Side-Channel Protections for CSIDH

Francisco Rodríguez-Henríquez.

Joint work with:
Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo and Benjamin Smith

October 9, 2019

Timeline of CSIDH

1996

Before CSIDH (ordinary curves): CRS scheme

- Couveignes first unpublished ideas;

Timeline of CSIDH

2006

Before CSIDH (ordinary curves):CRS scheme

- Rostovtsev and Stolbunov [Rostovtsev and Stolbunov, 2006];
- Couveignes [Couveignes, 2006];

Timeline of CSIDH

2006

Before CSIDH (ordinary curves): CRS scheme

- Rostovtsev and Stolbunov [Rostovtsev and Stolbunov, 2006];
- Couveignes [Couveignes, 2006];
- Stolbunov [Stolbunov, 2010];

Timeline of CSIDH

2010

Before CSIDH (ordinary curves):CRS scheme

- Rostovtsev and Stolbunov [Rostovtsev and Stolbunov, 2006];
- Couveignes [Couveignes, 2006];
- Stolbunov [Stolbunov, 2010];
- Childs, Jao and Soukharev [Childs et al., 2010];

Timeline of CSIDH

2018

Before CSIDH (ordinary curves):CRS scheme

- Rostovtsev and Stolbunov [Rostovtsev and Stolbunov, 2006];
- Couveignes [Couveignes, 2006];
- Stolbunov [Stolbunov, 2010];
- Childs, Jao and Soukharev [Childs et al., 2010];
- De Feo, Kieffer, and Smith [De Feo et al., 2018];

Timeline of CSIDH

2018

CSIDH (supersingular curves):

- April: Castryck, Lange, Martindale, Panny, and Renes proposed CSIDH [Castryck et al., 2018];

Timeline of CSIDH

2018

CSIDH (supersingular curves):

- April: Castryck, Lange, Martindale, Panny, and Renes proposed CSIDH [Castryck et al., 2018];
- August: Meyer and Reith [Meyer and Reith, 2018];

Timeline of CSIDH

2018

CSIDH (supersingular curves):

- April: Castryck, Lange, Martindale, Panny, and Renes proposed CSIDH [Castryck et al., 2018];
- August: Meyer and Reith [Meyer and Reith, 2018];
- Constant-time implementations:

Timeline of CSIDH

2018

CSIDH (supersingular curves):

- April: Castryck, Lange, Martindale, Panny, and Renes proposed CSIDH [Castryck et al., 2018];
- August: Meyer and Reith [Meyer and Reith, 2018];
- Constant-time implementations:
    - August'18: Jalali *et al.* [Jalali et al., 2019];

Timeline of CSIDH

2018

CSIDH (supersingular curves):

- April: Castryck, Lange, Martindale, Panny, and Renes proposed CSIDH [Castryck et al., 2018];
- August: Meyer and Reith [Meyer and Reith, 2018];
- Constant-time implementations:
    - August'18: Jalali *et al.* [Jalali et al., 2019];
    - October'18: Bernstein, Lange, Martindale, and Panny [Bernstein et al., 2019];

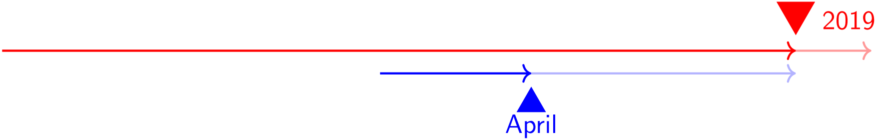
Timeline of CSIDH

2018

CSIDH (supersingular curves):

- April: Castryck, Lange, Martindale, Panny, and Renes proposed CSIDH [Castryck et al., 2018];
- August: Meyer and Reith [Meyer and Reith, 2018];
- Constant-time implementations:
  - August'18: Jalali *et al.* [Jalali et al., 2019];
  - October'18: Bernstein, Lange, Martindale, and Panny [Bernstein et al., 2019];
  - December'18: Meyer, Campos, and Reith [Meyer et al., 2019];

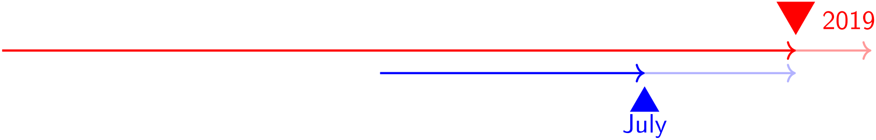
Timeline of CSIDH



CSIDH (supersingular curves):

- April: Castryck, Lange, Martindale, Panny, and Renes proposed CSIDH [Castryck et al., 2018];
- August: Meyer and Reith [Meyer and Reith, 2018];
- Constant-time implementations:
  - August'18: Jalali *et al.* [Jalali et al., 2019];
  - October'18: Bernstein, Lange, Martindale, and Panny [Bernstein et al., 2019];
  - December'18: Meyer, Campos, and Reith [Meyer et al., 2019];
  - April'19: Onuki, Aikawa, Yamazaki, and Takagi [Onuki et al., 2019];

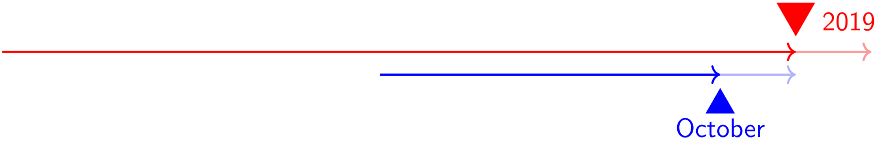
Timeline of CSIDH



CSIDH (supersingular curves):

- April: Castryck, Lange, Martindale, Panny, and Renes proposed CSIDH [Castryck et al., 2018];
- August: Meyer and Reith [Meyer and Reith, 2018];
- Constant-time implementations:
  - August'18: Jalali *et al.* [Jalali et al., 2019];
  - October'18: Bernstein, Lange, Martindale, and Panny [Bernstein et al., 2019];
  - December'18: Meyer, Campos, and Reith [Meyer et al., 2019];
  - April'19: Onuki, Aikawa, Yamazaki, and Takagi [Onuki et al., 2019];
  - July'19: Cervantes-Vázquez, Chenu, Chi-Domínguez, de Feo, RH and Smith [Cervantes-Vázquez et al., 2019]

Timeline of CSIDH



CSIDH (supersingular curves):

- April: Castryck, Lange, Martindale, Panny, and Renes proposed CSIDH [Castryck et al., 2018];
- August: Meyer and Reith [Meyer and Reith, 2018];
- Constant-time implementations:
  - August'18: Jalali *et al.* [Jalali et al., 2019];
  - October'18: Bernstein, Lange, Martindale, and Panny [Bernstein et al., 2019];
  - December'18: Meyer, Campos, and Reith [Meyer et al., 2019];
  - April'19: Onuki, Aikawa, Yamazaki, and Takagi [Onuki et al., 2019];
  - July'19: Cervantes-Vázquez, Chenu, Chi-Domínguez, de Feo, RH and Smith [Cervantes-Vázquez et al., 2019]
  - October'19: Hutchinson, LeGrow, Koziel and Azarderakhsh [Hutchinson et al., 2019]

CSIDH overview

The action $\mathfrak{a} * E_A$ defines a path on the isogeny graph over $\mathbb{F}_p$, and is determined by an integer vector $(e_1, \ldots, e_n) \in [\![-m, m]\!]^n$:

1) Nodes are supersingular elliptic curves over $\mathbb{F}_p$ in Montgomery form;
2) Edges are degree-ℓ_i isogenies.

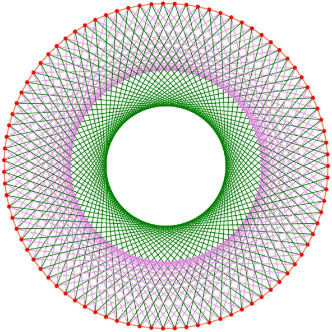


Figure 1: Isogeny graph over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. Nodes are supersingular curves and edges marked with orange, green , and violet inks denote isogenies of degree 5, 13 and 61, respectively.

CSIDH overview

The action $\mathfrak{a} * E_A$ defines a path on the isogeny graph over $\mathbb{F}_p$, and is determined by an integer vector $(e_1, \ldots, e_n) \in [\![-m, m]\!]^n$:

1) Nodes are supersingular elliptic curves over $\mathbb{F}_p$ in Montgomery form;
2) Edges are degree-ℓ_i isogenies. Two types of edges: isogeny with kernel generated by
   - 2.a) $(x, y) \in E_A[\ell_i, \pi - 1]$, or
   - 2.b) $(x, iy) \in E_A[\ell_i, \pi + 1]$.

Here, $x, y \in \mathbb{F}_p$, $\pi\colon (X, Y) \mapsto (X^p, Y^p)$ is the Frobenius map, $i = \sqrt{-1}$ and thus $i^p = -i$.

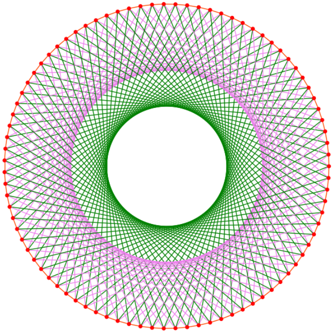


Figure 1: Isogeny graph over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. Nodes are supersingular curves and edges marked with orange, green , and violet inks denote isogenies of degree 5, 13 and 61, respectively.

CSIDH overview

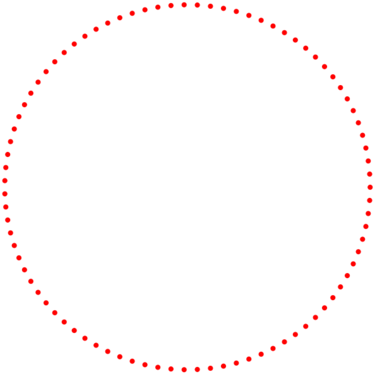


Figure 2: Action evaluation over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. Secret integer vector $(-1, 2, 1) \in [\![-2, 2]\!]^3$:

E_0

CSIDH overview

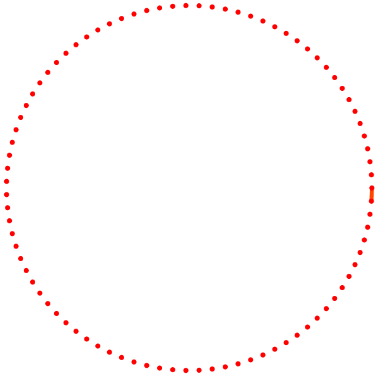


Figure 2: Action evaluation over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. Secret integer vector $(-1, 2, 1) \in [\![-2, 2]\!]^3$:

$$E_0 \longrightarrow E_{0x3A7D}$$

CSIDH overview

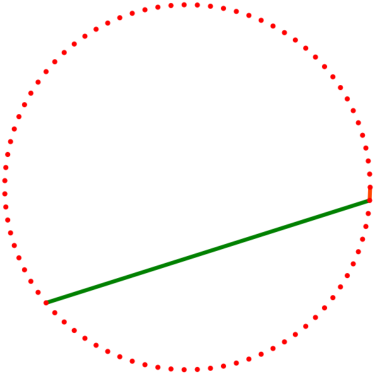


Figure 2: Action evaluation over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. Secret integer vector $(-1, 2, 1) \in [\![-2, 2]\!]^3$:

$$E_0 \longrightarrow E_{0x3A7D} \longrightarrow E_{0x2BF7}$$

CSIDH overview

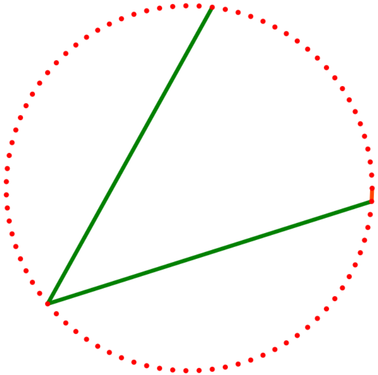


Figure 2: Action evaluation over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. Secret integer vector $(-1, 2, 1) \in [\![-2, 2]\!]^3$:

$$E_0 \longrightarrow E_{0x3A7D} \longrightarrow E_{0x2BF7} \longrightarrow E_{0x1404}$$

CSIDH overview

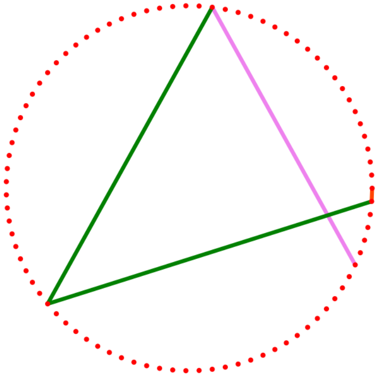


Figure 2: Action evaluation over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. Secret integer vector $(-1, 2, 1) \in [\![-2, 2]\!]^3$:

$$E_0 \longrightarrow E_{0x3A7D} \longrightarrow E_{0x2BF7} \longrightarrow E_{0x1404} \longrightarrow E_{0x5EB}$$

CSIDH overview

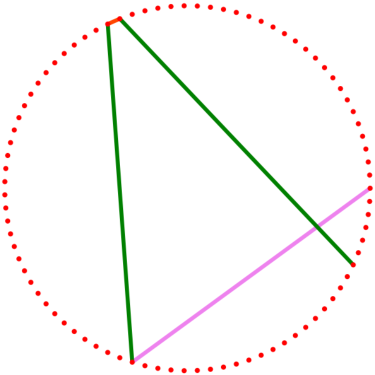


Figure 2: Action evaluation over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. In general, the action evaluation is *commutative*. Secret integer vector $(-1, 2, 1) \in [\![-2, 2]\!]^3$:

$$E_0 \longrightarrow E_{\mathrm{0x7A0}} \longrightarrow E_{\mathrm{0x8EC}} \longrightarrow E_{\mathrm{0x25B3}} \longrightarrow E_{\mathrm{0x5EB}}$$

CSIDH overview

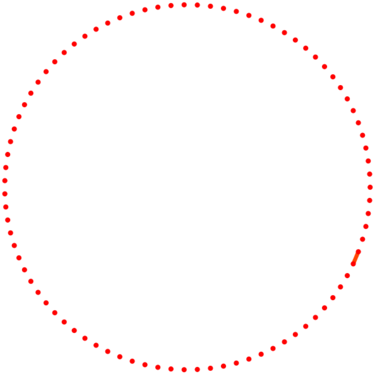


Figure 2: Action evaluation over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. Secret integer vector $(1, -2, -1) \in [\![-2, 2]\!]^3$ has *inverse* $(-1, 2, 1) \in [\![-2, 2]\!]^3$:

$$E_{0x5EB} \rightarrow E_{0x1D50} \rightarrow E_{0x8EC} \rightarrow E_{0x56D} \rightarrow E_0$$

CSIDH overview

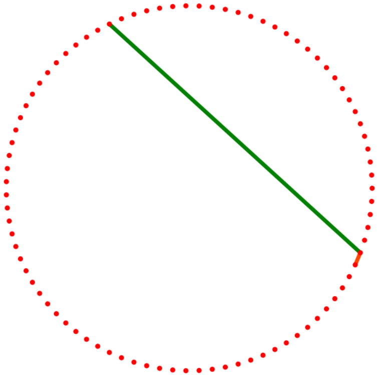


Figure 2: Action evaluation over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. Secret integer vector $(1, -2, -1) \in [\![-2, 2]\!]^3$ has *inverse* $(-1, 2, 1) \in [\![-2, 2]\!]^3$:

$$E_{0x5EB} \rightarrow E_{0x1D50} \rightarrow E_{0x8EC} \rightarrow E_{0x56D} \rightarrow E_0$$

CSIDH overview

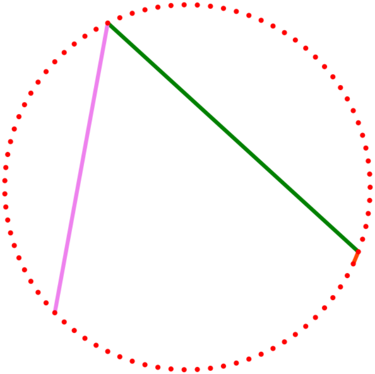


Figure 2: Action evaluation over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. Secret integer vector $(1, -2, -1) \in [\![-2, 2]\!]^3$ has *inverse* $(-1, 2, 1) \in [\![-2, 2]\!]^3$:

$$E_{0x5EB} \rightarrow E_{0x1D50} \rightarrow E_{0x8EC} \rightarrow E_{0x56D} \rightarrow E_0$$

CSIDH overview

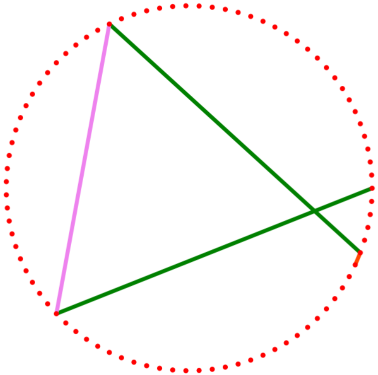


Figure 2: Action evaluation over $\mathbb{F}_p$ with $p = 4 \cdot (5 \cdot 13 \cdot 61) - 1$. Secret integer vector $(1, -2, -1) \in [\![-2, 2]\!]^3$ has *inverse* $(-1, 2, 1) \in [\![-2, 2]\!]^3$:

$$E_{0x5EB} \rightarrow E_{0x1D50} \rightarrow E_{0x8EC} \rightarrow E_{0x56D} \rightarrow E_0$$

CSIDH overview

CSIDH framework [Castryck et al., 2018].
Parameters:

- Small odd primes numbers ℓ_i such that $p = 4\prod_{i=1}^{n} \ell_i - 1$ is a prime number.
  Note: The original CSIDH article [Castryck et al., 2018] defined a 511-bit p with $\ell_1, \ldots, \ell_{n-1}$ the first 73 odd primes, and $\ell_n = 587$.
- A supersingular elliptic curve in Montgomery form $E_A/\mathbb{F}_p \colon y^2 = x^3 + Ax^2 + x$ with $\#E(\mathbb{F}_p) = p + 1$;

General description CSIDH:

The shared secret key is $(\mathfrak{a} \cdot \mathfrak{b}) * E_A$.

The security is given by the hardness of computing $\mathfrak{a}$ (or $\mathfrak{b}$) given the data colored in red ink.

$$\begin{tikzcd}
E_A \arrow[r, "\mathfrak{a}"] \arrow[d, "\mathfrak{b}"'] & \mathfrak{a} * E_A \arrow[d, "\mathfrak{b}"] \arrow[dl, dashed, shift right, "\mathfrak{a} * E_A"'] \\
\mathfrak{b} * E_A \arrow[r, "\mathfrak{a}"'] \arrow[ur, dashed, shift right, "\mathfrak{b} * E_A"'] & (\mathfrak{a} \cdot \mathfrak{b}) * E_A
\end{tikzcd}$$

CSIDH overview

CSIDH framework [Castryck et al., 2018].
Public/private keys:

- Alice private key: The set of small integer exponents $\{e_1, e_1, \ldots, e_n\}$, with $e_i \in [-m, m]$, Alice public key: $\mathfrak{a} * E_A$
- Bob private key: The set of small integer exponents $\{f_1, f_1, \ldots, f_n\}$, with $f_i \in [-m, m]$, Bob public key: $\mathfrak{b} * E_A$

Note: The private key space size is $(2m+1)^n$. Choosing $m = 5$ implies, $(2 \cdot 5 + 1)^{74} \approx 2^{256}$.

General description CSIDH:

The shared secret key is $(\mathfrak{a} \cdot \mathfrak{b}) * E_A$.

The security is given by the hardness of computing $\mathfrak{a}$ (or $\mathfrak{b}$) given the data colored in red ink.

$$\begin{tikzcd}
E_A \arrow[r, "\mathfrak{a}"] \arrow[d, "\mathfrak{b}"'] & \mathfrak{a} * E_A \arrow[d, "\mathfrak{b}"] \arrow[dl, dashed, bend right=5, "\mathfrak{a} * E_A"'] \\
\mathfrak{b} * E_A \arrow[r, "\mathfrak{a}"'] \arrow[ur, dashed, bend right=5, "\mathfrak{b} * E_A"'] & (\mathfrak{a} \cdot \mathfrak{b}) * E_A
\end{tikzcd}$$

Each ℓ_i is required e_i times for evaluating the *action* $\mathfrak{a} * E_A$ (similarly for $\mathfrak{b} * E_A$). Formally, this is written as $\mathfrak{a} = \mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}$.

CSIDH overview

CSIDH Hard problem [Castryck et al., 2018]:
Given two supersingular elliptic curves E, E' defined over $\mathbb{F}_p$ with the same $\mathbb{F}_p$-rational endomorphism ring, find an ideal $\mathfrak{a}$ such that, $\mathfrak{a} * E = E'$.

CSIDH overview

CSIDH Hard problem [Castryck et al., 2018]:
Given two supersingular elliptic curves E, E' defined over $\mathbb{F}_p$ with the same $\mathbb{F}_p$-rational endomorphism ring, find an ideal $\mathfrak{a}$ such that, $\mathfrak{a} * E = E'$.

- Classical security:
  - Brute force: The private key space size is $(2m+1)^n = (2 \cdot 5 + 1)^{74} \approx 2^{256}$.
  - Meet-in-the-middle: It has an associated complexity of $O(p^{1/4})$ [Adj et al., 2019]
- Quantum security:
  - Grover and claw finding: [Idealized] associated time complexity of $O(p^{1/6})$
  - Abelian hidden-shift problem: Estimated in 2^{40} quantum operations on a quantum computer of 2^{40} qubits to compute a single evaluation of the Kuperberg or Regev oracle with failure probability less than 2^{-32}.[Bernstein et al., 2019]
    Note: This claim is currently under dispute.
    See [Peikert, 2019, Bonnetain and Schrottenloher, 2018]

Pros and cons of CSIDH

- Advantages of CSIDH:
  - Key sizes: With the goal of providing a security level of λ-bits, CRS, SIDH, SIKE and CSIDH all choose primes $p \approx 2^{4\lambda}$. CSIDH keys are size $n \approx 4\lambda$, whereas SIDH/SIKE require $n' \approx 14\lambda - 24\lambda$ bits. This size is by far the smallest of all post-quantum cryptographic schemes.
  - NIKE: The CSIDH group action allows efficient public-key validation. Hence CSIDH supports a non-interactive (static-static) key exchange. This is a unique feature among all post-quantum cryptographic schemes.
  - Protocol bandwith: Combining the two features mentioned above, CSIDH can compute a shared secret exchanging only $n \approx 4\lambda$ bits per party.

Pros and cons of CSIDH

- Disadvantages of CSIDH:
  - Latency: Running on a high-end x64 Intel processor, CSIDH requires $\approx 480M$ clock cycles to compute a shared secret. For comparison, SIKE requires $\approx 20M$ clock cycles.
  - Quantum security: Quantum security claims of CSIDH are Currently under [much] dispute

CSIDH implementations

- [Castryck et al., 2018]: The original CSIDH. The authors define the curve and isogeny arithmetic on Montgomery curves;
- [Meyer and Reith, 2018]: Proposed a hybrid CSIDH using isogeny construction formulas defined on Twisted Edwards curves and then mapping into Montgomery form;
- [Bernstein et al., 2019]: Evaluated a constant-time implementation of CSIDH aiming to precisely assess its quantum security. In this work it was proposed and implemented the usage of SIDH strategies à la SIDH for CSIDH, (cf. [Bernstein et al., 2019, §8]). The authors also proposed a primitive version of the SIMBA strategy
- [Meyer et al., 2019], and [Onuki et al., 2019]: Both works kept using a hybrid CSIDH as in [Meyer and Reith, 2018]. They represent the current state-of-the-art in constant time implementations of CSIDH

CSIDH main building blocks

- Elliptic curve arithmetic:
  - Point addition and point doubling costs: $4M + 2S + 4A$ and $4M + 2S + 6A$ field operations, respectively.
  - Scalar multiplication kP cost: $\approx 1.5 \log_2(k)(4M + 2S + 6A)$ field operations.

Implementation note: Point addition costs virtually the same as a point doubling. Using a PRAC-like Montgomery ladder one can save about 25% of the cost of a classical Montgomery ladder

CSIDH main building blocks

- Elliptic curve arithmetic:
  - Point addition and point doubling costs: $4M + 2S + 4A$ and $4M + 2S + 6A$ field operations, respectively.
  - Scalar multiplication kP cost: $\approx 1.5 \log_2(k)(4M + 2S + 6A)$ field operations.

  Implementation note: Point addition costs virtually the same as a point doubling. Using a PRAC-like Montgomery ladder one can save about 25% of the cost of a classical Montgomery ladder

- Isogeny arithmetic:
  - Evaluation of a degree $\ell = 2k + 1$ isogeny: $4kM + 2S + (2k + 4)A$ field operations
  - Construction of a degree $\ell = 2k + 1$ isogeny: $\approx (8k - 7)M + (2k + 15)S + (2)A$ field operations

  Implementation note: One isogeny construction costs ≈ 2.6 isogeny evaluations.

Original CSIDH [Castryck et al., 2018]

Algorithm 1 Original CSIDH

Require: $A \in \mathbb{F}_p$ such that $E_A\colon\ y^2 = x^3 + Ax^2 + x$ is supersingular, and an integer exponent vector $(e_1, \ldots, e_n)$

Ensure: B such that $E_B : y^2 = x^3 + Bx^2 + x$ is $\mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$, $B \leftarrow A$

while some $e_i \neq 0$ **do**
    Sample a random $x \in \mathbb{F}_p$
    $s \leftarrow +1$ if $x^3 + Bx^2 + x$ is square in $\mathbb{F}_p$, else $s \leftarrow -1$
    $S \leftarrow \{i \mid e_i \neq 0, \mathrm{sign}(e_i) = s\}$
    if $S \neq \emptyset$ **then**
        $k \leftarrow \prod_{i \in S} \ell_i$
        $Q \leftarrow [(p+1)/k]P$, where P is the projective point with x-coordinate x.
        for $i \in S$ **do**
            $R \leftarrow [k/\ell_i]Q$ // Point to be used as kernel generator
            if $R \neq \infty$ **then**
                $(E_B, \phi) \leftarrow \texttt{QuotientIsogeny}(E_B, R)$
                $Q \leftarrow \phi(Q)$
                $(k, e_i) \leftarrow (k/\ell_i, e_i - s)$
            end if
        end for
    end if
end while
return B

Original CSIDH algorithm [Castryck et al., 2018]: Security problems

- Variable time: The private key determines the running time of the algorithm.
  - The worst case running time occurs for the private key: $(5, 5, \ldots, 5)$
  - No computation occurs for the private key: $(0, 0, \ldots, 0)$
  - The private key, $(5, -5, 5, \ldots, -5, 5)$ is processed 50% faster than the private key, $(5, 5, \ldots, 5)$
- Power analysis: The attacker can determine which private key elements share the same sign

Note: The CSIDH algorithm [Castryck et al., 2018] *implicitly* uses a two-point strategy

Constant-time CSIDH algorithm by [Meyer et al., 2019]

- Strategies towards a constant-time implementation of CSIDH
    - To move the range of the exponents from $[-5, 5]$ to $[0, 10]$. This allows to use only one point through the isogeny computations
    - To compute and evaluate *exactly* 10 isogenies per ℓ_i prime, using dummy isogenies if required
- Efficiency improvements
    - To use the Elligator 2 map to sample the points to be used in the main computation
    - To split isogeny computations into multiple batches (SIMBA approach)

      Note: it's useful to think the SIMBA approach as one form of a strategy à la SIDH.

Constant-time CSIDH algorithm by [Meyer et al., 2019]

- Strategies towards a constant-time implementation of CSIDH
  - To move the range of the exponents from $[-5, 5]$ to $[0, 10]$. This allows to use only one point through the isogeny computations
  - To compute and evaluate *exactly* 10 isogenies per ℓ_i prime, using dummy isogenies if required
- Efficiency improvements
  - To use the Elligator 2 map to sample the points to be used in the main computation
  - To split isogeny computations into multiple batches (SIMBA approach)

    Note: it's useful to think the SIMBA approach as one form of a strategy à la SIDH.
  - To use customized bounds m_i for each one of the seventy-four e_i exponents

[Simplified] CSIDH algorithm by [Meyer et al., 2019]

Algorithm 2: Constant-time evaluation of the class group action in CSIDH-512.

Input : $a \in \mathbb{F}_p$ such that $E_a : y^2 = x^3 + ax^2 + x$ is supersingular, and a list of integers $(e_1, ..., e_n)$ with $e_i \in \{0, 1, .., 10\}$ for all $i \leq n$.

Output: $a' \in \mathbb{F}_p$, the curve parameter of the resulting curve $E_{a'}$.

Initialize $k = 4$, $e = (e_1, ..., e_n)$ and $f = (f_1, ..., f_n)$, where $f_i = 10 - e_i$.
while some $e_i \neq 0$ or $f_i \neq 0$ **do**
    Sample random values $x \in \mathbb{F}_p$ until we have some x where $x^3 + ax^2 + x$ is a square in $\mathbb{F}_p$.
    Set $P = (x : 1)$, $P \leftarrow [k]P$, $S = \{i \mid e_i \neq 0 \text{ or } f_i \neq 0\}$.
    foreach $i \in S$ **do**
        Let $m = \prod_{j \in S, j > i} \ell_i$.
        Set $K \leftarrow [m]P$.
        if $K \neq \infty$ **then**
            if $e_i \neq 0$ **then**
                Compute a degree-ℓ_i isogeny $\varphi : E_a \to E_{a'}$ with $ker(\varphi) = \langle K \rangle$.
                $a \leftarrow a'$, $P \leftarrow \varphi(P)$, $e_i \leftarrow e_i - 1$.
            else
                Compute a degree-ℓ_i dummy isogeny:
                $a \leftarrow a$, $P \leftarrow [\ell_i]P$, $f_i \leftarrow f_i - 1$.
            if $e_i = 0$ *and* $f_i = 0$ **then**
                Set $k \leftarrow k \cdot \ell_i$.

Note: An improved version of this algorithm spends $\approx 48.5\%$ and $\approx 51.5\%$ computing scalar multiplications and isogeny-related operations, respectively.

Constant-time CSIDH algorithm by [Onuki et al., 2019]

- Efficiency improvements
  - To use two points to evaluate the action of an ideal, one in $\ker(\pi - 1)$ (i.e., in $E(\mathbb{F}_p)$) and one in $\ker(\pi + 1)$ (i.e., in $E(\mathbb{F}_{p^2})$ with x-coordinate in $\mathbb{F}_p$).
  - To move back the range of the exponents from $[0, 10]$ to $[-5, 5]$.
  - To construct and evaluate *exactly* 5 and 10 isogenies per ℓ_i prime, respectively (using dummy isogenies if required).

Implementation note: This CSIDH algorithm spends $\approx 55\%$ and $\approx 45\%$ computing scalar multiplications and isogeny-related operations, respectively.

CSIDH algorithm by [Onuki et al., 2019]

Algorithm 3 The Onuki–Aikawa–Yamazaki–Takagi CSIDH algorithm

Require: A supersingular curve $E_A : y^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$, and an integer exponent vector $(e_1, \ldots, e_n)$
Ensure: $E_B : y^2 = x^3 + Bx^2 + x$ such that $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.

$(e'_1, \ldots, e'_n) \leftarrow (m_i - |e_1|, \ldots, m_i - |e_n|)$,
$E_B \leftarrow E_A$
while some $e_i \neq 0$ or $e'_i \neq 0$ **do**
  $S \leftarrow \{i \mid e_i \neq 0 \text{ or } e'_i \neq 0\}$
  $k \leftarrow \prod_{i \in S} \ell_i$
  $(T_-, T_+) \leftarrow$ `Elligator`(E_B, u) // $T_- \in E_B[\pi - 1]$ and $T_+ \in E_B[\pi + 1]$
  $(P_0, P_1) \leftarrow ([(p+1)/k]T_+, [(p+1)/k]T_-)$
  for $i \in S$ **do**
    $s \leftarrow \mathrm{sign}(e_i)$ // Ideal $\mathfrak{l}_i^s$ to be used
    $Q \leftarrow [k/\ell_i]P_{\frac{1-s}{2}}$ // Secret kernel point generator
    $P_{\frac{1+s}{2}} \leftarrow [\ell_i]P_{\frac{1+s}{2}}$ // Secret point to be multiplied
    if $Q \neq \infty$ **then**
      if $e_i \neq 0$ **then**
        $(E_B, \varphi) \leftarrow$ `QuotientIsogeny`(E_B, Q)
        $(P_0, P_1) \leftarrow (\varphi(P_0), \varphi(P_1))$
        $e_i \leftarrow e_i - s$.
      else
        $E_B \leftarrow E_B$; $P_{\frac{1-s}{2}} \leftarrow [\ell_i]P_{\frac{1-s}{2}}$; $e'_i \leftarrow e'_i - 1$ // Dummies
      end if
    end if
    $k \leftarrow k/\ell_i$
  end for
end while
return B

Contributions of [Cervantes-Vázquez et al., 2019] to be discussed in the rest of this talk

1) A fully Twisted Edwards version of CSIDH;
2) Proposal of an efficient projective Elligator. Elligator is a procedure that maps strings to points in an elliptic curve. In the context of CSIDH an affine form of Elligator was proposed by [Meyer et al., 2019];
3) The usage of Shortest Differential Addition Chains (SDACs) in the CSIDH algorithm, which are cheaper than Classical Montgomery Ladders.
4) A stronger constant-time CSIDH algorithm without dummy operations.

A security issue regarding random point selection

In practice, one uses *Elligator*, which is an algorithm to efficiently sample points on a curve and its twist. However, elligator requires a random element $u \in \llbracket 2, \frac{p-1}{2} \rrbracket$ and also the inverse of $(u^2 - 1)$.

A security issue regarding random point selection

In practice, one uses *Elligator*, which is an algorithm to efficiently sample points on a curve and its twist. However, elligator requires a random element $u \in \llbracket 2, \frac{p-1}{2} \rrbracket$ and also the inverse of $(u^2 - 1)$.

- Aiming to avoid a costly inversion by $u^2 - 1$, Meyer, Campos and Reith, and Onuki *et al.* followed [Bernstein et al., 2019] and precomputed a set of ten pairs $(u, (u^2 - 1)^{-1})$;
- No randomness for u: But then, Elligator's output only depends on the A-coefficient of the current secret curve, which itself depends on the secret key.
- Running time of the algorithm varies and it is necessarily correlated to A and thus to the secret key.

A security issue regarding random point selection

To avoid field inversions, we write $V = (A : u^2 - 1)$, and we determine whether V is the abscissa of a projective point on E_A. Plugging V into the homogeneous equation

$$E_A : Y^2Z^2 = X^3Z + AX^2Z^2 + XZ^3$$

gives

$$Y^2(u^2-1)^2 = \big((A^2u^2 + (u^2-1)^2\big)A(u^2-1).$$

We can test the existence of a solution for Y by computing the Legendre symbol of the right hand side: if it is a square, the points with projective XZ-coordinates

$$T_+ = (A : u^2 - 1), \qquad T_- = (-Au^2 : u^2 - 1)$$

are in $E_A[\pi - 1]$ and $E_A[\pi + 1]$ respectively, otherwise their roles are swapped.

A security issue regarding random point selection

To avoid field inversions, we write $V = (A : u^2 - 1)$, and we determine whether V is the abscissa of a projective point on E_A. Plugging V into the homogeneous equation

$$E_A : Y^2Z^2 = X^3Z + AX^2Z^2 + XZ^3$$

gives

$$Y^2(u^2-1)^2 = \big((A^2u^2 + (u^2-1)^2\big)A(u^2-1).$$

We can test the existence of a solution for Y by computing the Legendre symbol of the right hand side: if it is a square, the points with projective XZ-coordinates

$$T_+ = (A : u^2 - 1), \qquad T_- = (-Au^2 : u^2 - 1)$$

are in $E_A[\pi - 1]$ and $E_A[\pi + 1]$ respectively, otherwise their roles are swapped. Consequently, u can be randomly chosen from $[\![2, \frac{p-1}{2}]\!]$, and elligator's output only depends on randomness.

Twisted Edwards or Montgomery curves?

From [Bernstein et al., 2008], we see that the Twisted Edwards curve

$$E_{a,d} : ax^2 + y^2 = 1 + dx^2y^2$$

is equivalent to the Montgomery curve

$$E_{(A:C)} : y^2 = x^3 + (A/C)x^2 + x$$

with constants

$$A_{24p} := A + 2C = a\,, \quad A_{24m} := A - 2C = d\,, \quad C_{24} := 4C = a - d\,.$$

In particular,

$$\psi : (X : Z) \longmapsto (Y : T) = (X - Z : X + Z)$$

ψ maps Montgomery XZ-coordinate points into Twisted Edwards YT-coordinate points, and

$$\psi^{-1} : (Y : T) \longmapsto (X : Z) = (T + Y : T - Y).$$

Twisted Edwards or Montgomery curves?

Using previous formulas, one can re-write the following Montgomery XZ-projective formulas in terms of Twisted Edwards YT-coordinates:

- Montgomery XZ-coordinates doubling
- Montgomery XZ-coordinates differential addition
- Montgomery XZ-coordinates degree-$(2k+1)$ isogeny evaluation.

Twisted Edwards or Montgomery curves?

Using previous formulas, one can re-write the following Montgomery XZ-projective formulas in terms of Twisted Edwards YT-coordinates:

- Montgomery XZ-coordinates doubling

$$
\begin{aligned}
X_{[2]P} &= C_{24}(X_P + Z_P)^2 (X_P - Z_P)^2, \\
Z_{[2]P} &= \left((X_P + Z_P)^2 - (X_P - Z_P)^2\right) \cdot \\
&\qquad \left(C_{24}(X_P - Z_P)^2 + A_{24p}((X_P + Z_P)^2 - (X_P - Z_P)^2)\right)
\end{aligned}
$$

- Montgomery XZ-coordinates differential addition
- Montgomery XZ-coordinates degree-$(2k+1)$ isogeny evaluation.

In particular, the computational costs of doubling and differential addition in YT-coordinates are $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{A}$, and $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$ (same as for XZ-coordinates).

Additionally, degree-$(2k+1)$ isogeny evaluation in XZ-coordinates costs $4k\mathbf{M} + 2\mathbf{S} + 6k\mathbf{A}$, whereas our YT-coordinate formula costs $4k\mathbf{M} + 2\mathbf{S} + (2k+4)\mathbf{A}$, thus saving $4k - 4$ field additions.

Twisted Edwards or Montgomery curves?

Using previous formulas, one can re-write the following Montgomery XZ-projective formulas in terms of Twisted Edwards YT-coordinates:

- Montgomery XZ-coordinates doubling

$$X_{[2]P} = (a-d){T_P}^2{Y_P}^2,$$
$$Z_{[2]P} = \left({T_P}^2 - {Y_P}^2\right)\cdot\left((a-d){Y_P}^2 + a({T_P}^2 - {Y_P}^2)\right)$$

- Montgomery XZ-coordinates differential addition
- Montgomery XZ-coordinates degree-$(2k+1)$ isogeny evaluation.

In particular, the computational costs of doubling and differential addition in YT-coordinates are $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{A}$, and $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$ (same as for XZ-coordinates).

Additionally, degree-$(2k+1)$ isogeny evaluation in XZ-coordinates costs $4k\mathbf{M} + 2\mathbf{S} + 6k\mathbf{A}$, whereas our YT-coordinate formula costs $4k\mathbf{M} + 2\mathbf{S} + (2k+4)\mathbf{A}$, thus saving $4k-4$ field additions.

Twisted Edwards or Montgomery curves?

Using previous formulas, one can re-write the following Montgomery XZ-projective formulas in terms of Twisted Edwards YT-coordinates:

- Montgomery XZ-coordinates doubling
- Montgomery XZ-coordinates differential addition

$$X_{P+Q} = Z_{P-Q}\big((X_P - Z_P)(X_Q + Z_Q) + (Z_P + Z_P)(X_Q - Z_Q)\big)^2$$
$$Z_{P+Q} = X_{P-Q}\big((X_P - Z_P)(X_Q + Z_Q) - (Z_P + Z_P)(X_Q - Z_Q)\big)^2$$

- Montgomery XZ-coordinates degree-$(2k+1)$ isogeny evaluation.

In particular, the computational costs of doubling and differential addition in YT-coordinates are $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{A}$, and $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$ (same as for XZ-coordinates).

Additionally, degree-$(2k+1)$ isogeny evaluation in XZ-coordinates costs $4k\mathbf{M} + 2\mathbf{S} + 6k\mathbf{A}$, whereas our YT-coordinate formula costs $4k\mathbf{M} + 2\mathbf{S} + (2k+4)\mathbf{A}$, thus saving $4k - 4$ field additions.

Twisted Edwards or Montgomery curves?

Using previous formulas, one can re-write the following Montgomery XZ-projective formulas in terms of Twisted Edwards YT-coordinates:

- Montgomery XZ-coordinates doubling
- Montgomery XZ-coordinates differential addition

$$X_{P+Q} = (T_{P-Q} - Y_{P-Q})\big(Y_P T_Q + T_P Y_Q\big)^2$$
$$Z_{P+Q} = (T_{P-Q} + Y_{P-Q})\big(Y_P T_Q - T_P Y_Q\big)^2$$

- Montgomery XZ-coordinates degree-$(2k+1)$ isogeny evaluation.

In particular, the computational costs of doubling and differential addition in YT-coordinates are $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{A}$, and $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$ (same as for XZ-coordinates).

Additionally, degree-$(2k+1)$ isogeny evaluation in XZ-coordinates costs $4k\mathbf{M} + 2\mathbf{S} + 6k\mathbf{A}$, whereas our YT-coordinate formula costs $4k\mathbf{M} + 2\mathbf{S} + (2k+4)\mathbf{A}$, thus saving $4k - 4$ field additions.

Twisted Edwards or Montgomery curves?

Using previous formulas, one can re-write the following Montgomery XZ-projective formulas in terms of Twisted Edwards YT-coordinates:

- Montgomery XZ-coordinates doubling
- Montgomery XZ-coordinates differential addition
- Montgomery XZ-coordinates degree-$(2k+1)$ isogeny evaluation. Let $\langle P \rangle$ be the kernel of the isogeny, and $(X_i : Z_i) = (x([i]P) : 1)$. Then

$$X' = X_P\Big(\prod_{i=1}^{k}\left((X-Z)(X_i+Z_i)+(X+Z)(X_i-Z_i)\right)\Big)^2$$

$$Z' = Z_P\Big(\prod_{i=1}^{k}\left((X-Z)(X_i+Z_i)-(X+Z)(X_i-Z_i)\right)\Big)^2$$

In particular, the computational costs of doubling and differential addition in YT-coordinates are $4\mathbf{M}+2\mathbf{S}+4\mathbf{A}$, and $4\mathbf{M}+2\mathbf{S}+6\mathbf{A}$ (same as for XZ-coordinates).

Additionally, degree-$(2k+1)$ isogeny evaluation in XZ-coordinates costs $4k\mathbf{M}+2\mathbf{S}+6k\mathbf{A}$, whereas our YT-coordinate formula costs $4k\mathbf{M}+2\mathbf{S}+(2k+4)\mathbf{A}$, thus saving $4k-4$ field additions.

Twisted Edwards or Montgomery curves?

Using previous formulas, one can re-write the following Montgomery XZ-projective formulas in terms of Twisted Edwards YT-coordinates:

- Montgomery XZ-coordinates doubling
- Montgomery XZ-coordinates differential addition
- Montgomery XZ-coordinates degree-$(2k+1)$ isogeny evaluation. Let $\langle P \rangle$ be the kernel of the isogeny, and $(Y_i : T_i) = (y([i]P) : 1)$. Then

$$X' = (T_P + Y_P)\Big(\prod_{i=1}^{k} (Y T_i + T Y_i)\Big)^2$$

$$Z' = (T_P - Y_P)\Big(\prod_{i=1}^{k} (Y T_i - T Y_i)\Big)^2$$

In particular, the computational costs of doubling and differential addition in YT-coordinates are $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{A}$, and $4\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$ (same as for XZ-coordinates).

Additionally, degree-$(2k+1)$ isogeny evaluation in XZ-coordinates costs $4k\mathbf{M} + 2\mathbf{S} + 6k\mathbf{A}$, whereas our YT-coordinate formula costs $4k\mathbf{M} + 2\mathbf{S} + (2k+4)\mathbf{A}$, thus saving $4k - 4$ field additions.

Outline

Addition chains for a faster scalar multiplication

Classical Montgomery ladders

$$
\begin{array}{c}
y(P), y([2]P) \\
\downarrow \\
y([3]P), y([4]P) \\
\downarrow \\
y([7]P), y([8]P) \\
\downarrow \\
y([15]P), y([16]P) \\
\downarrow \\
y([31]P), y([32]P) \\
\downarrow \\
y([63]P), y([64]P) \\
\downarrow \\
y([127]P), y([128]P)
\end{array}
$$

Example: given $y(P)$, $y([127]P)$ can be computed with 13 differential point operations.

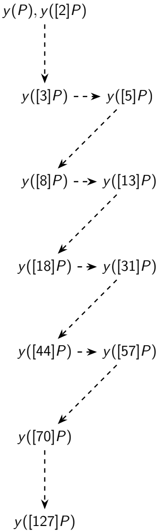
Classical Montgomery ladders

$$
\begin{array}{c}
y(P), y([2]P) \\
\downarrow \\
y([3]P), y([4]P) \\
\downarrow \\
y([7]P), y([8]P) \\
\downarrow \\
y([15]P), y([16]P) \\
\downarrow \\
y([31]P), y([32]P) \\
\downarrow \\
y([63]P), y([64]P) \\
\downarrow \\
y([127]P), y([128]P)
\end{array}
$$

Example: given $y(P)$, $y([127]P)$ can be computed with 13 differential point operations.

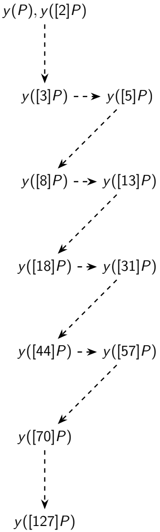
- Compute $y([\ell]P)$ requires $2 \times \lceil \log_2 \ell \rceil - 1$ differential point operations.

Shortest differential addition chains (SDACs)



Example: given $y(P)$, $y([127]P)$ can be computed with 11 differential point operations.

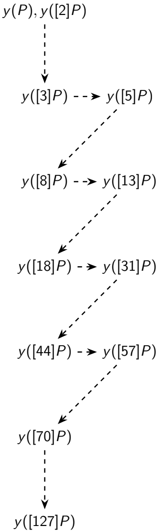
Shortest differential addition chains (SDACs)



Example: given $y(P)$, $y([127]P)$ can be computed with 11 differential point operations.

- Compute $y([\ell]P)$ requires $\approx 1.5 \times \lceil \log_2 \ell \rceil$ differential point operations,
- SDACs yields a saving of $\approx 25\%$ compared with the cost of the classical Montgomery ladder,

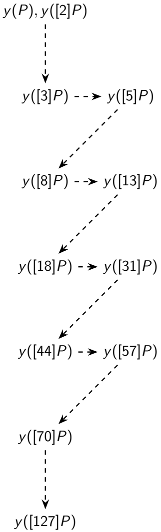
Shortest differential addition chains (SDACs)



Example: given $y(P)$, $y([127]P)$ can be computed with 11 differential point operations.

- Compute $y([\ell]P)$ requires $\approx 1.5 \times \lceil \log_2 \ell \rceil$ differential point operations,
- SDACs yields a saving of $\approx 25\%$ compared with the cost of the classical Montgomery ladder,
- SDACs are not constant-time,

Shortest differential addition chains (SDACs)



Example: given $y(P)$, $y([127]P)$ can be computed with 11 differential point operations.

- Compute $y([\ell]P)$ requires $\approx 1.5 \times \lceil \log_2 \ell \rceil$ differential point operations,
- SDACs yields a saving of $\approx 25\%$ compared with the cost of the classical Montgomery ladder,
- SDACs are not constant-time,
- But each scalar ℓ is public thus it's okay to use SDACs!

Constant-time CSIDH algorithm [Meyer et al., 2019, Onuki et al., 2019]

In both the original CSIDH and the Onuki *et al.* variants $e_i \in [\![-m_i, m_i]\!]$, while in Meyer-Campos-Reith variant $e_i \in [\![0, m_i]\!]$. Notice that in constant-time implementations of CSIDH, the exponents e_i are implicitly interpreted as

$$|e_i| = \underbrace{1 + 1 + \cdots + 1}_{e_i \text{ times}} + \underbrace{{\color{red}0 + 0 + \cdots}}_{m_i - e_i \text{ times}},$$

Then these procedures start constructing isogenies with kernel generated by $P \in E_A[\ell_i, \pi - \mathtt{sign}(e_i)]$ (for e_i iterations), followed by dummy isogeny computations (for $m_i - e_i$ iterations).

CSIDH with dummy operations

To mitigate power consumption analysis attacks, the constant-time algorithms proposed in [Meyer et al., 2019] and [Onuki et al., 2019] always compute the maximal amount of isogenies allowed by the exponent, using dummy isogeny computations if needed.

This countermeasure is susceptible of fault attacks.

Removing dummy operations

For our new approach, the exponents e_i are uniformly sampled from sets

$$\mathcal{S}(m_i) = \{e \mid e = m_i \bmod 2 \text{ and } |e| \leq m_i\},$$

i.e., centered intervals containing only even or only odd integers.

Removing dummy operations

For our new approach, the exponents e_i are uniformly sampled from sets

$$\mathcal{S}(m_i) = \{e \mid e = m_i \text{ mod } 2 \text{ and } |e| \le m_i\},$$

i.e., centered intervals containing only even or only odd integers.
Consequently, the exponents e_i can implicitly interpreted as

$$|e_i| = \underbrace{1 + 1 + \cdots + 1}_{e_i \text{ times}} + \underbrace{(1-1) - (1-1) + (1-1) - \cdots}_{m_i - e_i \text{ times}},$$

and then our approach starts by constructing isogenies with kernel generated by $P \in E_A[\ell_i, \pi - \texttt{sign}(e_i)]$ for e_i iterations, then alternates between isogenies with kernel generated by $P \in E_A[\ell_i, \pi - 1]$ and $P \in E_A[\ell_i, \pi + 1]$ for $(m_i - e_i)$ iterations.

A dummy-free CSIDH algorithm

Require: A supersingular curve E_A over $\mathbb{F}_p$, and an exponent vector $(e_1, \dots, e_n)$ with each $e_i \in [-m_i, m_i]$ and $e_i \equiv m_i \pmod 2$.

Ensure: $E_B = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_A$.

```
(t_1, ..., t_n) ← ( (sign(e_1)+1)/2, ..., (sign(e_n)+1)/2 )
(z_1, ..., z_n) ← (m_1, ..., m_n)
E_B ← E_A
while some z_i ≠ 0 do
    u ← Random({2, ..., (p−1)/2})
    (T_1, T_0) ← Elligator(E_B, u) // T_1 ∈ E_B[π − 1] and T_0 ∈ E_B[π + 1]
    (T_0, T_1) ← ([4]T_0, [4]T_1) // Now T_0, T_1 ∈ E_B [∏_i ℓ_i]
    for i ∈ {1, ..., n} do
        if z_i ≠ 0 then
            (G_0, G_1) ← (T_0, T_1)
            for j ∈ {i + 1, ..., n} do
                (G_0, G_1) ← ([ℓ_j]G_0, [ℓ_j]G_1)
            end for
            if G_0 ≠ ∞ and G_1 ≠ ∞ then
                cswap(G_0, G_1, t_i) // Secret kernel point generator: G_0
                cswap(T_0, T_1, t_i) // Secret point to be multiplied: T_1
                (E_B, φ) ← QuotientIsogeny(E_B, G_0)
                (T_0, T_1) ← (φ(T_0), φ(T_1))
                T_1 ← [ℓ_i]T_1
                cswap(T_0, T_1, t_i)
                b ← isequal(e_i, 0)
                e_i ← e_i + (−1)^{t_i}
                t_i ← t_i ⊕ b
                z_i ← z_i − 1
            else if G_0 ≠ ∞ then
                T_0 ← [ℓ_i]T_0
            else if G_1 ≠ ∞ then
                T_1 ← [ℓ_i]T_1
            end if
        end if
    end for
end while
return B
```

Derandomized CSIDH algorithms

- The CSIDH algorithms described here depend on the availability of high-quality randomness for their security.
- If the attacker knows the output of the PRNG, or if the quality of the PRNG output is less than ideal, this may degrade the security of all algorithms.

Derandomized CSIDH algorithms

- Hence, we suggest modifying CSIDH by restricting to exponents of the private key sampled from
  - $\{-1, 0, 1\}$, or
  - $\{-1, 1\}$ (if fault-injection attacks are a concern);
- One can then precompute two points of order $(p + 1)/4$ on the starting public curve, one in $E_A[\pi - 1]$ and the other in $E_A[\pi + 1]$.
- However, for achieving a 128 bits security level, the prime p goes from 511 bits to almost 1500 (slower but much stronger quantum security).

Note: This approach could be of interest for simulating quantum attacks where strict constant time behavior is desirable [Bernstein et al., 2019]

Running-time: field operations

Table 1: Field operation counts for constant-time CSIDH. Counts are given in millions of operations, averaged over 1024 random experiments. The performance ratio uses [Meyer et al., 2019] as a baseline, considers only multiplication and squaring operations, and assumes $M = S$.

Implementation	CSIDH Algorithm	M	S	A	Ratio
Castryck et al. [Castryck et al., 2018]	unprotected, unmodified	0.252	0.130	0.348	0.26
Meyer–Campos–Reith [Meyer et al., 2019]	unmodified	1.054	0.410	1.053	1.00
Onuki et al. [Onuki et al., 2019]	unmodified	0.733	0.244	0.681	0.67
This work	MCR-style	0.901	0.309	0.965	0.83
	OAYT-style	0.657	0.210	0.691	0.59
	No-dummy	1.319	0.423	1.389	1.19

Running-time: measured clock cycles

Table 2: Clock cycle counts for constant-time CSIDH implementations, averaged over 1024 experiments. The ratio is computed using [Meyer et al., 2019] as baseline implementation.

Implementation	**CSIDH algorithm**	**Mcycles**	**Ratio**
Castryck et al. [Castryck et al., 2018]	unprotected, unmodified	155	0.39
Meyer–Campos–Reith [Meyer et al., 2019]	unmodified	395	1.00
This work	MCR-style	337	0.85
	OAYT-style	239	0.61
	No-dummy	481	1.22

[Some] Open questions

1) Should we enlarge the CSIDH prime to improve its quantum security?
2) Using the framework by [Adj et al., 2019] for classical attacks should we shrink the CSIDH prime from 512-bits to something around 430 bits? [on-going work with A. Menezes and the Cinvestav crypto group]
3) What is the probability of failure due to Elligator output points that are not full torsion points? How the different algorithmic tricks so far proposed affect this probability?
4) Can strategies à la SIDH be applied more effectively?
    - Check again [Hutchinson et al., 2019]
    - Look for optimizations using heuristics and/or deep learning approaches

Thank you for your attention

I look forward to your comments and questions.

e-mail: francisco@cs.cinvestav.mx

Our software library is freely available from

`https://github.com/JJChiDguez/csidh.`

References I

- Adj, G., Cervantes-Vázquez, D., Chi-Domínguez, J., Menezes, A., and Rodríguez-Henríquez, F. (2019).

  On the cost of computing isogenies between supersingular elliptic curves.

  In Cid, C. and Jr., M. J. J., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 322–343. Springer.

- Bernstein, D. J., Birkner, P., Joye, M., Lange, T., and Peters, C. (2008).

  Twisted Edwards curves.

  In Vaudenay, S., editor, *Progress in Cryptology - AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 389–405. Springer.

References II

- Bernstein, D. J., Lange, T., Martindale, C., and Panny, L. (2019).
  Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies.
  In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 409–441.
- Bonnetain, X. and Schrottenloher, A. (2018).
  Submerging csidh.
  Cryptology ePrint Archive, Report 2018/537.
  https://eprint.iacr.org/2018/537.
- Castryck, W., Lange, T., Martindale, C., Panny, L., and Renes, J. (2018).
  CSIDH: an efficient post-quantum commutative group action.
  In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, pages 395–427.

References III

- Cervantes-Vázquez, D., Chenu, M., Chi-Domínguez, J., Feo, L. D., Rodríguez-Henríquez, F., and Smith, B. (2019).
  Stronger and faster side-channel protections for CSIDH.
  In *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, pages 173–193.
- Childs, A. M., Jao, D., and Soukharev, V. (2010).
  Constructing elliptic curve isogenies in quantum subexponential time.
  CoRR, abs/1012.4019.
- Couveignes, J. M. (2006).
  Hard homogeneous spaces.
  Cryptology ePrint Archive, Report 2006/291.

References IV

- De Feo, L., Kieffer, J., and Smith, B. (2018).
  Towards practical key exchange from ordinary isogeny graphs.
  In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, pages 365–394.
- Hutchinson, A., LeGrow, J., Koziel, B., and Azarderakhsh, R. (2019).
  Further optimizations of csidh: A systematic approach to efficient strategies, permutations, and bound vectors.
  Cryptology ePrint Archive, Report 2019/1121.
  `https://eprint.iacr.org/2019/1121`.
- Jalali, A., Azarderakhsh, R., Kermani, M. M., and Jao, D. (2019).
  Towards optimized and constant-time CSIDH on embedded devices.
  In *Constructive Side-Channel Analysis and Secure Design*, pages 215–231. Springer International Publishing.

References V

- Meyer, M., Campos, F., and Reith, S. (2019).
  On lions and elligators: An efficient constant-time implementation of CSIDH.
  In *Post-Quantum Cryptography - 10th International Workshop, PQCrypto 2019*.
- Meyer, M. and Reith, S. (2018).
  A faster way to the CSIDH.
  In *Progress in Cryptology - INDOCRYPT 2018 - 19th International Conference on Cryptology in India, New Delhi, India, December 9-12, 2018, Proceedings*, pages 137–152.
- Onuki, H., Aikawa, Y., Yamazaki, T., and Takagi, T. (2019).
  A faster constant-time algorithm of CSIDH keeping two torsion points.
  To appear in IWSEC 2019 – The 14th International Workshop on Security.

References VI

- Peikert, C. (2019).
  He gives c-sieves on the csidh.
  Cryptology ePrint Archive, Report 2019/725.
  https://eprint.iacr.org/2019/725.

- Rostovtsev, A. and Stolbunov, A. (2006).
  Public-key cryptosystem based on isogenies.
  Cryptology ePrint Archive, Report 2006/145.

- Stolbunov, A. (2010).
  Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves.
  Advances in Mathematics of Communication, 4(2).