Memory Optimization Techniques for Computing Discrete Logarithms in Compressed SIKE

A. Hutchinson [1], K. Karabina [1,2], G. Pereira [1,3]

[1]University of Waterloo Canada
[2]National Research Council Canada
[3]evolutionQ Inc. Canada

PQCrypto 2021, July 20-22, 2021

Acknowledgements: SIKE team members; NSERC; CryptoWorks21; Canada First Research Excellence Fund; Public Works and Government Services Canada; National Research Council Canada and the University of Waterloo Collaboration Center

Outline

1. Isogenies, SIKE, and compressed SIKE
2. Mathematical background and the problem definition
3. Contributions
4. Implementation results

Notation

- $p = {\ell_A}^{e_A}{\ell_B}^{e_B} - 1$ is prime, $\mathbb{F}_q$ is a finite field of size $q = p^2$
  - $\ell_A = 2$, $\ell_B = 3$
  - $(e_A, e_B) \in \{(216, 137), (250, 159), (305, 192), (372, 239)\}$
  - $\mathbb{F}_q = \mathbb{F}_p[i]/\langle i^2 + 1\rangle$
- E is a supersingular elliptic curve defined over $\mathbb{F}_q$
  - $E/\mathbb{F}_q: \ \{(x, y): \ y^2 = x^3 + 6x^2 + x\} \cup \mathcal{O}$
- $E(\mathbb{F}_q)$ denotes the set of $\mathbb{F}_q$-points on E
  - $|E(\mathbb{F}_q)| = (p + 1)^2$
- $E[\ell^e]$ denotes the ℓ^e-torsion group of E ($\ell^e P = \mathcal{O}$ for $P \in E[\ell^e]$)
  - $E[\ell^e] \cong \mathbb{Z}_{\ell^e} \oplus \mathbb{Z}_{\ell^e}$
  - $E[{\ell_A}^{e_A}] = \langle P_A, Q_A\rangle$ and $E[{\ell_B}^{e_B}] = \langle P_B, Q_B\rangle$

Isogenies

- Let E_1, E_2 be two elliptic curves over $\mathbb{F}_q$
- An *isogeny* $\phi : E_1 \to E_2$ is a non-constant rational map such that
  - ϕ is defined over $\mathbb{F}_q$
  - $\phi(\mathcal{O}) = \mathcal{O}$
- $\phi : E_1(\mathbb{F}_q) \to E_2(\mathbb{F}_q)$ is a group homomorphism
- $\ker(\phi) = \{P \in E_1 : \phi(P) = \mathcal{O}\}, \ E_2 \cong E_1/\ker(\phi)$
- Conversely, given a subgroup K of $E_1(\mathbb{F}_q)$, there exists (unique up to isomorphism)
$$\phi : E_1 \to E_2, \quad \text{where } \ker(\phi) = K$$
- Given K, ϕ and E_2 can be expressed explicitly (may not be efficient)

SIKE: Supersingular Isogeny Key Encapsulation

- SIKE is based on the supersingular isogeny Diffie-Hellman (SIDH) protocol (Jao and De Feo, 2011)
- SIKE was submitted to the NIST post-quantum cryptography standardization process (2017)
- SIKE was announced as one of the five alternate candidates in the public key encryption and key establishment category (2020)
- In SIKE, two parties A and B compute a shared secret key, which is essentially the j-invariant of the two isomorphic curves E_{AB} and E_{BA}: $j = j(E_{AB}) = j(E_{BA})$

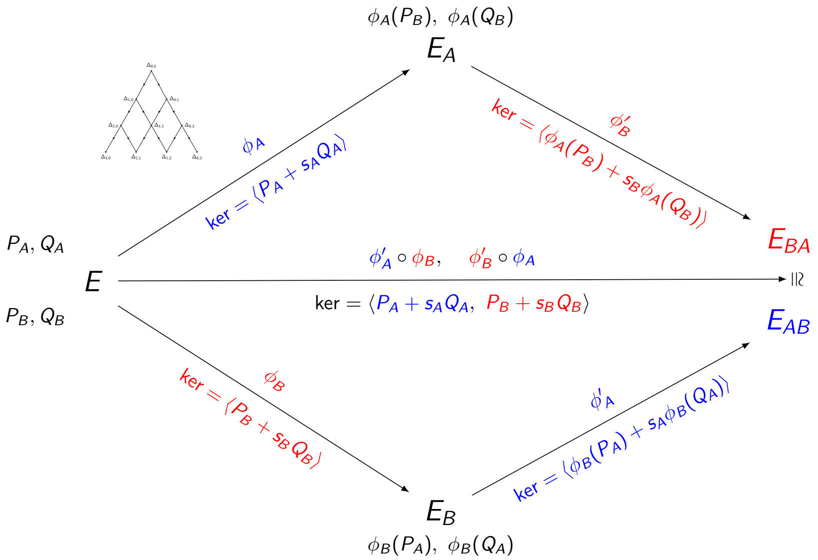
SIKE: Supersingular Isogeny Key Encapsulation

- SIKE is based on the supersingular isogeny Diffie-Hellman (SIDH) protocol (Jao and De Feo, 2011)
- SIKE was submitted to the NIST post-quantum cryptography standardization process (2017)
- SIKE was announced as one of the five alternate candidates in the public key encryption and key establishment category (2020)
- In SIKE, two parties A and B compute a shared secret key, which is essentially the j-invariant of the two isomorphic curves E_{AB} and E_{BA}: $j = j(E_{AB}) = j(E_{BA})$

A basic setting for SIKE:

- **Public parameters:** $p = 2^{e_2}3^{e_3} - 1$, $E : y^2 = x^3 + 6x^2 + x$, $E[2^{e_2}] = \langle P_A, Q_A \rangle$, $E[3^{e_3}] = \langle P_B, Q_B \rangle$
- **Secret key of A:** $s_A \in \mathbb{Z}_{2^{e_2}}$
- **Public key of A:** E_A, $\phi_A(P_B)$ $\phi_A(Q_B)$, where $\phi_A : E \to E_A$ is a secret isogeny with $\ker(\phi_A) = \langle P_A + s_A Q_A \rangle$
- The secret/public keys of B are defined similarly

Computations in SIKE



Compressing SIKE public keys

- Naive method:
  - Public key of A: $E_A : y^2 = x^3 + ax + b$, $a, b \in \mathbb{F}_q$; $P = \phi_A(P_B) = (x_P, y_P)$, $Q = \phi_A(Q_B) = (x_Q, y_Q) \in E[\ell^e]$
  - $a, b, x_P, y_P \in \mathbb{F}_q$ can be represented by $8 \log p$ bits

Compressing SIKE public keys

- Naive method:
  - Public key of A: $E_A : y^2 = x^3 + ax + b$, $a, b \in \mathbb{F}_q$; $P = \phi_A(P_B) = (x_P, y_P)$, $Q = \phi_A(Q_B) = (x_Q, y_Q) \in E[\ell^e]$
  - $a, b, x_P, y_P \in \mathbb{F}_q$ can be represented by $8 \log p$ bits

- Optimizations (Azarderakhsh *et al.*, 2016; Costello *et al.*, 2017; Gustavo *et al.*, 2018; Naehrig and Renes *et al.*, 2019; Pereira *et al.*, 2020):
  - Replace a, b by the j-invariant of E_A: $4 \log p \to 2 \log p$
  - Fix a basis for $E_A[\ell^e] = \langle R, S \rangle$ and rewrite

$$P = a_P R + b_P S; \;\; Q = a_Q R + b_Q S$$

  - Replace x_P, x_Q by $a_P, b_P, a_Q, b_Q \in \mathbb{Z}_{\ell^e}$: $4 \log p \to 2 \log p$ ($\ell^e \approx \sqrt{p}$)
  - Rescale $[a_P, b_P, a_Q, b_Q]$ by a_P^{-1}: $2 \log p \to 1.5 \log p$
  - Public key compression: $8 \log p \to 3.5 \log p$

Computational overheads in compressing public keys

- How to compute a_P, b_P, a_Q, b_Q?
- Compute g and h, where

$$g = e(R, S)$$
$$h = e(P, S) = e(a_P R + b_P S, S) = g^{a_P},$$

where $e : E_A[\ell^e] \times E_A[\ell^e] \to \mu_{\ell^e}$ is a bilinear pairing function, and μ_{ℓ^e} is the multiplicative group of order ℓ^e

- Compute a_P by solving the discrete logarithm of h base g
- Similarly, compute b_P, a_Q, b_Q
- Since ℓ is small, we use the Pohlig-Hellman algorithm to solve logarithms

The Discrete logarithm problem (DLP)

- p is prime, $p \equiv 3 \pmod 4$, $\mathbb{F}_q = \mathbb{F}_p[i]/\langle i^2 + 1 \rangle$
- $\mathbb{G}$ is the order-$(p+1)$ cyclotomic subgroup of $\mathbb{F}_q^*$
- For positive ℓ and ω with $\ell^\omega \mid (p+1)$, $\mathbb{G}_{\ell,\omega}$ is the order-ℓ^ω subgroup of $\mathbb{G}$
- $\mathbb{G}_{\ell,e} = \langle g \rangle$ is the largest of $\mathbb{G}_{\ell,\omega} = \langle \rho \rangle$, and $\rho = g^{\ell^{e-\omega}}$
- In particular, we are interested in:
  - $p = 2^{e_A} 3^{e_B} - 1$
  - $\ell = 2$ and $e \in \{216, 250, 305, 372\}$
  - $\ell = 3$ and $e \in \{137, 159, 192, 239\}$

Problem

Given $\mathbb{G}_{\ell,e} = \langle g \rangle$ and $h \in \mathbb{G}_{\ell,e}$, find $d \in \mathbb{Z}_{\ell^e}$ such that $g^d = h$

Remark

Inverses can be computed for free in $\mathbb{G}$:

$$h = (a + bi) \in \mathbb{G} \Rightarrow 1 = h^{p+1} = (a + bi)^p (a + pi) = (a - bi)(a + bi)$$

The Pohlig-Hellman (PH) algorithm

Problem

Given $\mathbb{G}_{\ell,e} = \langle g \rangle$ *and* $h \in \mathbb{G}_{\ell,e}$*, find* $d \in \mathbb{Z}_{\ell^e}$ *such that* $g^d = h$*.*

1. For $0 \leq k < e$ and $0 \leq d < \ell$, precompute and store the table

$$T[k][d] = g^{-d\ell^k}$$

2. Write $d = \sum_{i=0}^{e-1} d_i \ell^i$, $d_i \in [0, \ell)$ and use $g^{\ell^e} = 1$
3. Compute

$$\Delta_{0,0} = h = g^{\sum_{i=0}^{e-1} d_i \ell^i}, \; \Delta_{e-1,0} = \Delta_{0,0}^{\ell^{e-1}} = (g^{\ell^{e-1}})^{d_0} \in \mathbb{G}_{\ell,1}$$

4. Determine d_0: if $\Delta_{e-1,0} == T[e-1][d]^{-1}$ then $d_0 \leftarrow d$

The Pohlig-Hellman (PH) algorithm

Problem

Given $\mathbb{G}_{\ell,e} = \langle g \rangle$ and $h \in \mathbb{G}_{\ell,e}$, find $d \in \mathbb{Z}_{\ell^e}$ such that $g^d = h$.

1. For $0 \leq k < e$ and $0 \leq d < \ell$, precompute and store the table

$$T[k][d] = g^{-d\ell^k}$$

2. Write $d = \sum_{i=0}^{e-1} d_i \ell^i$, $d_i \in [0, \ell)$ and use $g^{\ell^e} = 1$
3. Compute

$$\Delta_{0,0} = h = g^{\sum_{i=0}^{e-1} d_i \ell^i},\ \Delta_{e-1,0} = \Delta_{0,0}^{\ell^{e-1}} = (g^{\ell^{e-1}})^{d_0} \in \mathbb{G}_{\ell,1}$$

4. Determine d_0: if $\Delta_{e-1,0} == T[e-1][d]^{-1}$ then $d_0 \leftarrow d$
5. Compute

$$\Delta_{0,1} = \Delta_{0,0} T[0][d_0] = g^{\sum_{i=1}^{e-1} d_i \ell^i}, \Delta_{e-2,1} = \Delta_{0,1}^{\ell^{e-2}} = (g^{\ell^{e-1}})^{d_1}$$

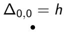
6. Determine d_1: if $\Delta_{e-2,1} == T[e-1][d]^{-1}$ then $d_1 \leftarrow d$

The PH algorithm

- For $k = 1, ..., e-1$, compute

$$\Delta_{0,k} = \Delta_{0,k-1} T[k-1][d_{k-1}] = g^{\sum_{i=k}^{e-1} d_i \ell^i}, \Delta_{e-1-k,k} = \Delta_{0,k}^{\ell^{e-1-k}} = (g^{\ell^{e-1}})^{d_k}$$

- Determine d_k: if $\Delta_{e-1-k,k} == T[e-1][d]^{-1}$ then $d_k \leftarrow d$

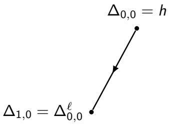


The PH algorithm

- For $k = 1, ..., e-1$, compute

$$\Delta_{0,k} = \Delta_{0,k-1} T[k-1][d_{k-1}] = g^{\sum_{i=k}^{e-1} d_i \ell^i}, \Delta_{e-1-k,k} = \Delta_{0,k}^{\ell^{e-1-k}} = (g^{\ell^{e-1}})^{d_k}$$

- Determine d_k: if $\Delta_{e-1-k,k} == T[e-1][d]^{-1}$ then $d_k \leftarrow d$

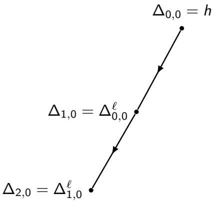


The PH algorithm

- For $k = 1, ..., e-1$, compute

$$\Delta_{0,k} = \Delta_{0,k-1} T[k-1][d_{k-1}] = g^{\sum_{i=k}^{e-1} d_i \ell^i}, \Delta_{e-1-k,k} = \Delta_{0,k}^{\ell^{e-1-k}} = (g^{\ell^{e-1}})^{d_k}$$

- Determine d_k: if $\Delta_{e-1-k,k} == T[e-1][d]^{-1}$ then $d_k \leftarrow d$

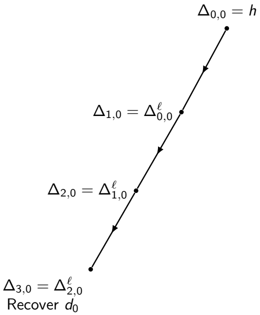


The PH algorithm

- For $k = 1, \ldots, e-1$, compute

$$\Delta_{0,k} = \Delta_{0,k-1} T[k-1][d_{k-1}] = g^{\sum_{i=k}^{e-1} d_i \ell^i}, \Delta_{e-1-k,k} = \Delta_{0,k}^{\ell^{e-1-k}} = (g^{\ell^{e-1}})^{d_k}$$

- Determine d_k: if $\Delta_{e-1-k,k} == T[e-1][d]^{-1}$ then $d_k \leftarrow d$

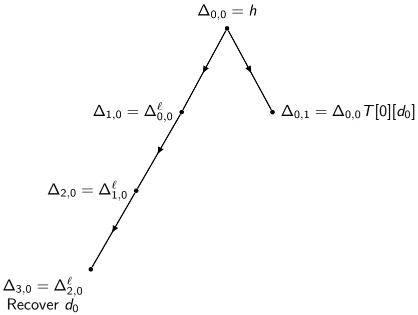


The PH algorithm

- For $k = 1, ..., e-1$, compute

$$\Delta_{0,k} = \Delta_{0,k-1} T[k-1][d_{k-1}] = g^{\sum_{i=k}^{e-1} d_i \ell^i}, \Delta_{e-1-k,k} = \Delta_{0,k}^{\ell^{e-1-k}} = (g^{\ell^{e-1}})^{d_k}$$

- Determine d_k: if $\Delta_{e-1-k,k} == T[e-1][d]^{-1}$ then $d_k \leftarrow d$

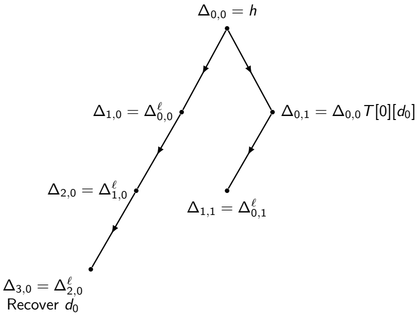


The PH algorithm

- For $k = 1, ..., e-1$, compute

$$\Delta_{0,k} = \Delta_{0,k-1} T[k-1][d_{k-1}] = g^{\sum_{i=k}^{e-1} d_i \ell^i}, \Delta_{e-1-k,k} = \Delta_{0,k}^{\ell^{e-1-k}} = (g^{\ell^{e-1}})^{d_k}$$

- Determine d_k: if $\Delta_{e-1-k,k} == T[e-1][d]^{-1}$ then $d_k \leftarrow d$

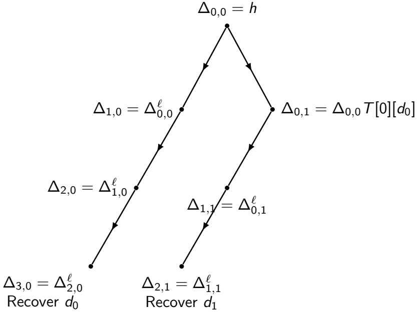


The PH algorithm

- For $k = 1, ..., e-1$, compute

$$\Delta_{0,k} = \Delta_{0,k-1} T[k-1][d_{k-1}] = g^{\sum_{i=k}^{e-1} d_i \ell^i}, \Delta_{e-1-k,k} = \Delta_{0,k}^{\ell^{e-1-k}} = (g^{\ell^{e-1}})^{d_k}$$

- Determine d_k: if $\Delta_{e-1-k,k} == T[e-1][d]^{-1}$ then $d_k \leftarrow d$

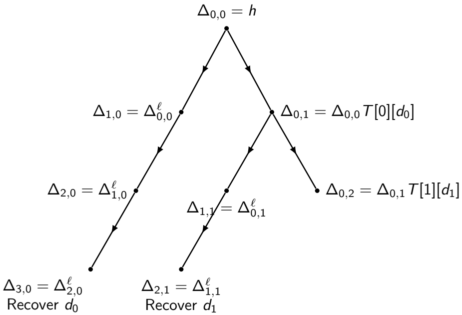


The PH algorithm

- For $k = 1, ..., e-1$, compute

$$\Delta_{0,k} = \Delta_{0,k-1} T[k-1][d_{k-1}] = g^{\sum_{i=k}^{e-1} d_i \ell^i}, \Delta_{e-1-k,k} = \Delta_{0,k}^{\ell^{e-1-k}} = (g^{\ell^{e-1}})^{d_k}$$

- Determine d_k: if $\Delta_{e-1-k,k} == T[e-1][d]^{-1}$ then $d_k \leftarrow d$

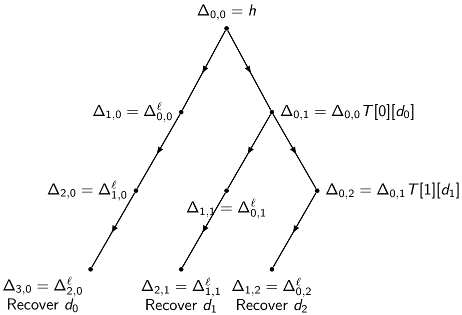


The PH algorithm

- For $k = 1, ..., e-1$, compute

$$\Delta_{0,k} = \Delta_{0,k-1} T[k-1][d_{k-1}] = g^{\sum_{i=k}^{e-1} d_i \ell^i}, \Delta_{e-1-k,k} = \Delta_{0,k}^{\ell^{e-1-k}} = (g^{\ell^{e-1}})^{d_k}$$

- Determine d_k: if $\Delta_{e-1-k,k} == T[e-1][d]^{-1}$ then $d_k \leftarrow d$

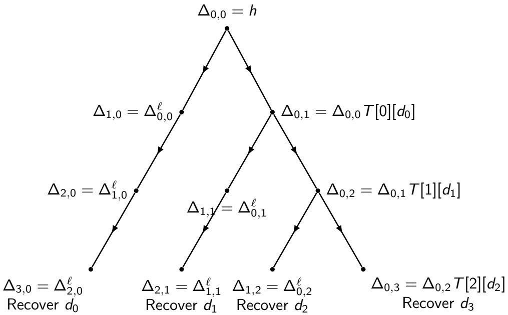


The PH algorithm

- For $k = 1, ..., e-1$, compute

$$\Delta_{0,k} = \Delta_{0,k-1} T[k-1][d_{k-1}] = g^{\sum_{i=k}^{e-1} d_i \ell^i}, \Delta_{e-1-k,k} = \Delta_{0,k}^{\ell^{e-1-k}} = (g^{\ell^{e-1}})^{d_k}$$

- Determine d_k: if $\Delta_{e-1-k,k} == T[e-1][d]^{-1}$ then $d_k \leftarrow d$
- **Cost:** $e(e-1)/2 = 6$ ℓ-exponentiations, $(e-1) = 3$ group multiplications, $e = 4$ table look ups
- **Storage:** $e\ell$ group elements in $\mathbb{F}_q^*$: $2e\ell \log p$ bits



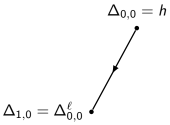
An Optimal PH algorithm

- Optimal trade-off between the multiplication and ℓ-exponentiation (Gustavo *et al.*, 2018):
- Among all possible traversing strategies, find one with minimum cost

$$\Delta_{0,0} = h$$

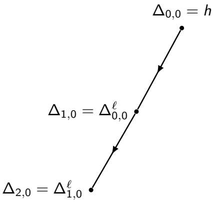
An Optimal PH algorithm

- Optimal trade-off between the multiplication and ℓ-exponentiation (Gustavo *et al.*, 2018):
- Among all possible traversing strategies, find one with minimum cost



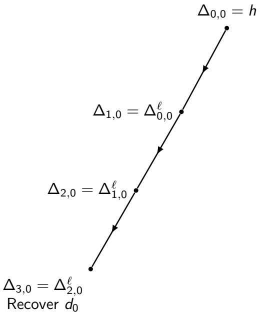
An Optimal PH algorithm

- Optimal trade-off between the multiplication and ℓ-exponentiation (Gustavo *et al.*, 2018):
- Among all possible traversing strategies, find one with minimum cost



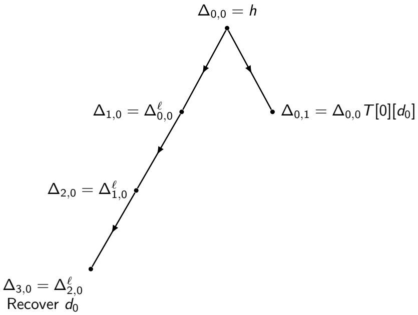
An Optimal PH algorithm

- Optimal trade-off between the multiplication and ℓ-exponentiation (Gustavo *et al.*, 2018):
- Among all possible traversing strategies, find one with minimum cost



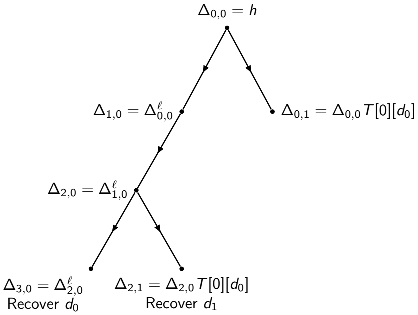
An Optimal PH algorithm

- Optimal trade-off between the multiplication and ℓ-exponentiation (Gustavo *et al.*, 2018):
- Among all possible traversing strategies, find one with minimum cost



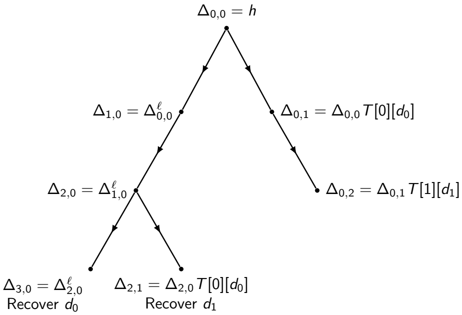
An Optimal PH algorithm

- Optimal trade-off between the multiplication and ℓ-exponentiation (Gustavo *et al.*, 2018):
- Among all possible traversing strategies, find one with minimum cost



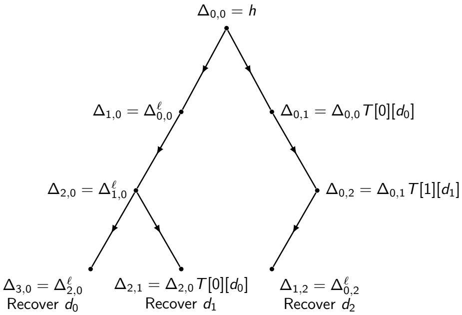
An Optimal PH algorithm

- Optimal trade-off between the multiplication and ℓ-exponentiation (Gustavo *et al.*, 2018):
- Among all possible traversing strategies, find one with minimum cost



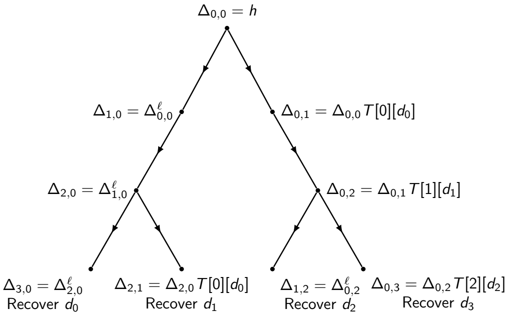
An Optimal PH algorithm

- Optimal trade-off between the multiplication and ℓ-exponentiation (Gustavo *et al.*, 2018):
- Among all possible traversing strategies, find one with minimum cost



An Optimal PH algorithm

- Optimal trade-off between the multiplication and ℓ-exponentiation (Gustavo *et al.*, 2018):
- Among all possible traversing strategies, find one with minimum cost
- **Cost:** 4 ℓ-exponentiations, 4 group multiplications, 4 table look ups
- **Storage:** $e\ell$ group elements in $\mathbb{F}_q^*$: $2e\ell \log p$ bits



Optimization 1: Signed-digits in the exponent

- $T[k][d] = g^{-d\ell^k},\ 0 \le k < e,\ 0 \le d < \ell$, stores $e\ell$ group elements
- Discard the entries with $d = 0$
- Write $d = \sum_{i=0}^{e-1} d_i' \ell^i$ for $d_i \in [-(\ell-1)/2, (\ell-1)/2]$ and modify T to T^{sgn} as

$$T^{\text{sgn}}[k][d] = g^{-d\ell^k},\ 0 \le k < e,\ \boxed{0 \le d \le (\ell-1)/2}$$

- Revise the step to determine d_k as
  1. If $\Delta_{e-1-k,k} == T^{\text{sgn}}[e-1][d]^{-1}$ then $d_k' \leftarrow d$
  2. If $\Delta_{e-1-k,k} == T^{\text{sgn}}[e-1][d]$ then $d_k' \leftarrow d - \ell$
- Updating $\Delta_{j,k+1} \leftarrow \Delta_{j,k} T[j+k][d_k]$ is revised similarly:
  1. If $d_k' > 0$, then $\Delta_{j,k+1} \leftarrow \Delta_{j,k} T^{\text{sgn}}[j+k][d_k']$
  2. If $d_k' < 0$, then $\Delta_{j,k+1} \leftarrow \Delta_{j,k} T^{\text{sgn}}[j+k][-d_k']^{-1}$

Remark

The table size is reduced by a factor of 2.

Optimization 2: Torus representations and arithmetic in $\mathbb{G}$

- $\mathbb{G} = \{a + bi : \ a, b \in \mathbb{F}_p, \ a^2 + b^2 = 1\}$ is the order-$(p+1)$ cyclotomic subgroup of ${\mathbb{F}_{p^2}}^*$

- Torus representation of $\mathbb{G}$ yields (Rubin and Silverberg, 2008):

$$\mathbb{G} = \{1\} \cup \left\{ \frac{\alpha + i}{\alpha - i} : \ \alpha \in \mathbb{F}_p \right\},$$

  where $a + bi = (\alpha + i)/(\alpha - i)$ with $\alpha = (a+1)/b$

- This compressed representation preserves group multiplication

- For $C : \mathbb{G} \setminus \{1, -1\} \to \mathbb{F}_p$ with $C(a + bi) = (a+1)/b$, we have

$$C(a+bi) = \alpha, C(c+di) = \beta, \alpha+\beta \neq 0 : \quad \boxed{C((a+bi)(c+di)) = \frac{\alpha\beta - 1}{\alpha + \beta}}$$

Torus representations and arithmetic

- Using projective coordinates $[x:y] := (x+yi)/(x-yi)$, we can write

$$\mathbb{G} = \{a+bi: \ a,b \in \mathbb{F}_{p^2}\} = \{[x:y]: \ x,y \in \mathbb{F}_p, xy \neq 0\},$$

where $a+bi \mapsto [a+1,b]$, $[x:y] \mapsto \frac{x^2-y^2}{x^2+y^2} + \frac{2xy}{x^2+y^2}i$

Proj. Squaring (**2m**):	$[x:y]^2 = [(x+y)(x-y) : 2xy]$
Proj. Cubing (**2m+2s**):	$[x:y]^3 = [x(x^2-3y^2) : y(3x^2-y^2)]$
Proj. Mul. (**3m**):	$[x:y][z:t] = [xz-yt : (x+y)(z+t)-xz-yt]$
Mixed Mul. (**2m**):	$[x:y][\alpha,1] = [x\alpha - y : x + y\alpha]$
Inversion (**0m**):	$[x:y]^{-1} = [-x:y]$
Proj. Equality check (**2m**):	$[x:y] = [z:t] \iff xt - yz = 0$
Mixed Equality check (**1m**):	$[x:y] = [\alpha:1] \iff x - y\alpha = 0$

Remark

Traditional costs: squaring (**2s**), and cubing (**2m+1s**), group multiplication (**3m**), and free equality check

Table compression via torus representation

- Use $C(a+bi) = (a+1)/b$ to further compress T^{sgn} by a factor of 2:

$$CT[k][d] = C(T^{\text{sgn}}[k][d]) = C(g^{-d\ell^k})$$

- This yields an overall compression of tables by a factor of 4
- Computational overheads:
  - Right traversals $\Delta_{j,k+1} \leftarrow \Delta_{j,k} \cdot CT[j+k][d_k]$ become *mixed multiplications*

    $\ell = 2,3$: The cost **3m** changes to **2m**

  - Left traversals $\Delta_{j+1,k} \leftarrow \Delta_{j,k}^{\ell}$ become *projective exponentiations*

    $\ell = 2$: The cost **2s** changes to **2m**

    $\ell = 3$: The cost **2m+1s** changes to **2m+2s**

  - Table look ups to determine d_k via

    $$\Delta_{e-1-k,k} == T[e-1][d]^{-1} \Rightarrow d_k \leftarrow d$$

    used to be free but now requires $(\ell-1)/2$ multiplication at each leaf, and there are $(e-1)$ leaves

Computational overheads and our proposal

- Computational overheads for determining d_k using torus-based representations become non-trivial if PH with width-ω windows is used:

$$d = \sum_{i=0}^{e-1} d_i \ell^i, \; d_i \in [0, \ell)$$
$$= \sum_{i=0}^{m-1} D_i L^i, \; m = \lceil e/\omega \rceil, \; L = \ell^\omega, \; D_i \in [0, L)$$

$$\Delta_{m-1-k,k} == T[m-1][D]^{-1} \; \Rightarrow \; D_k \leftarrow D$$

- Now, it requires $(L-1)/2$ multiplications per leaf

Our proposal

Instead of going through $(L-1)/2$ equality checks, solve for the logarithm D_k of $\Delta_{m-1-k,k}$ base $\rho = g^{L^{m-1}}$, which generates $\mathbb{G}_{\ell,\omega}$

$$\Delta_{m-1-k,k} = T[m-1][D_k]^{-1} = (g^{L^{m-1}})^{D_k}$$

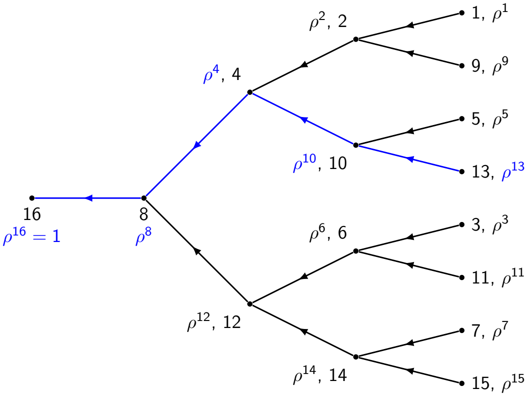
Discrete logarithms in $\mathbb{G}_{\ell,\omega}$

- Let $\mathbb{G}_{\ell,\omega} = \langle \rho \rangle$ be a cyclic group of order $L = \ell^\omega$, with torus-based representations
- Solving DLP with table look ups requires approximately $\ell^\omega/2$ $\mathbb{F}_p$-elements to store and $\ell^\omega/2$ equality checks, equivalently $\ell^\omega/2$ $\mathbb{F}_p$-multiplications (m)
- We propose new algorithms to solve DLP, with (mostly) linear complexity in ω

	Restriction	Average Complexity	Storage
Alg. 1	$\ell = 2$	$\left(\frac{7}{2}\omega - 4 + \frac{4}{2^\omega}\right)$ m	$2^\omega + \omega - 2$
Alg. 2	$\ell = 2$	$\left(3\omega - 4 + \frac{1}{2^{\omega-2}}\right)$ m	$2^{\omega-2} + \omega + 2$
Alg. 4	$\ell = 2$	$\left(2^{\omega-3} - \frac{1}{2}\right)$ m	$2^{\omega-1} + 1$
Alg. 1	$\ell = 3$	$\left(\frac{28}{5}\omega - \frac{33}{10} + \frac{33}{10\cdot 3^\omega}\right)$ m	$3^\omega + \omega - 2$
Alg. 3	$\ell = 3$	$\left(\frac{79}{15}\omega - \frac{33}{10} + \frac{33}{10\cdot 3^\omega}\right)$ m	$3^{\omega-1} + \omega - 1$

New algorithms to solve DLP in $\mathbb{G}_{\ell,\omega}$

- **Main ideas:**
  1. Construct an ℓ-ary like graph $\mathcal{G}_{\ell,\omega}$ of depth-ω such that an ℓ-exponentiation in $\mathbb{G}_{\ell,\omega}$ corresponds to traversing an edge in $\mathbb{G}_{\ell,\omega}$
  2. Given $h \in \mathbb{G}_{\ell,\omega}$, ℓ-exponentiate h until the result is identity. Extract the unique path in $\mathcal{G}_{\ell,\omega}$, so that the starting vertex yields the logarithm of h base ρ
- Example for $\ell = 2$, $\omega = 4$: $\mathbb{G}_{\ell,\omega} = \langle \rho \rangle$, $h = \rho^{13}$



Theoretical results

Theorem

Let $h \in \mathbb{G}_{\ell,\omega}$ be an element of order ℓ^k for some arbitrary $k \in \{1, \ldots, \omega\}$. Define a sequence $H = [h_0, \ldots, h_k]$ such that $h_k = h$ and $h_{j-1} = h_j^{\ell}$ for $j = 1, \ldots, k$. Then, there exists a unique path $P_{0,k} = v_{0,0}, v_{1,i_1}, \ldots, v_{k,i_k}$ in $\mathcal{G}_{\ell,\omega}$ such that $v_{j,i_j} \in V_j$ and $h_j = g_{j,i_j}$ for $j = 0, \ldots, k$.

Theorem

Let $1 \neq h \in \mathbb{G}_{\ell,\omega} = \langle \rho \rangle$. Given h and ρ, one can determine

1. *k, $i_1, i_2, \ldots, i_k$, that corresponds to the path $P_{0,k} = v_{0,0}, v_{1,i_1}, \ldots, v_{j,i_k}$ as in the above theorem;*
2. *$s_1, s_2, \ldots, s_k$ such that , $s_1 = i_1 + 1$, $s_j \in \{0, \ldots, (\ell - 1)\}$, and $i_j = \ell \cdot i_{j-1} + s_k$ for $j = 2, \ldots, k$.*

Moreover, $h = \rho^d$, where $d = \ell^{\omega-k} \sum_{j=1}^{k} s_j \ell^{j-1}$.

Refinements and other algorithms

1. Half of the paths in $\mathcal{G}_{\ell,\omega}$ can be eliminated at a cost of inverting elements in $\mathbb{G}_{\ell,\omega}$
2. We propose another algorithm for $\ell = 2$: exploit algebraic relations induced by the order of points in $\mathbb{G}_{\ell,\omega}$ and recursively enumerate them
3. Run time is exponential in ω but performs better than the previous one for small ω
4. Best of both worlds: a hybrid of the two algorithms

Parameters	Average Complexity
if $\omega_2 \leq 2$ and $\omega_2 \leq \omega_1$	$3\omega_2 + 2^{\omega_1 - 3} - \frac{1}{2} - \frac{\omega_2+2}{2^{\omega_1}} + \frac{2}{2^{\omega}}$
if $2 < \omega_2$ and $\omega_2 \leq \omega_1$	$3\omega_2 + 2^{\omega_1 - 3} - \frac{1}{2} + \frac{2^{\omega_2 - 3} - \omega_2 - \frac{5}{2}}{2^{\omega_1}} + \frac{2}{2^{\omega}}$
if $\omega_1 = 2$ and $2 < \omega_2$	$3\omega_2 - \frac{5}{4} + \frac{6}{2^{\omega}}$
if $2 < \omega_1$ and $\omega_1 < \omega_2$	$3\omega_2 + 2^{\omega_1 - 3} - 2^{\omega_1 - \omega_2 - 4} - \frac{5}{16} - \frac{\omega_1 + \frac{7}{2}}{2^{\omega_1}} + \frac{1}{2^{\omega_2}} + \frac{2}{2^{\omega}}$

Implementation results and comparisons

- $\ell = 2$ case enjoys the full factor-4 compression and slight speed-ups
- $\ell = 3$ case enjoys factor-2 compression but factor-4 compression suffers from a computational overhead: up to 9% in key generation
- A reasonable choice:
  - For $\ell = 2$, use signed-digits and torus representations (factor-4 compression)
  - For $\ell = 3$, use only signed-digits (factor-2 compression)
- New table sizes:
  - $\ell = 2$: 18KiB to 240KiB for all SIKE parameter sets
  - $\ell = 3$: 34KiB to 477KiB for all SIKE parameter sets

Implementation results and comparisons

- Comparative results of the average cost (in $\mathbb{F}_p$-multiplications) and table sizes (in KiB) to compute logarithms in $\mathbb{G}_{2,e}$ and $\mathbb{G}_{3,e}$
- Our results have been implemented in C and available at `https://github.com/microsoft/PQCrypto-SIDH`

DLP in $\mathbb{G}_{2,e}$	source	$\omega = 3$		$\omega = 4$		$\omega = 5$	
		time	size	time	size	time	size
$e = 216$ (SIKEp434)	previous	1944	70	1600	105	1415	342
	ours	1818	18	1542	27	1408	86
$e = 372$ (SIKEp751)	previous	3765	197	3126	296	2748	954
	ours	3476	49	2964	74	2688	240

DLP in $\mathbb{G}_{3,e}$	source	$\omega = 2$		$\omega = 3$		$\omega = 4$	
		time	size	time	size	time	size
$e = 137$ (SIKEp434)	previous	1845	151	1407	301	1185	688
	ours	2371	34	2029	73	1868	172
$e = 239$ (SIKEp751)	previous	3621	429	2793	859	2337	1932
	ours	4542	95	3886	207	3507	477

Thank You