptic curves in a post-quantum world.
- Smallest post-quantum public keys (< 200 bytes)
  - boosted by key compression techniques
  - applications with low bandwidth requirements
- Downside:
  - ≈2 order of magnitude slower than Four$\mathbb{Q}$-based DH or other fast post-quantum KEM schemes (NewHope/NTRU).

SIDH Parameter Setting

- $p = 2^m \cdot 3^n - 1$ for post-quantum sec. level ≈ 128 bits
  - Previous: 751-bit prime for $m = 372, n = 239$
  - [2018] Adj *et al.* suggest ≈ 448-bit primes are enough

SIDH Parameter Setting

- $p = 2^m \cdot 3^n - 1$ for post-quantum sec. level ≈ 128 bits
  - Previous: 751-bit prime for $m = 372, n = 239$
  - [2018] Adj *et al.* suggest ≈ 448-bit primes are enough
- $E_0/\mathbb{F}_{p^2} : By^2 = x^3 + Ax^2 + x$ a supersingular Montgomery curve of order $(\mathrm{p}+1)^2 = 2^{2m}3^{2n}$
  - $\langle P_A, Q_A \rangle = E(\mathbb{F}_{p^2})[2^m]$, $\langle P_B, Q_B \rangle = E(\mathbb{F}_{p^2})[3^n]$

SIDH Parameter Setting

- $p = 2^m \cdot 3^n - 1$ for post-quantum sec. level ≈ 128 bits
  - Previous: 751-bit prime for $m = 372, n = 239$
  - [2018] Adj *et al.* suggest ≈ 448-bit primes are enough
- $E_0/\mathbb{F}_{p^2} : By^2 = x^3 + Ax^2 + x$ a supersingular Montgomery curve of order $(\mathrm{p}+1)^2 = 2^{2m}3^{2n}$
  - $\langle P_A, Q_A \rangle = E(\mathbb{F}_{p^2})[2^m]$, $\langle P_B, Q_B \rangle = E(\mathbb{F}_{p^2})[3^n]$
- User private key: $s \in_R \mathbb{Z}/\ell^e\mathbb{Z}$ for $\ell \in \{2,3\}, e \in \{m, n\}$
- User public key: curve $\boldsymbol{E_{A,B}} = \phi(E_0)$ and points $\boldsymbol{\phi}(\boldsymbol{P})$, $\boldsymbol{\phi}(\boldsymbol{Q}) \in E_{A,B}$.

SIDH Public Key Compression

- Goal: transmit public key $\{E_{A,B}, \phi(P), \phi(Q)\}$

$E_{A,B}/\mathbb{F}_{p^2}: By^2 = x^3 + Ax^2 + x$

$\phi(P), \phi(Q) \in \mathrm{E_{A,B}}$

SIDH Public Key Compression

- [2011] Jao *et al.*'s public key representation:

Alice

$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{x}_{\boldsymbol{\phi}(\boldsymbol{P})}, \boldsymbol{x}_{\boldsymbol{\phi}(\boldsymbol{Q})} \in \mathbb{F}_{p^2}$

Pub. Key size: $\mathbf{8 \log} \boldsymbol{p}$ bits

Bob

$E_{A,B}/\mathbb{F}_{p^2}: \boldsymbol{B}y^2 = x^3 + \boldsymbol{A}x^2 + x$

$\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{\phi}(\boldsymbol{Q}) \in \mathrm{E}_{\mathrm{A,B}}$

SIDH Public Key Compression

- [2016] Azarderakhsh *et al.'s* key compression:

$j(\boldsymbol{E_{A,B}})$

$\boldsymbol{E_{A,B}}/\mathbb{F}_{p^2}: By^2 = x^3 + Ax^2 + x$

$\phi(P), \phi(Q) \in \mathrm{E_{A,B}}$

$E_{A',B'} \leftarrow \boldsymbol{j(E_{A,B})}$

isomorphic curve

SIDH Public Key Compression

- [2016] Azarderakhsh *et al.'s* key compression:

Alice

Bob

$\boldsymbol{j(E_{A,B})} \in \mathbb{F}_{p^2}$: $\mathbf{2\log p}$ bits

vs

$\boldsymbol{A, B} \in \mathbb{F}_{p^2}$: $\mathbf{4\log p}$ bits

$\boldsymbol{E_{A,B}}/\mathbb{F}_{p^2}: By^2 = x^3 + Ax^2 + x$

$\phi(P), \phi(Q) \in \mathrm{E_{A,B}}$

$\mathbf{2\log p}$ bits saved

SIDH Public Key Compression

- [2016] Azarderakhsh *et al.’s* key compression:

$j(\boldsymbol{E}_{\boldsymbol{A},\boldsymbol{B}})$

$\boldsymbol{E}_{\boldsymbol{A},\boldsymbol{B}}/\mathbb{F}_{p^2}: By^2 = x^3 + Ax^2 + x$

$\phi(P), \phi(Q) \in E_{A,B}$

There is a canonical basis $\{R_1, R_2\}$ such that

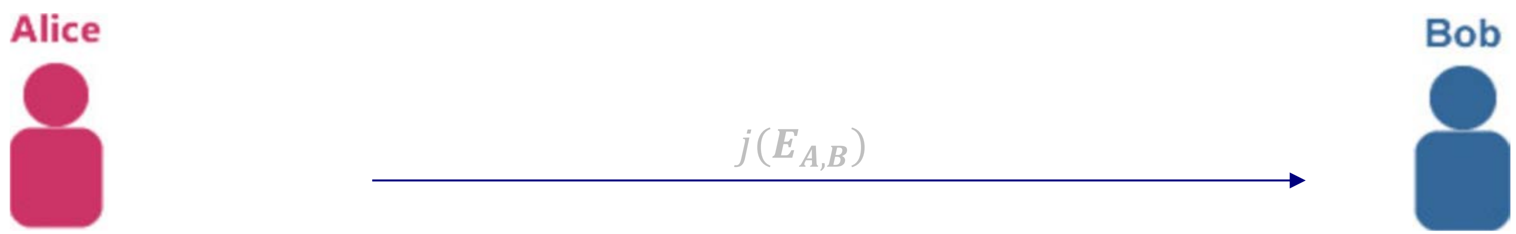
$$\langle R_1, R_2 \rangle = E_{A,B}[3^n]$$

Idea: express

$$\left.\begin{aligned} \phi(P) &= a_1 R_1 + a_2 R_2 \\ \phi(Q) &= b_1 R_1 + b_2 R_2 \end{aligned}\right\}$$

SIDH Public Key Compression

- [2016] Azarderakhsh *et al.'s* key compression:



$E_{A,B}/\mathbb{F}_{p^2}: By^2 = x^3 + Ax^2 + x$

$\phi(P), \phi(Q) \in E_{A,B}$

There is a canonical basis $\{R_1, R_2\}$ such that

$$\langle R_1, R_2 \rangle = E_{A,B}[3^n]$$

Linear algebra tasks

- Build a *basis*

Idea: express

$$\phi(P) = a_1 R_1 + a_2 R_2$$
$$\phi(Q) = b_1 R_1 + b_2 R_2$$

- Internal product: *pairing*
- Coeff. extraction: *DLOG*

SIDH Public Key Compression

- [2016] Azarderakhsh *et al.'s* key compression:

Alice $\xrightarrow{j(\boldsymbol{E_{A,B}})}$ **Bob**

$\boldsymbol{E_{A,B}}/F_{p^2}: By^2 = x^3 + Ax^2 + x$

$\phi(P), \phi(Q) \in E_{A,B}$

Compression (1/3):

- find a basis $\{R_1, R_2\}$

$$\phi(P) = a_1R_1 + a_2R_2$$

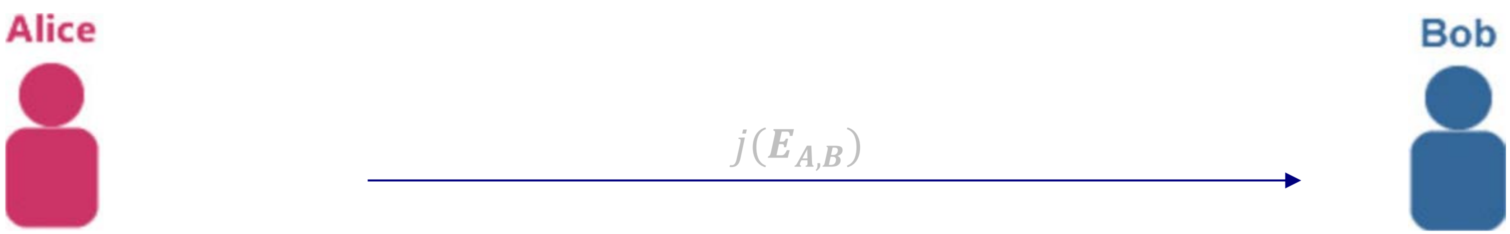
$$\phi(Q) = b_1R_1 + b_2R_2$$

Find R_1, R_2:

Expensive scalar multiplications involved

SIDH Public Key Compression

- [2016] Azarderakhsh *et al.'s* key compression:



$j(\boldsymbol{E_{A,B}})$ (Alice → Bob)

$\boldsymbol{E_{A,B}}/F_{p^2}: By^2 = x^3 + Ax^2 + x$

$\phi(P), \phi(Q) \in E_{A,B}$

Compression (2/3):

- prepare DLOG instances
- Cost: 5 pairings

$$\phi(P) = a_1R_1 + a_2R_2$$
$$\phi(Q) = b_1R_1 + b_2R_2$$
$$\boldsymbol{g = e_{3^n}(R_1, R_2)}$$
$$\boldsymbol{g_0 = e_{3^n}(R_1, \phi(P))}$$
$$\boldsymbol{g_1 = e_{3^n}(R_2, \phi(P))}$$
$$\boldsymbol{g_2 = e_{3^n}(R_1, \phi(Q))}$$
$$\boldsymbol{g_3 = e_{3^n}(R_2, \phi(Q))}$$

SIDH Public Key Compression

- [2016] Azarderakhsh *et al.'s* key compression:

Alice

Bob

$j(\boldsymbol{E_{A,B}})$

$\boldsymbol{E_{A,B}}/F_{p^2}: By^2 = x^3 + Ax^2 + x$

$\phi(P), \phi(Q) \in E_{A,B}$

Compression (3/3):

- Compute a_i's and b_i's
- Cost: 4 order 3^n DLOGs (Pohlig-Hellman)

$$\begin{cases} \phi(P) = a_1 R_1 + a_2 R_2 \\ \phi(Q) = b_1 R_1 + b_2 R_2 \\ \boldsymbol{a_1 = -\log_g g_1} \\ \boldsymbol{a_2 = \log_g g_0} \\ \boldsymbol{b_1 = -\log_g g_3} \\ \boldsymbol{b_2 = \log_g g_2} \end{cases}$$

SIDH Public Key Compression

- [2016] Azarderakhsh *et al.'s* key compression:

Alice → **Bob**

$j(\boldsymbol{E_{A,B}})$

$\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{b_1}, \boldsymbol{b_2} \in \mathbb{Z}_{3^n}$

$\boldsymbol{E_{A,B}}/F_{p^2}: By^2 = x^3 + Ax^2 + x$

$\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{\phi}(\boldsymbol{Q}) \in E_{A,B}$

SIDH Public Key Compression

- [2016] Azarderakhsh *et al.'s* key compression:

Alice

$E_{A,B}/F_{p^2}: By^2 = x^3 + Ax^2 + x$

$\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{\phi}(\boldsymbol{Q}) \in E_{A,B}$

Bob

$j(E_{A,B})$

$\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{b_1}, \boldsymbol{b_2} \in (\mathbb{Z}_{3^n})^4$: $\mathbf{2 \log} \boldsymbol{p}$ bits

Vs

$x_{\phi(P)}, x_{\phi(Q)} \in \mathbb{F}_{p^2}$: $\mathbf{4 \log} \boldsymbol{p}$ bits

$\boldsymbol{2\, log\, p}$ bits saved

SIDH Public Key Compression

- [2016] Azarderakhsh *et al.*'s key compression:

Alice

$j(\boldsymbol{E_{A,B}})$

$\boldsymbol{a_1, a_2, b_1, b_2}$

Bob

$\boldsymbol{E_{A,B}}/F_{p^2}: By^2 = x^3 + Ax^2 + x$

$\phi(P), \phi(Q) \in E_{A,B}$

Decompression

- Compute $\langle R_1, R_2 \rangle = E_{A',B'}[3^n]$
- Recover points:

$$\boldsymbol{\phi(P)} \leftarrow \boldsymbol{a_1} R_1 + \boldsymbol{a_2} R_2$$

$$\boldsymbol{\phi(Q)} \leftarrow \boldsymbol{b_1} R_1 + \boldsymbol{b_2} R_2$$

- Cost: 4 scalar muls.

SIDH Public Key Compression

- [2016] Azarderakhsh *et al.'s* key compression:

$j(E_{A,B}) \in \mathbb{F}_{p^2}$: $2\log p$ bits

$a_1, a_2, b_1, b_2 \in \mathbb{Z}_{3^n}$: $2\log p$ bits

vs

$A, B \in F_{p^2}$: $4\log p$ bits

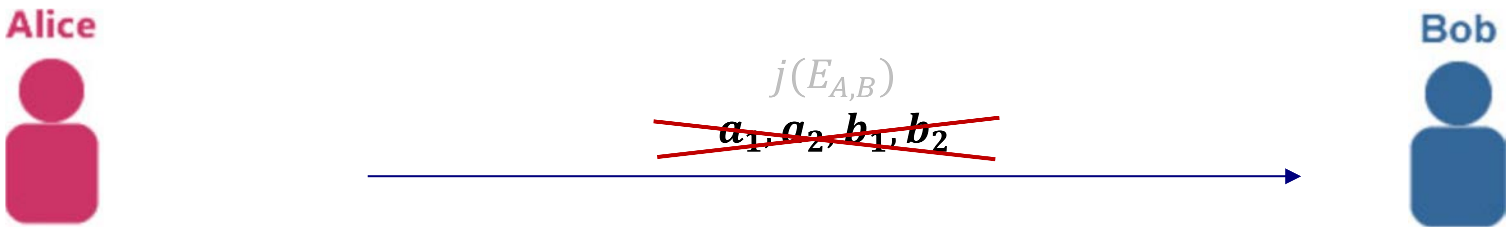
$x(\phi(P)), x(\phi(Q))$: $4\log p$ bits

Public key size: $4\, log\, p$ bits

- Keys shrunk by 2× ☺
- Compression time > 10× KEX ☹

SIDH Public Key Compression

- [2017] Costello *et al.* key compression:



$\boldsymbol{E}/F_{p^2}: By^2 = x^3 + \boldsymbol{A}x^2 + x$

$\boldsymbol{\phi(P)}, \boldsymbol{\phi(Q)} \in E$

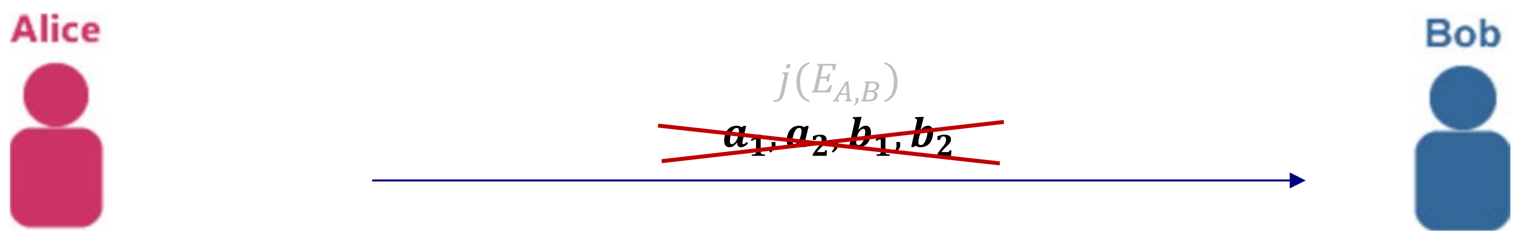
Further compression

- Bob recovers $\phi(P), \phi(Q)$ to compute the kernel

$$K = \langle \phi(P) + s_B \phi(Q) \rangle$$

SIDH Public Key Compression

- [2017] Costello *et al.* key compression:



$E/F_{p^2}: By^2 = x^3 + Ax^2 + x$

$\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{\phi}(\boldsymbol{Q}) \in E$

Further compression

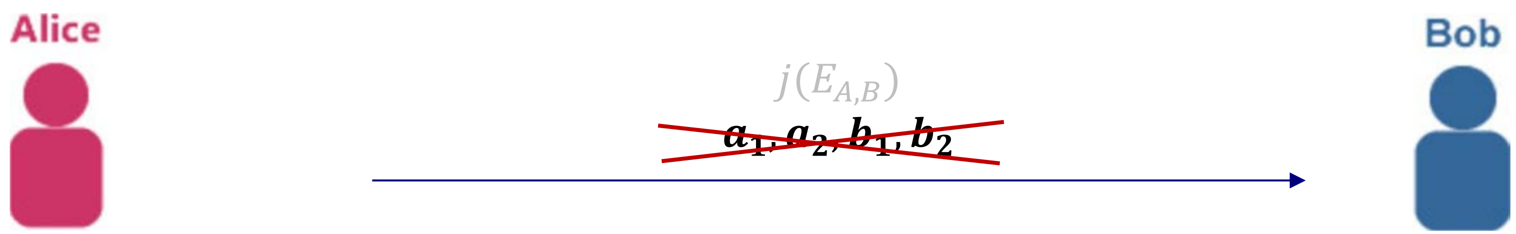
- After recovering $\phi(P), \phi(Q)$, Bob computes the kernel

$$K = \langle \phi(P) + s_B \phi(Q) \rangle$$

$$= \langle (a_1 + s_B b_1) R_1 + (a_2 + s_B b_1) R_2 \rangle$$

SIDH Public Key Compression

- [2017] Costello *et al.* key compression:



$\boldsymbol{E}/F_{p^2}: By^2 = x^3 + \boldsymbol{A}x^2 + x$

$\boldsymbol{\phi(P), \phi(Q)} \in E$

Further compression

- After recovering $\phi(P), \phi(Q)$, Bob computes the kernel

$$K = \langle \phi(P) + s_B\phi(Q) \rangle$$

$$= \langle (a_1 + s_B b_1)R_1 + (a_2 + s_B b_1)R_2 \rangle$$

- wlog. assume a_1 is invertible $mod\ 3^n$ (otherwise b_1 is), then

$$a_1^{-1}K = \langle (1 + \boldsymbol{s_B} b_1 a_1^{-1})R_1 + (a_2 a_1^{-1} + \boldsymbol{s_B} b_2 a_1^{-1})R_2 \rangle = K$$

SIDH Public Key Compression

- [2017] Costello *et al.'s* key compression:

Alice

$\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} \in (\mathbb{Z}_{3^n})^3$: $\mathbf{3/2 \log} \boldsymbol{p}$ bits

Bob

$\boldsymbol{E}/F_{p^2}: By^2 = x^3 + \boldsymbol{A}x^2 + x$

$\phi(P), \phi(Q) \in E$

3 elements in $\mathbb{Z}_{3^n}$ are enough:

$$\alpha = b_1 a_1^{-1} \in \mathbb{Z}_{3^n}$$

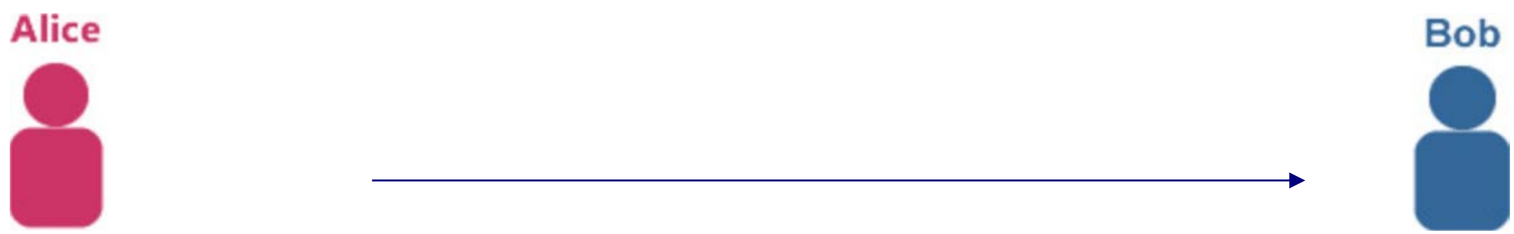
$$\beta = a_2 a_1^{-1} \in \mathbb{Z}_{3^n}$$

$$\gamma = b_2 a_1^{-1} \in \mathbb{Z}_{3^n}$$

Plus 1 bit about invertibility of a_1 or b_1

SIDH Public Key Compression

- 2017, Costello *et al.'s* key compression:



$E/F_{p^2}: By^2 = x^3 + Ax^2 + x$

$\phi(P), \phi(Q) \in E$

To compress $\phi(P), \phi(Q)$:

- generate basis $\{R_1, R_2\}$
- compute 5 pairings
  - NB: cost of 5-way Monty Inv.: 30 muls (report)
- compute 4 DLOGs, i.e., $\{a_1, a_2, b_1, b_2\}$
- compute α, β, γ from the quadruple above

Optimizations on steps 1, 2 and 3 of compression and on decompression.

SIDH Public Key Compression

- 2017, Costello *et al.'s* key compression:

$j(E) \in \mathbb{F}_{p^2}$: $\mathbf{2\log p}$ bits

$\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} \in (\mathbb{Z}_{3^n})^3$: $\mathbf{3/2\log p}$ bits

$\boldsymbol{E}/F_{p^2}: By^2 = x^3 + \boldsymbol{A}x^2 + x$

$\phi(P), \phi(Q) \in E$

Public key size: $\mathbf{3.5}\,\boldsymbol{log\,p}$ bits

- Ex.: $|pk| = 328$ bytes for $|p| = 751$ bits

Compression time $\approx \mathbf{1}\times$ KEX and decompression $\approx \mathbf{0.4}\times$ KEX

SIDH Public Key Compression

- Is the current (de)compression performance acceptable?

SIDH Public Key Compression

- Is the current (de)compression performance acceptable?
- Current state of classical elliptic curves:
  - CHES'2017*: speed records for ECDH on embedded devices using curve Fourℚ.
    - Compression = free (similar to original SIDH, send one coordinate of the point)
    - Decompression = 0.04x key agreement

*Liu Z, Longa P, Pereira G, Reparaz O, Seo H. FourQ on embedded devices with strong countermeasures against side-channel attacks.

SIDH Public Key Compression

- Is the current (de)compression performance acceptable?
- Current state of classical elliptic curves:
  - CHES’2017*: speed records for ECDH on embedded devices using curve Fourℚ.
    - Compression = free (similar to original SIDH, send one coordinate of the point)
    - Decompression = 0.04x key agreement
- This work’s goal is reduce this gap
  - Detect and improve the remaining SIDH key compression bottlenecks.

*Liu Z, Longa P, Pereira G, Reparaz O, Seo H. FourQ on embedded devices with strong countermeasures against side-channel attacks.

Faster SIDH Public Key Compression

- Most costly operations:
    I. Computing a basis $\{R_1, R_2\}$
    II. Computing 5 pairings
    III. Computing 4 discrete logs

Faster SIDH Public Key Compression

- Most costly operations:
  I. Computing a basis $\{R_1, R_2\}$
  II. Computing 5 pairings
  III. Computing 4 discrete logs
- New algorithms to address the above bottlenecks.

Faster SIDH Public Key Compression

- Most costly operations:
  I. Computing a basis $\{R_1, R_2\}$
  II. Computing 5 pairings
  III. Computing 4 discrete logs
- New algorithms to address the above bottlenecks.
  - Reverse basis decomposition
    - Pairings reduced to 4 instead of 5 for both sides.
    - 2 multiplications by large cofactor 3^n saved in the binary case.
    - Allows for faster discrete logs.: precompute (single, shared) table offline.

Reverse basis decomposition

- Previous works express the public key as

$$\phi(P) = a_1 R_1 + a_2 R_2$$

$$\phi(Q) = b_1 R_1 + b_2 R_2$$

- or in matrix notation

$$\begin{bmatrix} \phi(P) \\ \phi(Q) \end{bmatrix} = \overbrace{\begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix}}^{M_{2x2}} \begin{bmatrix} R_1 \\ R_2 \end{bmatrix}$$

Reverse basis decomposition

- Previous works express the public key as

$$\phi(P) = a_1 R_1 + a_2 R_2$$

$$\phi(Q) = b_1 R_1 + b_2 R_2$$

- or in matrix notation

$$\begin{bmatrix} \phi(P) \\ \phi(Q) \end{bmatrix} = \overbrace{\begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix}}^{M_{2x2}} \begin{bmatrix} R_1 \\ R_2 \end{bmatrix}$$

- Since $\{\phi(P), \phi(Q)\}$ also form a basis, matrix M is invertible and changing roles:

$$\begin{bmatrix} R_1 \\ R_2 \end{bmatrix} = \overbrace{\begin{bmatrix} c_1 & c_2 \\ d_1 & d_2 \end{bmatrix}}^{M^{-1}} \begin{bmatrix} \phi(P) \\ \phi(Q) \end{bmatrix}$$

- **Idea**: revert the process by starting from M^{-1} and recovering M from it?

Reverse basis decomposition

- Express $\{R_1, R_2\}$ in basis $\{\phi(P), \phi(Q)\}$

$$R_1 = c_1\phi(P) + c_2\phi(Q)$$
$$R_2 = d_1\phi(P) + d_2\phi(Q)$$

Reverse basis decomposition

- Express $\{R_1, R_2\}$ in basis $\{\phi(P), \phi(Q)\}$

$$R_1 = c_1\phi(P) + c_2\phi(Q)$$
$$R_2 = d_1\phi(P) + d_2\phi(Q)$$

$$e(\phi(P), R_1) =$$

Reverse basis decomposition

- Express $\{R_1, R_2\}$ in basis $\{\phi(P), \phi(Q)\}$

$$R_1 = c_1\phi(P) + c_2\phi(Q)$$
$$R_2 = d_1\phi(P) + d_2\phi(Q)$$

$$e(\phi(P), R_1) = e(\phi(P), c_1\phi(P) + c_2\phi(Q))$$
$$= e(\phi(P), c_1\phi(P)) \cdot e(\phi(P), c_2\phi(Q))$$
$$= e(\phi(P), \phi(P))^{c_1} \cdot e(\phi(P), \phi(Q))^{c_2}$$
$$= \boldsymbol{e(\phi(P), \phi(Q))^{c_2}}$$

Reverse basis decomposition

- Express $\{R_1, R_2\}$ in basis $\{\phi(P), \phi(Q)\}$

$$R_1 = c_1\phi(P) + c_2\phi(Q)$$
$$R_2 = d_1\phi(P) + d_2\phi(Q)$$

$$e(\phi(P), R_1) = e(\phi(P), c_1\phi(P) + c_2\phi(Q))$$
$$= e(\phi(P), c_1\phi(P)) \cdot e(\phi(P), c_2\phi(Q))$$
$$= e(\phi(P), \phi(P))^{c_1} \cdot e(\phi(P), \phi(Q))^{c_2}$$
$$= \boldsymbol{e(\phi(P), \phi(Q))^{c_2}}$$

$\boldsymbol{h}$

$$h = \boldsymbol{e(\phi(P), \phi(Q))}$$
$$= \boldsymbol{e\left(P, \widehat{\phi} \circ \phi(Q)\right)}$$

Reverse basis decomposition

- Express $\{R_1, R_2\}$ in basis $\{\phi(P), \phi(Q)\}$

$$R_1 = c_1\phi(P) + c_2\phi(Q)$$
$$R_2 = d_1\phi(P) + d_2\phi(Q)$$

$$e(\phi(P), R_1) = e(\phi(P), c_1\phi(P) + c_2\phi(Q))$$
$$= e(\phi(P), c_1\phi(P)) \cdot e(\phi(P), c_2\phi(Q))$$
$$= e(\phi(P), \phi(P))^{c_1} \cdot e(\phi(P), \phi(Q))^{c_2}$$
$$= \boldsymbol{e(\phi(P), \phi(Q))}^{c_2}$$

$\boldsymbol{h}$

$$h = \boldsymbol{e(\phi(P), \phi(Q))}$$
$$= \boldsymbol{e\left(P, \widehat{\phi} \circ \phi(Q)\right)}$$
$$= \boldsymbol{e(P, [deg\ \phi]Q)}$$

Reverse basis decomposition

- Express $\{R_1, R_2\}$ in basis $\{\phi(P), \phi(Q)\}$

$$R_1 = c_1\phi(P) + c_2\phi(Q)$$
$$R_2 = d_1\phi(P) + d_2\phi(Q)$$

$$e(\phi(P), R_1) = e(\phi(P), c_1\phi(P) + c_2\phi(Q))$$
$$= e(\phi(P), c_1\phi(P)) \cdot e(\phi(P), c_2\phi(Q))$$
$$= e(\phi(P), \phi(P))^{c_1} \cdot e(\phi(P), \phi(Q))^{c_2}$$
$$= \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{\phi}(\boldsymbol{Q}))^{c_2}$$

$\boldsymbol{h}$

$$h = \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{\phi}(\boldsymbol{Q}))$$
$$= \boldsymbol{e}\left(\boldsymbol{P}, \widehat{\boldsymbol{\phi}} \circ \boldsymbol{\phi}(\boldsymbol{Q})\right)$$
$$= \boldsymbol{e}(\boldsymbol{P}, [\boldsymbol{deg}\ \boldsymbol{\phi}]\boldsymbol{Q})$$
$$= \boldsymbol{e}(\boldsymbol{P}, \boldsymbol{Q})^{\boldsymbol{deg}\ \boldsymbol{\phi}}$$

Reverse basis decomposition

- Express $\{R_1, R_2\}$ in basis $\{\phi(P), \phi(Q)\}$

$$R_1 = c_1\phi(P) + c_2\phi(Q)$$
$$R_2 = d_1\phi(P) + d_2\phi(Q)$$

$$e(\phi(P), R_1) = e(\phi(P), c_1\phi(P) + c_2\phi(Q))$$
$$= e(\phi(P), c_1\phi(P)) \cdot e(\phi(P), c_2\phi(Q))$$
$$= e(\phi(P), \phi(P))^{c_1} \cdot e(\phi(P), \phi(Q))^{c_2}$$
$$= \boldsymbol{e(\phi(P), \phi(Q))^{c_2}}$$

$\boldsymbol{h}$

$$h = \boldsymbol{e(\phi(P), \phi(Q))}$$
$$= \boldsymbol{e\left(P, \hat{\phi} \circ \phi(Q)\right)}$$
$$= \boldsymbol{e(P, [deg\ \phi]Q)}$$
$$\boxed{= \boldsymbol{e(P, Q)^{deg\ \phi}}}$$

h only depends on public information $(P, Q, \deg\phi)$, thus can be precomputed once and for all and made available in the public parameters.

Reverse basis decomposition

- Express $\{R_1, R_2\}$ in basis $\{\phi(P), \phi(Q)\}$

$$R_1 = c_1\phi(P) + c_2\phi(Q)$$

$$R_2 = d_1\phi(P) + d_2\phi(Q)$$

$$\boldsymbol{h} = \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{\phi}(\boldsymbol{Q}))$$ → fixed in the public params

$$\boldsymbol{h_0} = \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{R_1})$$

$$\boldsymbol{h_1} = \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{R_2})$$

$$\boldsymbol{h_2} = \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{Q}), \boldsymbol{R_1})$$

$$\boldsymbol{h_3} = \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{Q}), \boldsymbol{R_2})$$

4 pairings computed at runtime (NB: cost of 4-way Monty inv.: 12 muls)

Reverse basis decomposition

- Express $\{R_1, R_2\}$ in basis $\{\phi(P), \phi(Q)\}$

$$R_1 = c_1\phi(P) + c_2\phi(Q)$$
$$R_2 = d_1\phi(P) + d_2\phi(Q)$$

$$\boldsymbol{h} = \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{\phi}(\boldsymbol{Q})) \longrightarrow$$ fixed in the public params

$$\boldsymbol{h_0} = \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{R_1})$$
$$\boldsymbol{h_1} = \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{P}), \boldsymbol{R_2})$$
$$\boldsymbol{h_2} = \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{Q}), \boldsymbol{R_1})$$
$$\boldsymbol{h_3} = \boldsymbol{e}(\boldsymbol{\phi}(\boldsymbol{Q}), \boldsymbol{R_2})$$

} 4 pairings computed at runtime (NB: cost of 4-way Monty inv.: 12 muls)

$$c_1, c_2, d_1, d_2 = \log_h\{h_0, h_1, h_2, h_3\}$$ } recover M^{-1}

Reverse basis decomposition

- Reverting to $M = (M^{-1})^{-1}$, i.e., recover $\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{b_1}, \boldsymbol{b_2}$:

$$\begin{bmatrix} \boldsymbol{a_1} & \boldsymbol{a}_2 \\ \boldsymbol{b_1} & \boldsymbol{b_2} \end{bmatrix} = \frac{1}{\Delta} \begin{bmatrix} d_2 & -d_1 \\ -c_2 & c_1 \end{bmatrix}$$

where $\Delta = \det M^{-1} = c_1 d_2 - c_2 d_1 \ (mod\ \ell^e)$

Reverse basis decomposition

- Reverting to $M = (M^{-1})^{-1}$, i.e., recover $\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{b_1}, \boldsymbol{b_2}$:

$$\begin{bmatrix} \boldsymbol{a_1} & \boldsymbol{a_2} \\ \boldsymbol{b_1} & \boldsymbol{b_2} \end{bmatrix} = \frac{1}{\Delta} \begin{bmatrix} d_2 & -d_1 \\ -c_2 & c_1 \end{bmatrix}$$

where $\Delta = \det M^{-1} = c_1 d_2 - c_2 d_1 \ (mod\ \ell^e)$

- But Alice only sends (assuming a_1 invertible):

$$\alpha = b_1 a_1^{-1}$$

$$\beta = a_2 a_1^{-1}$$

$$\gamma = b_2 a_1^{-1}$$

Reverse basis decomposition

- Reverting to $M = (M^{-1})^{-1}$, i.e., recover $\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{b_1}, \boldsymbol{b_2}$:

$$\begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 \\ \boldsymbol{b}_1 & \boldsymbol{b}_2 \end{bmatrix} = \frac{1}{\Delta} \begin{bmatrix} d_2 & -d_1 \\ -c_2 & c_1 \end{bmatrix}$$

where $\Delta = \det M^{-1} = c_1 d_2 - c_2 d_1 \ (mod\ \ell^e)$

- But Alice only sends (assuming a_1 invertible):

$$\alpha = -\frac{c_2}{\Delta} \cdot \frac{\Delta}{d_2} = -\frac{c_2}{d_2}$$

$$\beta = -\frac{d_1}{\Delta} \cdot \frac{\Delta}{d_2} = -\frac{d_1}{d_2}$$

$$\gamma = \frac{c_1}{\Delta} \cdot \frac{\Delta}{d_2} = \frac{c_1}{d_2}$$

1 inv. + 3 muls. $(mod\ \ell^e)$
Same operations as before

Reverse basis decomposition

- Swapped (reduced) Tate pairing arguments

$$h_0 = e(\phi(P), R_1)$$
$$h_1 = e(\phi(P), R_2)$$
$$h_2 = e(\phi(Q), R_1)$$
$$h_3 = e(\phi(Q), R_2)$$

Reverse basis decomposition

- Swapped (reduced) Tate pairing arguments
- Second argument do not need to be cofactor reduced

$$h_0 = e(\phi(P), R'_1)$$

$$h_1 = e(\phi(P), R'_2)$$

$$h_2 = e(\phi(Q), R'_1)$$

$$h_3 = e(\phi(Q), R'_2)$$

such that $[h]R_i' = R_i$

Reverse basis decomposition

- Swapped (reduced) Tate pairing arguments
- Second argument do not need to be cofactor reduced

$$h_0 = e(\phi(P), R'_1)$$

$$h_1 = e(\phi(P), R'_2)$$

$$h_2 = e(\phi(Q), R'_1)$$

$$h_3 = e(\phi(Q), R'_2)$$

such that $[h]R_i' = R_i$

$$\boldsymbol{R_1}' = \boldsymbol{c_1}'\phi(P) + \boldsymbol{c_2}'\phi(Q)$$

$$\boldsymbol{R_2}' = \boldsymbol{d_1}'\phi(P) + \boldsymbol{d_2}'\phi(Q)$$

s.t. $[h]c_i' = c_i, [h]d_i' = d_i$

DLOGs are up to cofactor h^{-1}

Simply post-multiply by h in $\mathbb{Z}_{\ell^e}$

Reverse basis decomposition

- Swapped (reduced) Tate pairing arguments
- Second argument do not need to be cofactor reduced

$$h_0 = e(\phi(P), R'_1)$$

$$h_1 = e(\phi(P), R'_2)$$

$$h_2 = e(\phi(Q), R'_1)$$

$$h_3 = e(\phi(Q), R'_2)$$

such that $[h]R_i' = R_i$

$$\boldsymbol{R_1}' = \boldsymbol{c_1}'\phi(P) + \boldsymbol{c_2}'\phi(Q)$$

$$\boldsymbol{R_2}' = \boldsymbol{d_1}'\phi(P) + \boldsymbol{d_2}'\phi(Q)$$

s.t. $[h]c_i' = c_i, [h]d_i' = d_i$

DLOGs are up to cofactor h^{-1}

Simply post-multiply by h in $\mathbb{Z}_{\ell^e}$

- Two scalar muls. by 3^n saved in the binary torsion using Entangled Basis.

SIDH Public Key Compression

- Most costly operations:
  I. Computing a basis $\{\boldsymbol{R_1}, \boldsymbol{R_2}\}$
  II. Computing 5 pairings
  III. Computing 4 discrete logs
- New algorithms to address the above bottlenecks.
  I. Entangled basis for the (Alice) binary $\boldsymbol{2^m}$-torsion

  Idea: generate a candidate basis $\{R_1, R_2\}$ by "subverting Elligator 2" formulas

"Entangled" basis generation

- Elligator 2 in a nutshell:
  - Montgomery curve: $\mathrm{E}/\mathbb{F}_{\mathrm{p}^2}\colon By^2 = x^3 + Ax^2 + x$
  - Let $u \in \mathbb{F}_{p^2}$ be a non-square.
  - Define $v \coloneqq 1/(1 + ur^2)$ where $r \in \mathbb{F}_{\mathrm{p}^2}$.
  - [Thm. Bernstein *et al.*] If u is a non-square, then exactly one of

$$\boldsymbol{x = -Av}$$

or

$$\boldsymbol{x = Av - A}$$

is the abscissa of a point on E.

“Entangled” basis generation

- Recall: to build a basis for $E[2^m]$ we need two full order L.I. points
- Getting points of order 2^m on Montgomery curves is cheaper using the 2-descent:
  - A point (x, y) is not in the image of $[2]E$ iff x is a non-square.
- Search only for non-square abscissas.

“Entangled” basis generation

- The entangled basis for $E[2^m]$:
  - Montgomery curve: $\mathrm{E}/\mathbb{F}_{\mathrm{p}^2}: By^2 = x^3 + Ax^2 + x$
  - Let $u \in \mathbb{F}_{p^2}$ be a ~~non~~-square where $\boldsymbol{u} = \boldsymbol{u_0^2}$ for $\boldsymbol{u_0} \in \mathbb{F}_{\boldsymbol{p^2}} \setminus \mathbb{F}_{\boldsymbol{p}}$.
  - Define 2 tables $\boldsymbol{T_s}, \boldsymbol{T_n}$ of pairs $(\mathbf{r}, \boldsymbol{v} \coloneqq \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{ur^2}})$ that contain only v squares and non-squares, respectively, and $\boldsymbol{r} \in \mathbb{F}_{\boldsymbol{p}}$.
  - If A is square we pick candidates v from T_n such that $\boldsymbol{x} = -\boldsymbol{Av}$ is non-square and pick v from T_s otherwise.
  - Theorem: choosing the parameters as above, the points whose abscissas are

    $$\boldsymbol{x} = -\boldsymbol{Av} \quad \text{and} \quad \boldsymbol{x} = \boldsymbol{Av} - \boldsymbol{A}$$

    are either both not on E or both on E, of order multiple of 2^m and linear independent.

Faster Basis Generation

- Entangled Basis $E[2^m] = \langle [3^n]S_1, [3^n]S_2 \rangle$
  - Find one basis point and the other is for free!
  - Two cofactor multiplications by 3^n saved on compression!
    - Recall Bob can compute $e_{2^n}(\phi(*), R'_i)$ and still compress his key
  - No L.I. test required!
    - Previous works remove cofactors 3^n and multiply both candidate points by 2^{m-1}.
  - Theoretical estimates and practical experiments show a 15× (!) speedup

SIDH Public Key Compression

- Most costly operations:
  1. Computing a basis $\{R_1, R_2\}$
  2. Computing 5 pairings
  3. Computing 4 discrete logs
- New algorithms to address the three above bottlenecks.
  - In addition to the reduction in number of pairings we investigated the plain Tate pairing over Weierstrass form with Jacobian coordinates and notice a faster pairing computation than Costello *et al.*'s version based on Montgomery-like formulas.
  - No need to store numerators and denominators separately due to (partial) denominator elimination.
  - Improvement of about 28% for binary and 22% for ternary pairings.

SIDH Public Key Compression

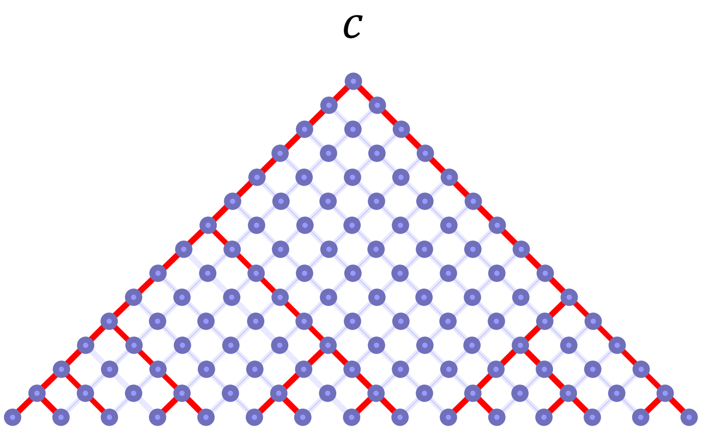
- Most costly operations:
  I. Computing a basis $\{R_1, R_2\}$
  II. Computing 5 pairings
  III. Computing 4 discrete logs
- New algorithms to address the three above bottlenecks.
  III. An optimal strategy for Pohlig-Hellman
    - Inspired by Shoup's RDL method
    - Adopts Jao-De Feo-Plût's isogeny computation to obtain optimal strategy
    - Attain $O(e \lg e)$ complexity which was informally conjectured by Shoup
    - Combination is non-trivial (more improvements for DL than are possible for isogeny computation)

Discrete log and optimal strategy

$c \in \mu_{\ell^e}$

$c = g^{d_0 + d_1 \ell + \cdots + d_{e-1} \ell^{e-1}}$

$g = e_{\ell^e}(P, Q)^{deg \phi}$



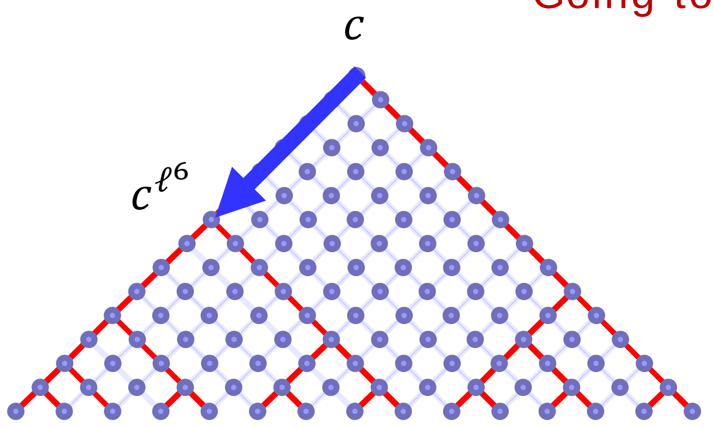
Discrete log and optimal strategy

$c \in \mu_{\ell^e}$

$c = g^{d_0 + d_1 \ell + \cdots + d_{e-1} \ell^{e-1}}$

$g = e_{\ell^e}(P, Q)^{deg \phi}$

Going to the left raises to the ℓ

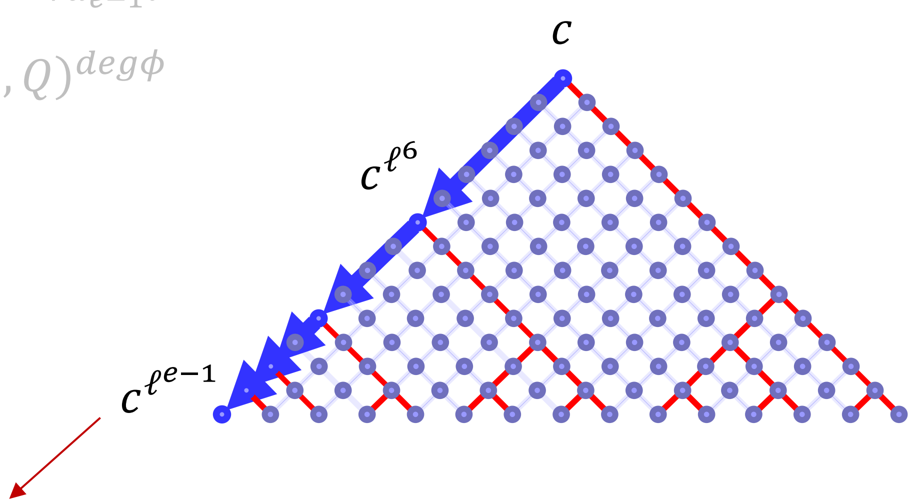


Discrete log and optimal strategy

$c \in \mu_{\ell^e}$

$c = g^{d_0 + d_1\ell + \cdots + d_{e-1}\ell^{e-1}}$

$g = e_{\ell^e}(P, Q)^{deg\phi}$



Element of order ℓ, thus $c^{\ell^{e-1}} = g^{d_0}$ (by Pohlig-Hellman we can recover all d_i)

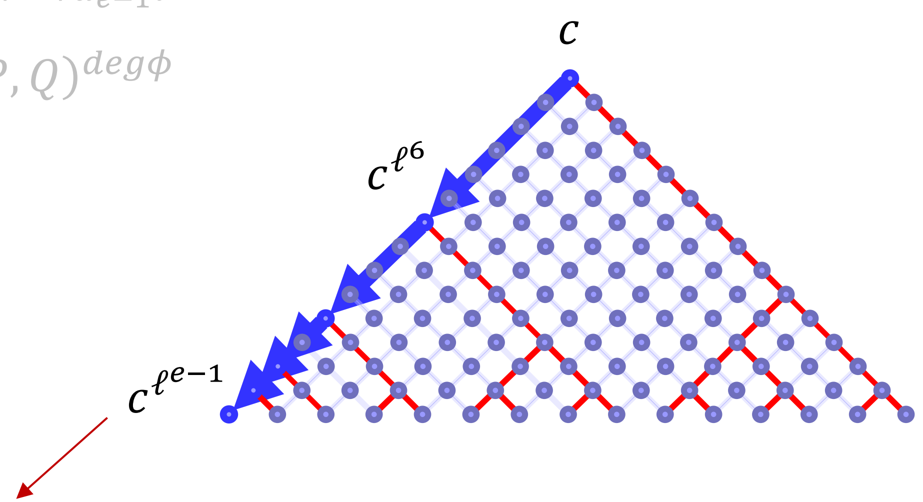
Recover small discrete log. using brute force $d_0 = \log_{g^{\ell^{e-1}}} c^{\ell^{e-1}}$

Discrete log and optimal strategy

$c \in \mu_{\ell^e}$

$c = g^{d_0 + d_1\ell + \cdots + d_{e-1}\ell^{e-1}}$

$g = e_{\ell^e}(P, Q)^{deg\phi}$



Element of order ℓ, thus $c^{\ell^{e-1}} = g^{d_0}$ (by Pohlig-Hellman we can recover all d_i)

Recover small discrete log. using brute force $d_0 = \log_{g^{\ell^{e-1}}} c^{\ell^{e-1}}$

g is fixed, use the powers $g^{0\ell^{e-1}}, g^{1\ell^{e-1}}, \cdots, g^{(\ell-1)\ell^{e-1}}$ (due to RBD),

so only comparisons are done in the loop instead of exponentiations.

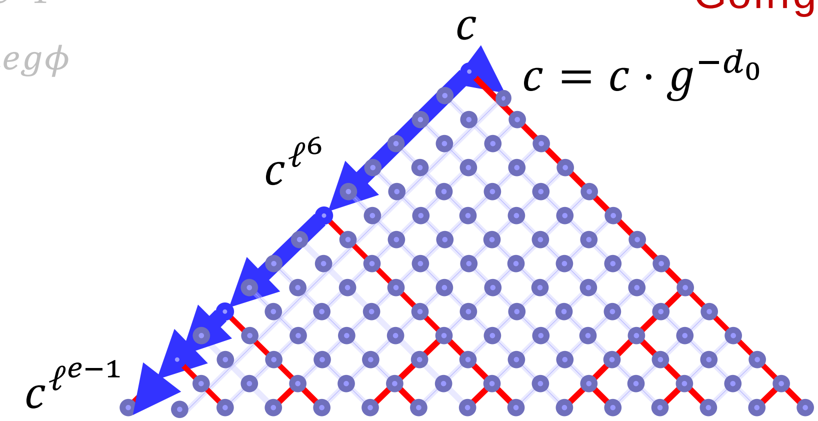
Discrete log and optimal strategy

$c \in \mu_{\ell^e}$

$c = g^{d_0 + d_1 \ell + \cdots + d_{e-1} \ell^{e-1}}$

$g = e_{\ell^e}(P, Q)^{\deg \phi}$

Going to the right erases the digit

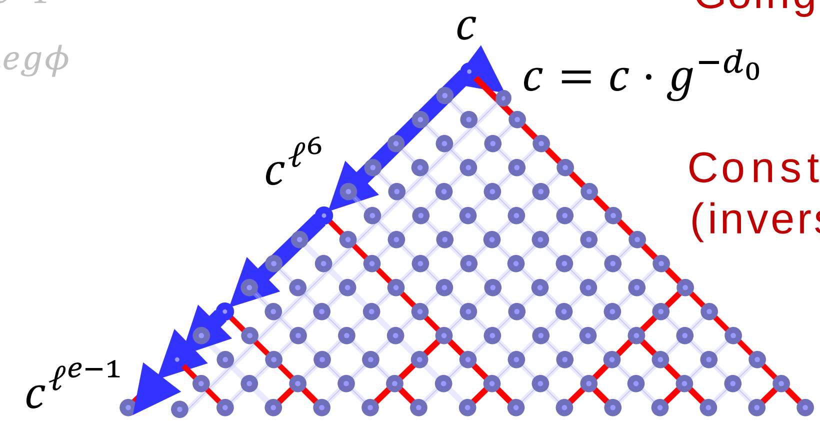


Discrete log and optimal strategy

$c \in \mu_{\ell^e}$

$c = g^{d_0 + d_1\ell + \cdots + d_{e-1}\ell^{e-1}}$

$g = e_{\ell^e}(P, Q)^{\deg\phi}$



Going to the right erases the digit

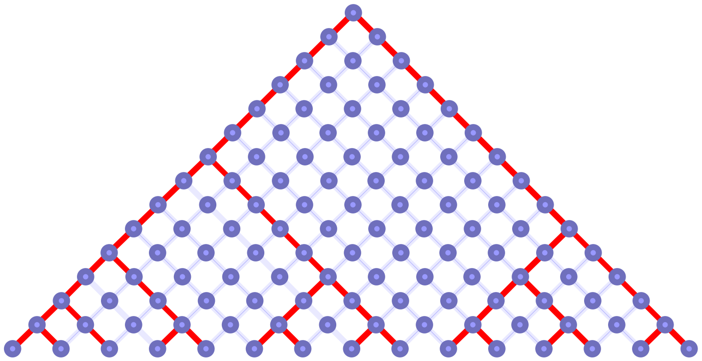
Constant cost: $1M$ + negation
(inversion is just a conjugation in μ_{ℓ^e})

Discrete log and optimal strategy

$c \in \mu_{\ell^e}$

$c = g^{d_0 + d_1 \ell + \cdots + d_{e-1} \ell^{e-1}}$

$g = e_{\ell^e}(P, Q)^{\deg \phi}$



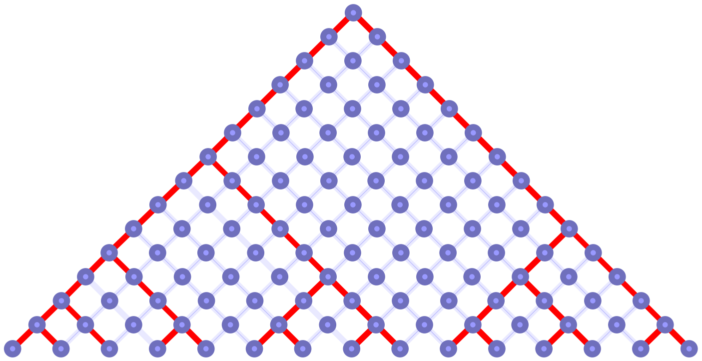
- This problem reminds exactly the computation of ℓ^e-degree isogenies.
  - Use Jao-De Feo-Plut algorithm to compute optimal strategy in $O(e \lg e)$

Discrete log and optimal strategy

$c \in \mu_{\ell^e}$

$c = g^{d_0 + d_1 \ell + \cdots + d_{e-1} \ell^{e-1}}$

$g = e_{\ell^e}(P, Q)^{deg\phi}$



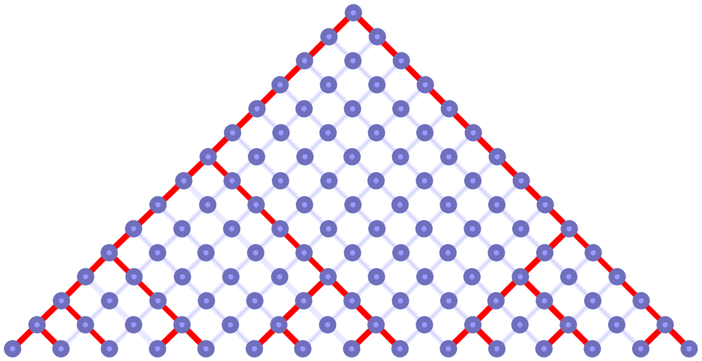
- This problem reminds exactly the computation of ℓ^e-degree isogenies.
  - Use Jao-De Feo-Plut algorithm to compute optimal strategy in $O(e \lg e)$
- Side-product: generate opt-strategy from $O(e^2)$ to $O(e \log e)$
  - One could compute the strategy "on-the-fly"

Discrete log and optimal strategy

$c \in \mu_{\ell^e}$

$c = g^{d_0 + d_1 \ell + \cdots + d_{e-1} \ell^{e-1}}$

$g = e_{\ell^e}(P, Q)^{deg \phi}$



- This problem reminds exactly the computation of ℓ^e-degree isogenies.
  - Use Jao-De Feo-Plut algorithm to compute optimal strategy in $O(e \lg e)$
- Side-product: generate opt-strategy from $O(e^2)$ to $O(e \log e)$
  - One could compute the strategy "on-the-fly"
- Possible to use windowed-DL to recover $d_i \; mod \; \ell^w$ at each leaf.

Discrete log and optimal strategy

Table 3: Discrete logarithm computation costs (assuming $\mathbf{s} \approx 0.8\mathbf{m}$)

group	Costello *et al.* [5]	ours, $w = 1$ (ratio)	ours, $w = 3$ (ratio)	ours, $w = 6$ (ratio)
$\mu_{2^{372}}$	8271.6$\mathbf{m}$	4958.4$\mathbf{m}$ (0.60)	3127.9$\mathbf{m}$ (0.39)	2103.7$\mathbf{m}$ (0.25)
$\mu_{3^{239}}$	7999.2$\mathbf{m}$	4507.6$\mathbf{m}$ (0.56)	2638.1$\mathbf{m}$ (0.33)	1739.8$\mathbf{m}$ (0.22)

Binary discrete logs: 1.7×−4× faster

Ternary discrete logs: 1.8×−4.6× faster

Implementation

- No need for isochronous methods (only public information involved).
- C implementation available on GitHub (fork of MSR PQCrypto-SIDH)

Table 4: Benchmarks in cycles on an Intel Core i5 clocked at 2.9 GHz (clang compiler with -O3 flag, and s = m in this implementation).

operations	2^m-torsion ($w = 2$)			3^n-torsion ($w = 1$)		
	SIDH v2.0 [5]	ours	ratio	SIDH v2.0 [5]	ours	ratio
basis generation	24497344	1690452	**14.49**	20632876	17930437	**1.15**
discrete log.	6206319	2776568	**2.24**	4710245	3069234	**1.53**
pairing phase	33853114	25755714	**1.31**	39970384	30763841	**1.30**
compression	78952537	38755681	**2.04**	78919488	61768917	**1.28**
decompression	30057506	9990949	**3.01**	25809348	23667913	**1.09**

- Binary torsion
  - Compression time reduced by 2×. Expect > 3× using larger w.
  - Decompression time reduced by 3×

Implementation

- No need for isochronous methods (only public information involved).
- C implementation available on GitHub (fork of MSR PQCrypto-SIDH)

Table 4: Benchmarks in cycles on an Intel Core i5 clocked at 2.9 GHz (clang compiler with `-O3` flag, and $\mathbf{s} = \mathbf{m}$ in this implementation).

	2^m-torsion ($w = 2$)			3^n-torsion ($w = 1$)		
operations	SIDH v2.0 [5]	ours	ratio	SIDH v2.0 [5]	ours	ratio
basis generation	24497344	1690452	**14.49**	20632876	17930437	**1.15**
discrete log.	6206319	2776568	**2.24**	4710245	3069234	**1.53**
pairing phase	33853114	25755714	**1.31**	39970384	30763841	**1.30**
compression	78952537	38755681	**2.04**	78919488	61768917	**1.28**
decompression	30057506	9990949	**3.01**	25809348	23667913	**1.09**

- Ternary torsion
  - Compression $1.3\times$ speedup. Expect $> 2\times$ using larger w
  - Decompression time reduced by $1.1\times$. (new improvements will be available soon)

Summary

- Improvements in all compression bottlenecks
- Publicly source code on top of the well-known SIDH library
- Other results:
  - Faster point tripling: 5M+6S instead of 6M+5S by Rao *et al*
  - Slightly faster 3-torsion basis generation
- Future work:
  - Generalize entangled basis for non-binary torsions (seems hard)
  - Improve the new bottleneck (pairings)

Questions?

Geovandro C. C. F. Pereira
geovandro.pereira@uwaterloo.ca

UNIVERSITY OF WATERLOO | IQC Institute for Quantum Computing

Questions?

Thanks!

Geovandro C. C. F. Pereira
geovandro.pereira@uwaterloo.ca

University of Waterloo | IQC Institute for Quantum Computing

References

- [2011] Jao, D. and De Feo, L. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In International Workshop on Post-Quantum Cryptography (pp. 19-34). Springer, Berlin, Heidelberg.

- [2016] Azarderakhsh, R., Jao, D., Kalach, K., Koziel, B. and Leonardi, C. Key compression for isogeny-based cryptosystems. In *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography* (pp. 1-10). ACM.

- [2017] Costello, C., Jao, D., Longa, P., Naehrig, M., Renes, J. and Urbanik, D. Efficient compression of SIDH public keys. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 679-706). Springer, Cham.

SIDH Public Key Compression

Appendix

IMPROVED POINT TRIPLING

Point tripling

- New xz-only tripling algorithm for the Montgomery curve $E : By^2 = x^3 + Ax^2 + x$.
- Cost: $5\mathbf{M} + 6\mathbf{S} + 9\mathbf{A}$ (counting any left shift as an addition).
- Best previous algorithm in the literature (by S. R. S. Rao) only attains $6\mathbf{M} + 5\mathbf{S} + 7\mathbf{A}$.
- Given (x, z), compute $(x_3, z_3) = 3 \cdot (x, z)$:
  - $t_1 \leftarrow x^2$, $t_2 \leftarrow z^2$, $t_3 \leftarrow (t_1 - t_2)^2$,
  - $t_s \leftarrow t_1 + t_2$, $t_4 \leftarrow (x + z)^2 - t_s$,
  - $t_4 \leftarrow t_3 \cdot (A/2)$, $t_5 \leftarrow 4t_2$, $t_6 \leftarrow 4t_1$,
  - $t_4 \leftarrow t_4 + t_s$, $t_7 \leftarrow t_4 \cdot t_5$, $t_8 \leftarrow t_4 \cdot t_6$,
  - $t_1 \leftarrow (t_3 - t_7)^2$, $t_2 \leftarrow (t_3 - t_8)^2$,
  - $x_3 \leftarrow x \cdot t_1$, $z_3 \leftarrow z \cdot t_2$.

ENTANGLED BASIS

Faster Basis Generation

- Entangled Basis generation for $E[2^m]$
  - 2-descent used to get points of full order 2^m.
    - 2-descent: given $E/F_q: y^2 = (x-\alpha_1)(x-\alpha_2)(x-\alpha_3)$, then a point $(x', y') \in 2E$ iif $x' - \alpha_1, x' - \alpha_2,\ x' - \alpha_3$ are all squares in F_q.
    - Corollary: for a Montgomery curve $E_M/F_{p^2}: By^2 = x(x^2 + Ax + 1)$, a point $(x', y') \notin 2E$ iif x' is non-square in F_{p^2}.
    - Therefore, in order to find full order 2^m points, run through candidates (precomputed table of non-squares) where x' is non-square.

“Entangled” basis generation

- Entangled algorithm(A, u_0, u):
  - test $\boldsymbol{A} =: \boldsymbol{a} + \boldsymbol{b}\boldsymbol{i}$:
    $\boldsymbol{z} \leftarrow \boldsymbol{a}^2 + \boldsymbol{b}^2$
    $\boldsymbol{s} \leftarrow \boldsymbol{z}^{(p+1)/4}$
    check $\boldsymbol{s}^2 = \boldsymbol{z}$
  - repeat // k times
    lookup next entry $\left(r,\ v = 1/(1 + ur^2)\right)$ from T
    $x \leftarrow -A \cdot v$ // (NB: x nonsquare)
    $t \leftarrow x \cdot (x^2 + A \cdot x + 1)$
    test $t =: c + di$ quadraticity:
    $z \leftarrow c^2 + d^2$
    $s \leftarrow z^{(p+1)/4}$
    until $s^2 = z$
  - compute $y \leftarrow \sqrt{x^3 + A \cdot x^2 + x}$:
    $z \leftarrow (c + s)/2$
    $\alpha \leftarrow z^{(p+1)/4}$
    $\beta \leftarrow d \cdot (2\alpha)^{-1}$
    $y \leftarrow (\alpha^2 = z)\ ?\ \alpha + \beta i : -\beta + \alpha i$
  - compute basis:
    $S_1 \leftarrow (x, y)$, $S_2 \leftarrow (ur^2 x, u_0 r y)$ // low cost for small r

Test A quadraticity and select $T \leftarrow T_s\ (or\ T_n)$

“Entangled” basis generation

- Entangled algorithm(A, u_0, u):
  - test $A =: a + bi$:
    $z \leftarrow a^2 + b^2$
    $s \leftarrow z^{(p+1)/4}$
    check $s^2 = z$
  - repeat // $\boldsymbol{k}$ times
    lookup next entry $\left(\boldsymbol{r},\ \boldsymbol{v} = \mathbf{1}/\left(\mathbf{1} + \boldsymbol{ur}^{\mathbf{2}}\right)\right)$ from T //free
    $\boldsymbol{x} \leftarrow -\boldsymbol{A} \cdot \boldsymbol{v}$ // (NB: x nonsquare)
    $\boldsymbol{t} \leftarrow \boldsymbol{x} \cdot \left(\boldsymbol{x}^{\mathbf{2}} + \boldsymbol{A} \cdot \boldsymbol{x} + \mathbf{1}\right)$
    test $\boldsymbol{t} =: \boldsymbol{c} + \boldsymbol{di}$ quadraticity:
    $\boldsymbol{z} \leftarrow \boldsymbol{c}^{\mathbf{2}} + \boldsymbol{d}^{\mathbf{2}}$
    $\boldsymbol{s} \leftarrow \boldsymbol{z}^{(\boldsymbol{p}+\mathbf{1})/\mathbf{4}}$
    until $\boldsymbol{s}^{\mathbf{2}} = \boldsymbol{z}$
  - compute $y \leftarrow \sqrt{x^3 + A \cdot x^2 + x}$:
    $z \leftarrow (c + s)/2$
    $\alpha \leftarrow z^{(p+1)/4}$
    $\beta \leftarrow d \cdot (2\alpha)^{-1}$
    $y \leftarrow (\alpha^2 = z)\ ?\ \alpha + \beta i : -\beta + \alpha i$
  - compute basis:
    $S_1 \leftarrow (x, y)$, $S_2 \leftarrow (ur^2 x, u_0 r y)$ // low cost for small r

Test A quadraticity and select $T \leftarrow T_s\ (or\ T_n)$

Find first candidate on E

"Entangled" basis generation

- Entangled algorithm(A, u_0, u):
  - test $A =: a + bi$:
    $z \leftarrow a^2 + b^2$
    $s \leftarrow z^{(p+1)/4}$
    check $s^2 = z$
  - repeat // k times
    lookup next entry $\left(r,\ v = 1/(1 + ur^2)\right)$ from T //free
    $x \leftarrow -A \cdot v$ // (NB: x nonsquare)
    $t \leftarrow x \cdot (x^2 + A \cdot x + 1)$
    test $t =: c + di$ quadraticity:
    $z \leftarrow c^2 + d^2$
    $s \leftarrow z^{(p+1)/4}$
    until $s^2 = z$
  - compute $\boldsymbol{y} \leftarrow \sqrt{\boldsymbol{x}^3 + \boldsymbol{A} \cdot \boldsymbol{x}^2 + \boldsymbol{x}}$:
    $\boldsymbol{z} \leftarrow (\boldsymbol{c} + \boldsymbol{s})/2$
    $\boldsymbol{\alpha} \leftarrow \boldsymbol{z}^{(p+1)/4}$
    $\boldsymbol{\beta} \leftarrow \boldsymbol{d} \cdot (2\boldsymbol{\alpha})^{-1}$
    $\boldsymbol{y} \leftarrow \left(\boldsymbol{\alpha}^2 = \boldsymbol{z}\right)\, ?\, \boldsymbol{\alpha} + \boldsymbol{\beta i} : -\boldsymbol{\beta} + \boldsymbol{\alpha i}$
  - compute basis:
    $S_1 \leftarrow (x, y),\ S_2 \leftarrow (ur^2 x, u_0 r y)$ // low cost for small r

Test A quadraticity and select $T \leftarrow T_s$ $(or\ T_n)$

Find first candidate on E

Recover y of first candidate on E

“Entangled” basis generation

- Entangled algorithm(A, u_0, u):
  - test $A =: a + bi$:
    $z \leftarrow a^2 + b^2$
    $s \leftarrow z^{(p+1)/4}$
    check $s^2 = z$
  - repeat // k times
    lookup next entry $\big(r,\ v = 1/(1 + ur^2)\big)$ from T //free
    $x \leftarrow -A \cdot v$ // (NB: x nonsquare)
    $t \leftarrow x \cdot (x^2 + A \cdot x + 1)$
    test $t =: c + di$ quadraticity:
    $z \leftarrow c^2 + d^2$
    $s \leftarrow z^{(p+1)/4}$
    until $s^2 = z$
  - compute $\boldsymbol{y} \leftarrow \sqrt{\boldsymbol{x}^3 + \boldsymbol{A} \cdot \boldsymbol{x}^2 + \boldsymbol{x}}$:
    $\boldsymbol{z} \leftarrow (\boldsymbol{c} + \boldsymbol{s})/2$
    $\boldsymbol{\alpha} \leftarrow \boldsymbol{z}^{(\boldsymbol{p}+1)/4}$
    $\boldsymbol{\beta} \leftarrow \boldsymbol{d} \cdot (2\boldsymbol{\alpha})^{-1}$
    $\boldsymbol{y} \leftarrow \big(\boldsymbol{\alpha}^2 = \boldsymbol{z}\big)\ ?\ \boldsymbol{\alpha} + \boldsymbol{\beta i} : -\boldsymbol{\beta} + \boldsymbol{\alpha i}$
  - compute basis:
    $S_1 \leftarrow (x, y)$, $S_2 \leftarrow (ur^2x, u_0ry)$ // low cost for small r

Test A quadraticity and select $T \leftarrow T_s$ $(or\ T_n)$

Find first candidate on E

Recover y of first candidate on E

Second candidate