

Batching CSIDH Group Actions using AVX-512

Hao Cheng, Georgios Fotiadis, Johann Großschädl,
Peter Y. A. Ryan, Peter B. Rønne



Department of Computer Science (DCS)

Interdisciplinary Centre for Security Reliability & Trust (SnT)

University of Luxembourg

CHES 2021

September 13–17, 2021

CSIDH

Commutative Supersingular Isogeny Diffie-Hellman – CSIDH

- ▶ Based on the [action of an ideal class group](#) on a set of supersingular ECs.
- ▶ A [non-interactive](#) key exchange protocol.

Advantages

- ▶ Allows for [full public-key validation](#).
- ▶ Short key lengths.
- ▶ [“Drop-in”](#) PQ replacement for classical ECDH (e.g. X25519).

Major disadvantage: High computational cost

- ▶ CSIDH-512 (group action + PK validation): ≈ 239 M Skylake cycles [[CCC⁺19](#)].
- ▶ SIKEp434 encapsulation or decapsulation: ≈ 10 M Skylake cycles [[JAC⁺20](#)].
- ▶ X25519 variable-base scalar multiplication: < 100 k Skylake cycles [[NS20](#)].

The CSIDH Group Action

Setting:

- ▶ Small, odd primes ℓ_1, \dots, ℓ_n s.t. $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ is prime
- ▶ $E_A : y^2 = x^3 + Ax^2 + x$ supersingular elliptic curve over \mathbb{F}_p (s.t. $A^2 \neq 4$)
- ▶ Hence $\#E(\mathbb{F}_p) = p + 1 = 4 \cdot \ell_1 \cdots \ell_n$

CSIDH group action:

- ▶ Compute the action of an ideal $\mathfrak{a} = \mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}$ on E_A .
- ▶ $\mathfrak{l}_1, \dots, \mathfrak{l}_n$: prime ideals, e_1, \dots, e_n : small exponents chosen from some $[-b, b]$
- ▶ Equiv. compute curve E' as the image of the isogeny ϕ of degree $\ell_1^{e_1} \cdots \ell_n^{e_n}$.
- ▶ CSIDH key feature: break into smaller isogenies:

$$\begin{array}{ccc} |\mathfrak{l}_1| & \text{isogenies of degree} & \ell_1 \\ \vdots & \vdots & \vdots \\ |\mathfrak{l}_n| & \text{isogenies of degree} & \ell_n \end{array}$$

- ▶ Each isogeny using Vélu formulæ [Vél71] with $O(\ell_i \log p^2)$ bit operations

The CSIDH protocol

CSIDH parameters:

- ▶ **Public params:** prime $p = 4 \cdot \ell_1 \cdots \ell_n - 1$, starting curve: $E_0 : y^2 = x^3 + x$
- ▶ **Secret keys:** exponents $(e_1, \dots, e_n), (d_1, \dots, d_n)$ chosen randomly from $[-b, b]^n$

Alice (Client/User)

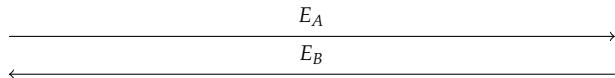
$$sk_A = (e_1, \dots, e_n)$$

$$pk_A = \mathbf{a} \star E_0 = E_A$$

Bob (Server)

$$sk_B = (d_1, \dots, d_n)$$

$$pk_B = \mathbf{b} \star E_0 = E_B$$



Check whether E_B is valid

if valid, $k_A = \mathbf{a} \star E_B = \mathbf{a} \star (\mathbf{b} \star E_0)$

Check whether E_A is valid

if valid, $k_B = \mathbf{b} \star E_A = \mathbf{b} \star (\mathbf{a} \star E_0)$

Curves k_A and k_B are **isomorphic**

(Some) Constant-time Implementations/Optimizations

Bernstein, Lange, Martindale, and Panny [BLMP19]

- ▶ 1st constant-time implementation, dummy computations, failure probability

Meyer, Campos, and Reith [MCR19]: MCR-style

- ▶ Only positive secret exponents, Elligator 2 map, SIMBA- $m-\mu$

Onuki, Aikawa, Yamazaki, and Takagi [OAYT19]: OAYT-style

- ▶ Two-point approach, allows for negative secret exponents

Cervantes-Vázquez *et al.* [CCC⁺19]:

- ▶ Improved MCR and OAYT algorithms (efficient formulas in twisted Edwards model, differential addition chains, dummy-free group action)

Hutchinson, LeGrow, Koziel, and Azarderakhsh [HLKA20]

- ▶ Optimal strategies for choosing b_i , the ordering of ℓ_i and SIMBA parameters

Our Software: CSIDH-512 Group Action using AVX-512

High-throughput implementation (batched, 8 parallel instances)

- ▶ Use case 1: increase throughput of server-side TLS processing.
- ▶ Use case 2: minimize latency of a single CSIDH-based signature, e.g. CSI-FiSh [BKV19] and SeaSign [DG19].

Low-latency implementation (1 instance)

- ▶ Use case: accelerate CSIDH on client side of TLS.

Both the high-throughput version and the low-latency version were implemented with AVX-512F and AVX-512IFMA instructions.

x64 software used as starting point:

- ▶ OAYT and Dummy-Free style implementations of Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez, and Smith [CCC⁺19]. *Latincrypt'19* (SIMBA, two-point approach, twisted Edwards model, differential addition chains)

AVX-512

Advanced Vector eXtensions (Intel)

- ▶ 512-bit vectors, Single Instruction Multiple Data
- ▶ Operations on e.g. eight 64-bit elements in parallel (usually 29-bit limbs)

Example (AVX-512F): $r = _mm512_mul_epu32(a, b)$

$$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7 \rangle \times \langle b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7 \rangle$$

↓

$$\langle a_0 \times b_0, a_1 \times b_1, a_2 \times b_2, a_3 \times b_3, a_4 \times b_4, a_5 \times b_5, a_6 \times b_6, a_7 \times b_7 \rangle$$

AVX-512IFMA (vpmadd52luq and vpmadd52huq)

- ▶ Two new instructions for big number multiplication (52-bit limbs)
- ▶ Integer Fused Multiply-Add:

1. $a \times b \rightarrow t$

2. $t_L/t_H + c \rightarrow r$

(52 × 52)-bit → 104-bit

52-bit + 64-bit → 64-bit

Original CSIDH Group Action [CLM⁺18]

In: secret exp. (e_1, \dots, e_n) , s.t. $e_i \in [-b_i, b_i]$; starting curve E_A

Out: target curve $E_B = [l_1^{e_1} \cdots l_n^{e_n}] * E_A$

```
01  $E_B \leftarrow E_A$ 
02 while some  $e_j \neq 0$ :
03    $(P_0, P_1) \leftarrow \text{Elligator}(E_B)$  //  $P_0 \in E[\pi - 1]$  and  $P_1 \in E[\pi + 1]$ 
04    $(T_0, T_1) \leftarrow ([4]P_0, [4]P_1)$  //  $\text{ord}(T_0) = \text{ord}(T_1) = (p + 1)/4 = l_1 \cdots l_n$ 
05   for  $i = 1, \dots, n$ :
06      $s \leftarrow \text{sign}(e_i)$  //  $s = 0$ , if  $e_i < 0$ ;  $s = 1$ , if  $e_i \geq 0$ 
07      $\text{cswap}(T_0, T_1, 1 - s)$  // swap if  $s = 0 \Rightarrow e_i < 0$ 
08      $R \leftarrow [l_{i+1} \cdots l_n]T_0$  //  $\text{ord}(R) = l_i$ 
09      $T_1 \leftarrow [l_i]T_1$  //  $\text{ord}(T_1) = l_{i+1} \cdots l_n$ 
10     if  $R \neq \infty$  then:
11        $E_C, \phi \leftarrow \{\text{isogeny}; \text{evaluation}; \text{update}\}$ 
12        $E_B \leftarrow E_C, e_i \leftarrow e_i + (-1)^s$  // update curve and secret exp.
13      $\text{cswap}(T_0, T_1, 1 - s)$  // back to the initial point ordering
14 return  $E_B$ 
```


Original CSIDH Group Action [CLM⁺18]

The algorithm is **not constant-time**, which depends on $e_i \in [-b_i, b_i]$.

$$\left. \begin{array}{l} \text{isog. of deg: } \ell_1 \rightarrow |e_1| \leq b_i \text{ times} \\ \text{isog. of deg: } \ell_2 \rightarrow |e_2| \leq b_i \text{ times} \\ \vdots \\ \text{isog. of deg: } \ell_{74} \rightarrow |e_{74}| \leq b_i \text{ times} \end{array} \right\} \Rightarrow \#\text{isog} = \sum_{i=1}^{74} |e_i|$$

OAYT-style CSIDH [OAYT19]: computes b_i isogenies instead of $|e_i|$ for each ℓ_i
 $\Rightarrow |e_i|$ “real” + $(b_i - |e_i|)$ “dummy” isogenies:

$$\left. \begin{array}{l} b_1 \text{ isog. of deg: } \ell_1 \rightarrow |e_1| \text{ “real”} + (b_1 - |e_1|) \text{ “dummy”} \\ b_2 \text{ isog. of deg: } \ell_2 \rightarrow |e_2| \text{ “real”} + (b_2 - |e_2|) \text{ “dummy”} \\ \vdots \\ b_{74} \text{ isog. of deg: } \ell_{74} \rightarrow |e_{74}| \text{ “real”} + (b_{74} - |e_{74}|) \text{ “dummy”} \end{array} \right\} \Rightarrow \#\text{isog} = \sum_{i=1}^{74} b_i = 404$$

OAYT-Style CSIDH Group Action [OAYT19, CCC⁺19]

In: secret exp. (e_1, \dots, e_n) , s.t. $e_i \in [-b_i, b_i]$; starting curve E_A

Out: target curve $E_B = [l_1^{e_1} \cdots l_n^{e_n}] * E_A$

```
01  $E_B \leftarrow E_A, \#isog \leftarrow 0$ 
02 while some  $b_j \neq 0$ :
03    $(P_0, P_1) \leftarrow \text{Elligator}(E_B); (T_0, T_1) \leftarrow ([4]P_0, [4]P_1)$ 
04   for  $i = 1, \dots, n$ :
06      $s \leftarrow \text{sign}(e_i)$  //  $s = 0$ , if  $e_i < 0$ ;  $s = 1$ , if  $e_i \geq 0$ 
07      $\text{cswap}(T_0, T_1, 1 - s)$  // swap if  $s = 0 \Rightarrow e_i < 0$ 
08      $R \leftarrow [l_{i+1} \cdots l_n]T_0$  //  $\text{ord}(R) = l_i$ 
09      $T_1 \leftarrow [l_i]T_1$  //  $\text{ord}(T_1) = l_{i+1} \cdots l_n$ 
10     if  $R \neq \infty$  then:
11        $w \leftarrow 1 - \text{isequal}(e_i, 0)$  //  $w = 0$ , if  $e_i = 0$ ;  $w = 1$ , if  $e_i \neq 0$ 
12        $E_C, \phi \leftarrow \{\text{isogeny}; \text{evaluation}; \text{update}\}$ 
13        $\text{cswap}(E_B, E_C, w)$  // update curve if  $w = 1 \Rightarrow e_i \neq 0$ 
14        $e_i \leftarrow e_i + (-1)^s \cdot w, b_i --$ 
15        $\text{cswap}(T_0, T_1, 1 - s)$  // back to the initial point ordering
16 return  $E_B$ 
```

Obstacles to Batching CSIDH

SIMD requires the parallel group action instances to possess **the same operation (instruction) sequence**.

The kernel generator R might be **a point at infinity** in some instances.

- ▶ Leads to different operation sequences in eight parallel instances.
- ▶ $\text{Prob}_{R=\infty} = 1/\ell_i$, considerably high when the prime $\ell_i = 3, 5, 7, \dots$

```
group action #1 ( $R \neq \infty$ )
09 ...
10 if  $R \neq \infty$  then:
11    $w \leftarrow 1 - \text{isequal}(e_i, 0)$ 
12    $E_C, \phi \leftarrow \{\text{isog}; \text{eval}; \text{update}\}$ 
13    $\text{cswap}(E_B, E_C, w)$ 
14    $e_i \leftarrow e_i + (-1)^s \cdot w, b_i --$ 
15  $\text{cswap}(T_0, T_1, 1 - s)$ 
```

```
group action #2 ( $R = \infty$ )
09 ...
10 if  $R \neq \infty$  then:
11    $w \leftarrow 1 - \text{isequal}(e_i, 0)$ 
12    $E_C, \phi \leftarrow \{\text{isog}; \text{eval}; \text{update}\}$ 
13    $\text{cswap}(E_B, E_C, w)$ 
14    $e_i \leftarrow e_i + (-1)^s \cdot w, b_i --$ 
15  $\text{cswap}(T_0, T_1, 1 - s)$ 
```

Not **batching-friendly**!

The Extra-Dummy Batching Method

Extra dummy isogeny computations for $R = \infty$ [BLMP19, JAKJ19].

In: secret exp. (e_1, \dots, e_n) , s.t. $e_i \in [-b_i, b_i]$; starting curve E_A ; (c_1, \dots, c_n)

Out: target curve $E_B = [l_1^{e_1} \cdots l_n^{e_n}] * E_A$

```
01  $E_B \leftarrow E_A$ , #isog  $\leftarrow 0$ , each  $g_j \leftarrow b_j + c_j$ 
02 while some  $g_j \neq 0$ :
03    $(P_0, P_1) \leftarrow \text{Elligator}(E_B)$ ;  $(T_0, T_1) \leftarrow ([4]P_0, [4]P_1)$ 
04   for  $i = 1, \dots, n$ :
06      $s \leftarrow \text{sign}(e_i)$  //  $s = 0$ , if  $e_i < 0$ ;  $s = 1$ , if  $e_i \geq 0$ 
07      $\text{cswap}(T_0, T_1, 1 - s)$  // swap if  $s = 0 \Rightarrow e_i < 0$ 
08      $R \leftarrow [l_{i+1} \cdots l_n]T_0$  //  $\text{ord}(R) = l_i$ 
09      $T_1 \leftarrow [l_i]T_1$  //  $\text{ord}(T_1) = l_{i+1} \cdots l_n$ 
10      $f \leftarrow 1 - \text{isinfinity}(R)$  //  $f = 0$ , if  $R = \infty$ ;  $f = 1$ , if  $R \neq \infty$ 
11      $w \leftarrow 1 - \text{isequal}(e_i, 0)$  //  $w = 0$ , if  $e_i = 0$ ;  $w = 1$ , if  $e_i \neq 0$ 
12      $E_C, \phi \leftarrow \{\text{isogeny}; \text{evaluation}; \text{update}\}$ 
13      $\text{cswap}(E_B, E_C, w \& f)$  // update curve if  $w \& f = 1 \Rightarrow e_i \neq 0 \ \& \ R \neq \infty$ 
14      $e_i \leftarrow e_i + (-1)^s \cdot (w \& f)$ ,  $g_i --$ 
15      $\text{cswap}(T_0, T_1, 1 - s)$  // back to the initial point ordering
16 return  $E_B$ 
```

The Extra-Dummy Batching Method

isog of deg ℓ_1	\rightarrow	$ e_1 $	“real”	+	$(b_1 + c_1 - e_1)$	“dummy”	} \Rightarrow #isog = $\sum_{i=1}^{74} (b_i + c_i)$
isog of deg ℓ_2	\rightarrow	$ e_2 $	“real”	+	$(b_2 + c_2 - e_2)$	“dummy”	
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	
isog of deg ℓ_{74}	\rightarrow	$ e_{74} $	“real”	+	$(b_{74} + c_{74} - e_{74})$	“dummy”	

Problem:

- ▶ Failure probability $\text{Prob}_{\text{fail}}$ when $\#\text{dummy} > (b_i + c_i - |e_i|)$ for at least one $i = 1, \dots, 74 \Leftrightarrow$ when too many $R = \infty$.
- ▶ $\text{Prob}_{\text{fail}} < 2^{-32}$ needs ~ 900 isogenies (originally 404).

Solution:

- ▶ Hybrid mode!

Hybrid Mode

Batched component.

- ▶ Incomplete batched implementation performing 8 instances.
- ▶ Computes the main bulk of the group action evaluation.

Unbatched component.

- ▶ Latency-optimized implementation accelerating a single group action evaluation such as [CCC⁺19] and our low-latency AVX-512 implementation.
- ▶ Perform the remaining computation for each instance.

$$\begin{aligned} \langle \hat{B}^{(1)}, \dots, \hat{B}^{(8)} \rangle, \langle \hat{\mathbf{b}}^{(1)}, \dots, \hat{\mathbf{b}}^{(8)} \rangle, \langle \mathbf{e}'^{(1)}, \dots, \mathbf{e}'^{(8)} \rangle &\leftarrow \text{batched}(\langle A^{(1)}, \dots, A^{(8)} \rangle, \mathbf{b}, \langle \mathbf{e}^{(1)}, \dots, \mathbf{e}^{(8)} \rangle) \\ &\downarrow \\ B^{(k)} &\leftarrow \text{unbatched}(\hat{B}^{(k)}, \hat{\mathbf{b}}^{(k)}, \mathbf{e}'^{(k)}) \text{ for } k = 1, \dots, 8 \\ &\downarrow \\ &B^{(1)}, \dots, B^{(8)} \end{aligned}$$

$$\Rightarrow \# \text{isog} = \sum_{i=1}^{74} b_i + \sum_{i=1}^{74} \hat{b}_i \approx 414$$

The Extra-Dummy Batching Method (Batched Component)

In: secret exp. (e_1, \dots, e_n) , s.t. $e_i \in [-b_i, b_i]$; starting curve E_A ;
Out: target curve $E_B = [l_1^{e_1} \cdots l_n^{e_n}] * E_A$; secret exp. (e'_1, \dots, e'_n) ; bound $(\hat{b}_1, \dots, \hat{b}_n)$

```
01  $E_B \leftarrow E_A$ ,  $\#isog \leftarrow 0$ ,  $(e'_1, \dots, e'_n) \leftarrow (e_1, \dots, e_n)$ , each  $\hat{b}_j \leftarrow b_j$ 
02 while some  $b_j \neq 0$ :
03    $(P_0, P_1) \leftarrow \text{Elligator}(E_B)$ ;  $(T_0, T_1) \leftarrow ([4]P_0, [4]P_1)$ 
04   for  $i = 1, \dots, n$ :
06      $s \leftarrow \text{sign}(e'_i)$  //  $s = 0$ , if  $e'_i < 0$ ;  $s = 1$ , if  $e'_i \geq 0$ 
07      $\text{cswap}(T_0, T_1, 1 - s)$  // swap if  $s = 0 \Rightarrow e'_i < 0$ 
08      $R \leftarrow [l_{i+1} \cdots l_n]T_0$  //  $\text{ord}(R) = l_i$ 
09      $T_1 \leftarrow [l_i]T_1$  //  $\text{ord}(T_1) = l_{i+1} \cdots l_n$ 
10      $f \leftarrow 1 - \text{isinfinity}(R)$  //  $f = 0$ , if  $R = \infty$ ;  $f = 1$ , if  $R \neq \infty$ 
11      $w \leftarrow 1 - \text{isequal}(e'_i, 0)$  //  $w = 0$ , if  $e'_i = 0$ ;  $w = 1$ , if  $e'_i \neq 0$ 
12      $E_C, \phi \leftarrow \{\text{isogeny}; \text{evaluation}; \text{update}\}$ 
13      $\text{cswap}(E_B, E_C, w \& f)$  // update curve if  $w \& f = 1 \Rightarrow e'_i \neq 0 \ \& \ R \neq \infty$ 
14      $e'_i \leftarrow e'_i + (-1)^s \cdot (w \& f)$ ,  $b_i --$ ,  $\hat{b}_i \leftarrow \hat{b}_i - f$ 
15      $\text{cswap}(T_0, T_1, 1 - s)$  // back to the initial point ordering
16 return  $E_B$ ,  $(e'_1, \dots, e'_n)$ ,  $(\hat{b}_1, \dots, \hat{b}_n)$ 
```

The Extra-Infinity Batching Method

```
    group action #1 ( $R \neq \infty$ )
09 ...
10 if  $R \neq \infty$  then:
11    $w \leftarrow 1 - \text{isequal}(e_i, 0)$ 
12    $E_C, \phi \leftarrow \{\text{isog}; \text{eval}; \text{update}\}$ 
13    $\text{cswap}(E_B, E_C, w)$ 
14    $e_i \leftarrow e_i + (-1)^s \cdot w, b_i - -$ 
15 else:
16    $T_0 \leftarrow [\ell_i]T_0$ 
17  $\text{cswap}(T_0, T_1, 1 - s)$ 
```

```
    group action #2 ( $R = \infty$ )
09 ...
10 if  $R \neq \infty$  then:
11    $w \leftarrow 1 - \text{isequal}(e_i, 0)$ 
12    $E_C, \phi \leftarrow \{\text{isog}; \text{eval}; \text{update}\}$ 
13    $\text{cswap}(E_B, E_C, w)$ 
14    $e_i \leftarrow e_i + (-1)^s \cdot w, b_i - -$ 
15 else:
16    $T_0 \leftarrow [\ell_i]T_0$ 
17  $\text{cswap}(T_0, T_1, 1 - s)$ 
```

If $R = \infty$ in at least one instance

\Rightarrow execute the “else”-branch in all group actions!

The ℓ_i -torsion parts of some points $T_0^{(k)}$ have not vanished.

The Extra-Infinity Batching Method

$$\begin{array}{l} \text{inf} \leftarrow \text{isinfinit y}(R^{(1)}) \mid \dots \mid \text{isinfinit y}(R^{(8)}) \\ \downarrow \\ \text{inf} = 0, \quad R^{(k)} \neq \infty \text{ for all eight instances,} \quad \text{goto "if"-branch} \\ \text{inf} = 1, \quad R^{(k)} = \infty \text{ in at least one instance,} \quad \text{goto "else"-branch} \end{array}$$

Problem:

- ▶ Still #isog = 404 but more **infinity-related computations**
 - ▶ $T_0 \leftarrow [\ell_i]T_0$ in the “else”-branch.
 - ▶ $R \leftarrow [\ell_{i+1} \cdots \ell_n]T_0, T_1 \leftarrow [\ell_i]T_1$ in the inner “for” loop.
 - ▶ $(P_0, P_1) \leftarrow \text{Elligator}(E_B), (T_0, T_1) \leftarrow ([4]P_0, [4]P_1)$ in the outer “while” loop.
- ▶ $\text{Prob}_{\text{inf}=1} = 1 - (1 - 1/\ell_i)^8$, quite high when ℓ_i is 3, 5, 7...

Solution:

- ▶ Hybrid mode!

Hybrid Mode

Divide small primes into two subsets

$$\Rightarrow \boldsymbol{\ell} = (\ell_1, \ell_2, \dots, \ell_{74}) \rightarrow \boldsymbol{\ell}_{\text{unbatch}} = (3, 5, 7, 11, 13, 17, 19), \boldsymbol{\ell}_{\text{batch}} = \mathbb{C}_{\ell} \boldsymbol{\ell}_{\text{unbatch}}$$

$$\Rightarrow \mathbf{b} = (b_1, b_2, \dots, b_{74}) \rightarrow \mathbf{b}_{\text{unbatch}}, \mathbf{b}_{\text{batch}}$$

$$\Rightarrow \mathbf{e} = (e_1, e_2, \dots, e_{74}) \rightarrow \mathbf{e}_{\text{unbatch}}, \mathbf{e}_{\text{batch}}$$

$$\langle \hat{B}^{(1)}, \hat{B}^{(2)}, \dots, \hat{B}^{(8)} \rangle \leftarrow \text{batched}(\langle A^{(1)}, A^{(2)}, \dots, A^{(8)} \rangle, \mathbf{b}_{\text{batch}}, \langle \mathbf{e}_{\text{batch}}^{(1)}, \mathbf{e}_{\text{batch}}^{(2)}, \dots, \mathbf{e}_{\text{batch}}^{(8)} \rangle)$$

↓

$$B^{(k)} \leftarrow \text{unbatched}(\hat{B}^{(k)}, \mathbf{b}_{\text{unbatch}}, \mathbf{e}_{\text{unbatch}}^{(k)}) \text{ for } k = 1, \dots, 8$$

↓

$$B^{(1)}, B^{(2)}, \dots, B^{(8)}$$

Analysis

Example: n of the eight instances meeting $R = \infty$ in an iteration.

Extra-dummy

- ▶ Completes the computation in this iteration of the “for” loop.
- ▶ Later computes n “compensatory” isogenies in the unbatched component.

Extra-infinity

- ▶ Perform infinity-related computations.
⇒ needed by n instances, not needed by $8 - n$ instances.

Conclusion

- ▶ n is small, extra-dummy better
- ▶ n is close to 8, extra-infinity better

The Combined Batching Method

Set a new variable:

$$n_{\text{inf}} = \text{isinfity}(R^{(1)}) + \dots + \text{isinfity}(R^{(8)})$$

Set a precomputed threshold:

$$n_{\text{thld}} = 3$$

```
03 ...
04 if i = 1, ..., n:
09 ...
10    $n_{\text{inf}} = \sum \text{isinfity}(R^{(k)})$ 
11   if  $n_{\text{inf}} \leq n_{\text{thld}}$  then:
12     extra-dummy
13   else:
14     extra-infinity
15   cswap( $T_0, T_1, 1 - s$ )
```

High-Throughput Implementation

Class group action: three different batching methods.

Curve arithmetic: according to x64 software [CCC⁺19] with minor optimizations.

Prime-field operations: (8×1) -way implementation based on the limb-slicing technique [CGT⁺21].

- ▶ \mathbb{F}_p multiplication: experiments with different variants.
- ▶ \mathbb{F}_p squaring: classic optimization (compute the repeated partial products only once, due to $a_i a_j = a_j a_i$).

Field Operation	ISA	Integer Mul/Sqr	Reduction	Structure	Latency
Multiplication	IFMA	Product-Scan.	Operand-Scan.	Interleaved	436 cc
Squaring	IFMA	Product-Scan.	Operand-Scan.	Interleaved	344 cc
Multiplication	AVX-512F	Karatsuba	Product-Scan.	Separated	848 cc
Squaring	AVX-512F	Product-Scan.	Product-Scan.	Interleaved	723 cc

Low-Latency Implementation

Serve as **the unbatched component** in the hybrid mode for high-throughput implementation.

Class group action: according to x64 software [CCC⁺19].

Curve arithmetic: can be parallelized to 2-way, with the cost $iM + jS \rightarrow \frac{i}{2}M^2 + \frac{j}{2}S^2$.

Prime-field operations: (2×4) -way implementation based on the implementation of Orisaka, Aranha, and López [OAL18] (originally designed for SIDH).

- ▶ \mathbb{F}_p multiplication: interleaved version based on [OAL18].
- ▶ \mathbb{F}_p squaring: classic optimization (compute the repeated partial products only once, due to $a_i a_j = a_j a_i$).

Benchmark of CSIDH-512 Group Actions (OAYT)

- ▶ Intel Core i3-1005G1 Ice Lake CPU clocked at 1.2 GHz (turbo boost disabled).
- ▶ Source code: <https://gitlab.uni.lu/APSIA/AVX-CSIDH>.
- ▶ Compiled with GCC 9.3.0.

	Reference	ISA	Impl.	#Inst.	Cycles	Cycles/#inst.	Speedup
	[CLM ⁺ 18] [†]	x64	1-way	1	133.7 M	133.7 M	1.52×
	[OAYT19]	x64	1-way	1	248.4 M	248.4 M	0.82×
	[CCC ⁺ 19]	x64	1-way	1	203.6 M	203.6 M	1.00×
	[CR20]	x64	1-way	1	195.0 M	195.0 M	1.04×
	[HLKA20]	x64	1-way	1	194.7 M	194.7 M	1.05×
This work	Low-Latency	AVX-512F	(2 × 4)-way	1	232.2 M	232.2 M	0.88×
	Extra-Dummy	AVX-512F	(8 × 1)-way	8	858.0 M	107.3 M	1.90×
	Extra-Infinity	AVX-512F	(8 × 1)-way	8	1003.9 M	125.5 M	1.62×
	Combined	AVX-512F	(8 × 1)-way	8	850.1 M	106.3 M	1.92×
This work	Low-Latency	IFMA	(2 × 4)-way	1	132.1 M	132.1 M	1.54×
	Extra-Dummy	IFMA	(8 × 1)-way	8	454.1 M	56.8 M	3.59×
	Extra-Infinity	IFMA	(8 × 1)-way	8	550.5 M	68.8 M	2.96×
	Combined	IFMA	(8 × 1)-way	8	446.9 M	55.9 M	3.64×

[†] Not constant-time.

Benchmark of CSIDH-512 Group Actions (Dummy-Free)

- ▶ Intel Core i3-1005G1 Ice Lake CPU clocked at 1.2 GHz (turbo boost disabled).
- ▶ Source code: <https://gitlab.uni.lu/APSIA/AVX-CSIDH>.
- ▶ Compiled with GCC 9.3.0.


	Reference	ISA	Impl.	#Inst.	Cycles	Cycles/#inst.	Speedup
	[CCC ⁺ 19]	x64	1-way	1	433.3 M	433.3 M	1.00×
	[CR20]	x64	1-way	1	394.3 M	394.3 M	1.10×
This work	Low-Latency	AVX-512F	(2 × 4)-way	1	447.0 M	447.0 M	0.97×
	Extra-Dummy	AVX-512F	(8 × 1)-way	8	1811.0 M	226.4 M	1.91×
	Extra-Infinity	AVX-512F	(8 × 1)-way	8	2172.3 M	271.5 M	1.60×
	Combined	AVX-512F	(8 × 1)-way	8	1801.4 M	225.2 M	1.92×
This work	Low-Latency	IFMA	(2 × 4)-way	1	253.8 M	253.8 M	1.71×
	Extra-Dummy	IFMA	(8 × 1)-way	8	967.0 M	120.9 M	3.58×
	Extra-Infinity	IFMA	(8 × 1)-way	8	1220.5 M	152.6 M	2.84×
	Combined	IFMA	(8 × 1)-way	8	955.3 M	119.4 M	3.63×

Conclusions

- ▶ Vector engines like AVX-512 offer great potential to optimize CSIDH.
- ▶ The first vectorized implementation of CSIDH.
- ▶ 3.6-fold throughput gain compared to the state-of-the-art x64 implementation.
- ▶ Vectorizing methods can also be used for larger primes, and certain parts of the code can be reused.

Thank you!

References I

-  Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren.
Csi-fish: Efficient isogeny based signatures through class group computations.
In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 227–247. Springer International Publishing, 2019.
-  Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny.
Quantum circuits for the csidh: Optimizing quantum evaluation of isogenies.
In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 409–441. Springer International Publishing, 2019.
-  Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith.
Stronger and faster side-channel protections for CSIDH.
In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology – LATINCRYPT 2019*, pages 173–193. Springer International Publishing, 2019.

References II

-  Hao Cheng, Johann Großschädl, Jiaqi Tian, Peter B. Rønne, and Peter Y. A. Ryan.
High-throughput elliptic curve cryptography using avx2 vector instructions.
In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O’Flynn, editors, *Selected Areas in Cryptography – SAC 2020*, pages 698–719. Springer International Publishing, 2021.
-  Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes.
Csidh: An efficient post-quantum commutative group action.
In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 395–427. Springer International Publishing, 2018.
-  Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez.
Optimal strategies for CSIDH.
Cryptology ePrint Archive, Report 2020/417, 2020.
<https://eprint.iacr.org/2020/417>.

References III

-  Luca De Feo and Steven D. Galbraith.
SeaSign: Compact isogeny signatures from class group actions.
In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 759–789. Springer International Publishing, 2019.
-  Aaron Hutchinson, Jason LeGrow, Brian Koziel, and Reza Azarderakhsh.
Further optimizations of csidh: A systematic approach to efficient strategies, permutations, and bound vectors.
In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *Applied Cryptography and Network Security*, pages 481–501. Springer International Publishing, 2020.
-  David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson.
Sike – supersingular isogeny key encapsulation.
Available for download at <https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/submissions/SIKE-Round3.zip>, 2020.

References IV

-  Amir Jalali, Reza Azarderakhsh, Mehran Mozaffari Kermani, and David Jao.
Towards optimized and constant-time csidh on embedded devices.
In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 215–231. Springer International Publishing, 2019.
-  Michael Meyer, Fabio Campos, and Steffen Reith.
On lions and elligators: An efficient constant-time implementation of csidh.
In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, pages 307–325.
Springer International Publishing, 2019.
-  Kaushik Nath and Palash Sarkar.
Efficient 4-way vectorizations of the Montgomery ladder.
Cryptology ePrint Archive, Report 2020/378, 2020.
<https://eprint.iacr.org/2020/378>.

References V

-  Gabriell Orisaka, D. Aranha, and J. López.
Finite field arithmetic using avx-512 for isogeny-based cryptography.
In XVIII Simpósio Brasileiro de Segurança da Informação e Sistemas Computacionais (SBSeg 2018),
pages 49–56, 2018.
-  Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi.
A faster constant-time algorithm of csidh keeping two points.
Cryptology ePrint Archive, Report 2019/353, 2019.
<https://eprint.iacr.org/2019/353>.
-  Jacques Vélu.
Isogénies Entre Courbes Elliptiques.
Comptes Rendus de l'Académie des Sciences de Paris, 273:238–241, 1971.