CTIDH: Faster constant-time CSIDH

Gustavo Banegas Daniel J. Bernstein Fabio Campos Tung Chou
Tanja Lange Michael Meyer Benjamin Smith **Jana Sotáková**

Affiliations and more at https://ctidh.isogeny.org/

September 7, 2021

CTIDH: Faster constant-time CSIDH

CSIDH [CLM+18]

is a post-quantum isogeny-based non-interactive key exchange protocol.

It uses a group action on a certain set of elliptic curves.

- Secret keys sampled from some keyspace $\mathsf{sk} \in \mathcal{K}$ give group elements,
- Public keys are elliptic curves obtained by evaluating the group action $\star$

$$\mathsf{pk} = \mathsf{sk} \star E$$

CTIDH: Faster constant-time CSIDH

CSIDH [CLM+18]

is a post-quantum isogeny-based non-interactive key exchange protocol.

It uses a group action on a certain set of elliptic curves.

- Secret keys sampled from some keyspace $\mathsf{sk} \in \mathcal{K}$ give group elements,
- Public keys are elliptic curves obtained by evaluating the group action $\star$

$$\mathsf{pk} = \mathsf{sk} \star \mathsf{E}$$

CTIDH

is a new keyspace and a new constant-time algorithm for the group action in CSIDH.

- constant-time claims verified using `valgrind`
- speedups compared to previous best work:

  `CSIDH-512`: 438006 multiplications (best previous 789000)
  125.53 million Skylake cycles (best previous more than 200 million).

Background on CSIDH

CSIDH [CLM+18]

Start with a prime $p = 4\ell_1 \cdot \ell_n - 1$ with ℓ_i small primes.

There is a abelian group G acting on a set of elliptic curves $\mathcal{E} = \{E/\mathbb{F}_p : \#E(\mathbb{F}_p) = p + 1\}$, represented in Montgomery form

$$E_A : y^2 = x^3 + Ax^2 + x \quad \text{for some } A \in \mathbb{F}_p^* \setminus \{\pm 2\}$$

For every $\ell_i \mid p + 1$, we have a group element $g_i \in G$ with efficient action via isogenies:

$$E_{A'} = g_i \star E_A. \qquad \longleftrightarrow \qquad \phi : E_A \to E_{A'} \quad \ell_i\text{-isogeny.}$$

Secret keys $(e_1, \ldots, e_n) \in \mathbb{Z}^n$; public keys

$$E_{A'} = \left(\prod_{i=1}^{n} g_i^{e_i} \right) \star E_A.$$

Constant-time evaluation

Constant-time evaluation of the group action

If the input is a CSIDH curve and a private key, and the output is the result of the CSIDH action, then the algorithm time provides no information about the private key, and provides no information about the output.

Secret keys $(e_1, \ldots, e_n) \in \mathbb{Z}^n$; public keys

$$E_{A'} = \left(\prod_{i=1}^{n} g_i^{e_i} \right) \star E_A.$$

Keyspace

Goal

For $(e_1, \ldots, e_n) \in \mathbb{Z}^n$, evaluate the group action

$$E_{A'} = \left(\prod_{i=1}^{n} g_i^{e_i} \right) \star E_A.$$

- Exponent vectors $(e_1, \ldots, e_n)$ sampled from some keyspace $\mathcal{K} \subset \mathbb{Z}^n$;
- Large enough keyspace: $\#\mathcal{K} \approx 2^{256}$;

Keyspace

Goal

For $(e_1, \ldots, e_n) \in \mathbb{Z}^n$, evaluate the group action

$$E_{A'} = \left(\prod_{i=1}^{n} g_i^{e_i} \right) \star E_A.$$

- Exponent vectors $(e_1, \ldots, e_n)$ sampled from some keyspace $\mathcal{K} \subset \mathbb{Z}^n$;
- Large enough keyspace: $\#\mathcal{K} \approx 2^{256}$;

Examples of keyspaces

1. Original CSIDH [CLM+18]: $|e_i| \leq m$ for all i with $(2m+1)^n \approx 2^{256}$,
2. [MCR19] use $0 \leq e_i \leq 10$ for `CSIDH-512`;
3. [CDRH20] allow the m_i to vary for efficiency.

Batching

The batching idea

`CSIDH-512` prime $p = 4 \cdot 3 \cdot 5 \cdot \cdots \cdot 373 \cdot 578 - 1$.

Batching

The batching idea

CSIDH-512 prime $p = 4 \cdot 3 \cdot 5 \cdot \cdots \cdot 373 \cdot 578 - 1$.

We start with the exponent vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$:

primes	3	5	7	11	13	17	19	23	29	31	...
exponent vector	1	-2	0	3	-1	1	0	2	-1	0	...

Batching

The batching idea

CSIDH-512 prime $p = 4 \cdot 3 \cdot 5 \cdot \cdots \cdot 373 \cdot 578 - 1$.

We start with the exponent vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$.

Now we split the primes into batches:

primes	{ 3	5	7 }	{ 11	13	17	19 }	{ 23	29	31 }	...
exponent vector	1	-2	0	3	-1	1	0	2	-1	0	...

Batching

The batching idea

CSIDH-512 prime $p = 4 \cdot 3 \cdot 5 \cdot \dots \cdot 373 \cdot 578 - 1$.
We start with the exponent vector $(e_1, \dots, e_n) \in \mathbb{Z}^n$.
Now we group the entries in the exponent vector isogenies per batch:

primes	{ 3	5	7 }	{ 11	13	17	19 }	{ 23	29	31 }	...
exponent vector	1	-2	0	3	-1	1	0	2	-1	0	...
per batch		3			5				3		

exponent vector $(e_1, \dots, e_n) \in \mathbb{Z}^n$ comes from the subset in which we compute

- 3 $\{3, 5, 7\}$-isogenies,
- 5 $\{11, 13, 17, 19\}$-isogenies,
- and 3 $\{23, 29, 31\}$-isogenies.

Batching

The batching idea

CSIDH-512 prime $p = 4 \cdot 3 \cdot 5 \cdot \cdots \cdot 373 \cdot 578 - 1$.
We start with the exponent vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$.
Now we group the entries in the exponent vector isogenies per batch:

primes	{ 3	5	7 }	{ 11	13	17	19 }	{ 23	29	31 }	...
exponent vector	1	-2	0	3	-1	1	0	2	-1	0	...
per batch		3			5				3		

exponent vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$ comes from the subset in which we compute

- up to 3 $\{3, 5, 7\}$-isogenies,
- up to 5 $\{11, 13, 17, 19\}$-isogenies,
- and up to 3 $\{23, 29, 31\}$-isogenies.

Batching

The batching idea

CSIDH-512 prime $p = 4 \cdot 3 \cdot 5 \cdot \cdots \cdot 373 \cdot 578 - 1$.

Now we group the isogenies per batch:

primes	{ 3	5	7 }	{ 11	13	17	19 }	{ 23	29	31 }	...
exponent vector	1	-2	0	3	-1	1	0	2	-1	0	...
per batch		3			5				3		

Batching Keyspace

For B batches: For $N \in \mathbb{Z}^B_{>0}$ and $m \in \mathbb{Z}^B_{\geq 0}$, we define

$$\mathcal{K}_{N,m} := \left\{(e_1, \ldots, e_n) \in \mathbb{Z}^n \mid \textstyle\sum_{j=1}^{N_i} |e_{i,j}| \leq m_i \text{ for } 1 \leq i \leq B\right\}.$$

Isogeny magic

In CSIDH, start with prime $p = 4\ell_1 \dots \ell_n - 1$ for ℓ_i small odd primes.

Group action

For every $\ell_i \mid p+1$, we have an element g_i that we can act with using ℓ_i-isogenies:

$$E_{A'} = g_i \star E_A$$

Isogeny magic

In CSIDH, start with prime $p = 4\ell_1 \dots \ell_n - 1$ for ℓ_i small odd primes.

Group action

For every $\ell_i \mid p+1$, we have an element g_i that we can act with using ℓ_i-isogenies:

$$E_{A'} = g_i \star E_A$$

Group action via isogenies

Replace the group element g_i

$$g_i : E_A \mapsto E_{A'}$$

Isogeny magic

In CSIDH, start with prime $p = 4\ell_1 \ldots \ell_n - 1$ for ℓ_i small odd primes.

Group action

For every $\ell_i \mid p + 1$, we have an element g_i that we can act with using ℓ_i-isogenies:

$$E_{A'} = g_i \star E_A$$

Group action via isogenies

Replace the group element g_i with an ℓ_i-isogeny ϕ:

$$\phi : E_A \to E_{A'}$$

Isogenies are algebraic group homomorphisms of elliptic curves

$$\phi : y^2 = x^3 + Ax^2 + x \longrightarrow y^2 = x^3 + A'x^2 + x$$
$$(x, y) \longmapsto (f(x, y), g(x, y)) \qquad f, g \text{ rational functions over } \mathbb{F}_p$$

Isogeny magic

In CSIDH, start with prime $p = 4\ell_1 \dots \ell_n - 1$ for ℓ_i small odd primes.

Group action

For every $\ell_i \mid p+1$, we have an element g_i that we can act with using ℓ_i-isogenies:

$$E_{A'} = g_i \star E_A$$

Group action via isogenies

Replace the group element g_i with an ℓ_i-isogeny ϕ:

$$\phi : E_A \to E_{A'}$$

Isogenies are algebraic group homomorphisms of elliptic curves:

$$P \in E_A \longmapsto \phi(P) \in E_{A'}$$
$$\text{order } \ell_i N \longrightarrow \text{order } N.$$

Computing the group action

Computing the action by $g_i \leftrightarrow \ell_i$

Simplified algorithm to compute the group action $E_{A'} = g_i \star E_A$:

Computing the group action

Computing the action by $g_i \leftrightarrow \ell_i$

Simplified algorithm to compute the group action $E_{A'} = g_i \star E_A$:

1. find a point P of order ℓ_i on E_A:
   1.1 generate a point T of order $p+1$ on E_A,
   1.2 multiply $P = [\frac{p+1}{\ell_i}]T$.
2. Compute the ℓ_i-isogeny $\phi : E_A \to E_{A'}$ with kernel P:
   2.1 enumerate the multiples $[i]P$ of the point P for $i \in S$, with $S = \{1, 2, \ldots, \frac{\ell-1}{2}\}$ [Vél71] or $S = \{1, 3, 5, \ldots, \ell - 2\}$ [BDFLS20],
   2.2 construct a polynomial $h(X) = \prod_{i \in S}(x - x([i]P))$,
   2.3 Compute the coefficient A' from $h(X)$.

Computing the group action

Computing the action by $g_i \leftrightarrow \ell_i$

Simplified algorithm to compute the group action $E_{A'} = g_i \star E_A$:

1. find a point P of order ℓ_i on E_A:
   1.1 generate a point T of order $p+1$ on E_A,
   1.2 multiply $P = [\frac{p+1}{\ell_i}]T$. Costs $\approx 10 \log_2(p)$ mult in $\mathbb{F}_p$.
2. Compute the ℓ_i-isogeny $\phi : E_A \to E_{A'}$ with kernel P: Cost $\leq 6\ell_i$ mult in $\mathbb{F}_p$
   2.1 enumerate the multiples $[i]P$ of the point P for $i \in S$,
   with $S = \{1, 2, \ldots, \frac{\ell-1}{2}\}$ [Vél71] or $S = \{1, 3, 5, \ldots, \ell-2\}$ [BDFLS20],
   2.2 construct a polynomial $h(X) = \prod_{i \in S}(x - x([i]P))$,
   2.3 Compute the coefficient A' from $h(X)$.

Amortize the cost

Exponent vector $(1, 1, 1, 0, \ldots, 0)$

We compute ℓ_i-isogenies for $\ell_1 = 3$ and $\ell_2 = 5$ and $\ell_3 = 7$:

Amortize the cost

Exponent vector $(1, 1, 1, 0, \ldots, 0)$

We compute ℓ_i-isogenies for $\ell_1 = 3$ and $\ell_2 = 5$ and $\ell_3 = 7$:

1. Find a suitable point:
   1.1 Generate a random point T of order $p + 1$,
   1.2 Compute $T_1 = \left[\frac{p+1}{3\cdot5\cdot7}\right] T$ has exact order $3 \cdot 5 \cdot 7$,
2. Compute the isogenies:
   2.1 3-isogeny:
      2.1.1 Compute $P_1 = [5 \cdot 7] T_1$ has order 3,
      2.1.2 Use P_1 to construct 3-isogeny ϕ_1,
      2.1.3 Point $T_2 = \phi_1(T_1)$ has order $5 \cdot 7$ on the new curve,
   2.2 5-isogeny:
      2.2.1 Compute $P_2 = [7] T_2$ has order 5,
      2.2.2 Construct 5-isogeny ϕ_2 with kernel P_2,
      2.2.3 The point $T_3 = \phi_2(T_2)$ has order 7 on the new curve,
   2.3 7-isogeny: construct the isogeny ϕ_3 with kernel $P_3 = T_3$.

Towards atomic blocks

Exponent vector $(1, 0, 1, 0, \ldots, 0)$

We compute ℓ_i-isogenies for $\ell_1 = 3$ and $\ell_3 = 7$ but no 5-isogeny:

Towards atomic blocks

Exponent vector $(1, 0, 1, 0, \ldots, 0)$

We compute ℓ_i-isogenies for $\ell_1 = 3$ and $\ell_3 = 7$ but no 5-isogeny:

1. Find a suitable point:
   - 1.1 Generate a random point T of order $p + 1$,
   - 1.2 Compute $T_1 = \left[\frac{p+1}{3 \cdot 5 \cdot 7}\right] T$ has exact order $3 \cdot 5 \cdot 7$,
2. Compute the isogenies:
   - 2.1 3-isogeny:
     - 2.1.1 Compute $P_1 = [5 \cdot 7] T_1$ has order 3,
     - 2.1.2 Use P_1 to construct 3-isogeny ϕ_1,
     - 2.1.3 Point $T_2 = \phi_1(T_1)$ has order $5 \cdot 7$ on the new curve,
   - 2.2 No 5-isogeny:
     - 2.2.1 Compute the isogeny as before but throw away the results,
     - 2.2.2 Adjust to code to always compute $[5] T_2$,
     - 2.2.3 The point $T_3 = [5] T_2$ has order 7 on the same curve,
   - 2.3 7-isogeny: construct the isogeny ϕ_3 with kernel $P_3 = T_3$.

Atomic blocks

Definition (Atomic Blocks, simplified)

Let $I = (I_1, \ldots, I_k) \in \mathbb{Z}^k$ be such that $1 \leq I_1 < I_2 < \cdots < I_k \leq n$.
An *atomic block* of length k is a probabilistic algorithm α_I taking inputs A and $\epsilon \in \{0,1\}^k$ and returning $A' \in \mathbb{F}_p$ such that $E_{A'} = (\prod_i g_{I_i}^{\epsilon_i}) \star E_A$, satisfying

- there is a function τ such that, for each (A, ϵ) the distribution of the time taken by α_I, given that A' is returned by α_I on input (A, ϵ), is $\tau(I)$.

Atomic blocks

Definition (Atomic Blocks, simplified)

Let $I = (I_1, \dots, I_k) \in \mathbb{Z}^k$ be such that $1 \leq I_1 < I_2 < \cdots < I_k \leq n$.
An *atomic block* of length k is a probabilistic algorithm α_I taking inputs A and $\epsilon \in \{0,1\}^k$ and returning $A' \in \mathbb{F}_p$ such that $E_{A'} = (\prod_i g_{I_i}^{\epsilon_i}) \star E_A$, satisfying

- there is a function τ such that, for each (A, ϵ) the distribution of the time taken by α_I, given that A' is returned by α_I on input (A, ϵ), is $\tau(I)$.

Evaluating $3, 5,$ and 7-isogeny

On the previous slide, we saw an atomic block α_I with $I = (1, 2, 3)$ that computes

$$E_{A'} = g_1^{\epsilon_1} g_2^{\epsilon_2} g_3^{\epsilon_3} \star E_A$$

for $(\epsilon_1, \epsilon_2, \epsilon_3) \in \{0,1\}^3$ without leaking timing information about $(\epsilon_1, \epsilon_2, \epsilon_3)$.

Atomic blocks for batches

Atomic blocks for batches

Suppose we have batches $\{3, 5, 7\}, \{11, 13, 17\}, \ldots$ And we want to compute one 5-isogeny and one 11-isogeny, i.e. exponent vector $(0, 1, 0, 1, 0, 0, 0, \ldots)$

Atomic blocks for batches

Atomic blocks for batches

Suppose we have batches $\{3,5,7\}, \{11,13,17\}, \ldots$ And we want to compute one 5-isogeny and one 11-isogeny, i.e. exponent vector $(0,1,0,1,0,0,0,\ldots)$

1. Find a suitable point:
   - 1.1 Generate a random point T of order $p+1$,
   - 1.2 Compute $T_1 = \left[\frac{p+1}{(3\cdot5\cdot7)(11\cdot13\cdot17)}\right] T$ has order $(3\cdot5\cdot7)(11\cdot13\cdot17)$.
2. Compute the isogenies:
   - 2.1 $\{3,5,7\}$-isogeny:
     - 2.1.1 Compute $P_1 = [(11\cdot13\cdot17)]T_1$ has order $(3\cdot5\cdot7)$,
     - 2.1.2 Use $[3\cdot7]P_1$ of order 5 to construct 5 -isogeny ϕ_1,
     - 2.1.3 Point $T_2 = [3\cdot7]\phi_1(T_1)$ has order $11\cdot13\cdot17$ on the new curve,
   - 2.2 $\{11,13,17\}$-isogeny:
     - 2.2.1 Compute $P_2 = [13\cdot17]T_2$ has order 11,
     - 2.2.2 Construct 11-isogeny ϕ_2 with kernel P_2.

Atomic blocks for batches

Atomic blocks for batches

Suppose we have batches $\{3, 5, 7\}, \{11, 13, 17\}, \ldots$ And we want to compute one 5-isogeny and one 11-isogeny, i.e. exponent vector $(0, 1, 0, 1, 0, 0, 0, \ldots)$

1. Find a suitable point:
   1.1 Generate a random point T of order $p + 1$,
   1.2 Compute $T_1 = \left[\frac{p+1}{(3\cdot5\cdot7)(11\cdot13\cdot17)}\right] T$ has order $(3 \cdot 5 \cdot 7)(11 \cdot 13 \cdot 17)$.
2. Compute the isogenies:
   2.1 $\{3, 5, 7\}$-isogeny:
      2.1.1 Compute $P_1 = [(11 \cdot 13 \cdot 17)] T_1$ has order $(3 \cdot 5 \cdot 7)$,
      2.1.2 Use $[3 \cdot 7] P_1$ of order 5 to construct 5-isogeny ϕ_1,
      2.1.3 Point $T_2 = [3 \cdot 7]\phi_1(T_1)$ has order $11 \cdot 13 \cdot 17$ on the new curve,
   2.2 $\{11, 13, 17\}$-isogeny:
      2.2.1 Compute $P_2 = [13 \cdot 17] T_2$ has order 11,
      2.2.2 Construct 11-isogeny ϕ_2 with kernel P_2.

Matryoshka isogeny

How to construct the isogeny with the same code for all primes in the batch:

Matryoshka isogeny

How to construct the isogeny with the same code for all primes in the batch:

Matryoshka Isogeny for the batch $\{11, 13, 17\}$

Compute the 11-isogeny

1. enumerate the multiples $[i]P$ of the point P for $i \in S$, with $S = \{1, 2, \ldots, 5\}$
2. construct $h(X) = \prod_{i=1}^{5}(x - x([i]P))$,
3. Compute the coefficient A' from $h(X)$.

Matryoshka isogeny

How to construct the isogeny with the same code for all primes in the batch:

Matryoshka Isogeny for the batch $\{11, 13, 17\}$

Compute the ~~11~~ 13-isogeny

1. enumerate the multiples $[i]P$ of the point P for $i \in S$,
   with $S = \{1, 2, \ldots, 5, 6\}$
2. construct $h(X) = \prod_{i=1}^{5}(x - x([i]P)) \cdot (x - x([6]P))$,
3. Compute the coefficient A' from $h(X)$.

Matryoshka isogeny

How to construct the isogeny with the same code for all primes in the batch:

Matryoshka Isogeny for the batch $\{11, 13, 17\}$

Compute the ~~11~~ ~~13~~ 17-isogeny

1. enumerate the multiples $[i]P$ of the point P for $i \in S$,
   with $S = \{1, 2, \ldots, 5, 6, 7, 8\}$
2. construct $h(X) = \prod_{i=1}^{5}(x - x([i]P)) \cdot (x - x([6]P)) \cdot (x - x([7]P))(x - x([8]P))$,
3. Compute the coefficient A' from $h(X)$.

Matryoshka isogeny

With small overhead and dummy operations, we can compute the isogeny for any prime in the batch with the same code at the cost of computing isogeny for the largest prime.
Known for Vélu formulas [BLMP19], new for $\sqrt{\text{é}}$lu from [BDFLS20], newly used for batching.

Selection of the parameters

Evaluation cost function

Greedy algorithm to find efficient batching:

- For every batch configuration (number of batches, bounds of each batch), we can estimate the cost of the group action evaluation.
- Adaptively change batch configuration to find one with smaller cost (and large enough keyspace).

batch	size	primes	bound
1	2	3, 5	10
2	3	7, 11, 13	14
3	4	17, 19, 23, 29	16
4	4	31, 37, 41, 43	17
5	5	47, 53, 59, 61, 67	17
6	5	71, 73, 79, 83, 89	17
7	6	97, 101, 103, 107, 109, 113	18
8	7	127, 131, 137, 139, 149, 151, 157	18
9	7	163, 167, 173, 179, 181, 191, 193	18
10	8	197, 199, 211, 223, 227, 229, 233, 239	18
11	8	241, 251, 257, 263, 269, 271, 277, 281	18
12	6	283, 293, 307, 311, 313, 317	13
13	8	331, 337, 347, 349, 353, 359, 367, 373	13
14	1	587	1

valgrind constant time verification

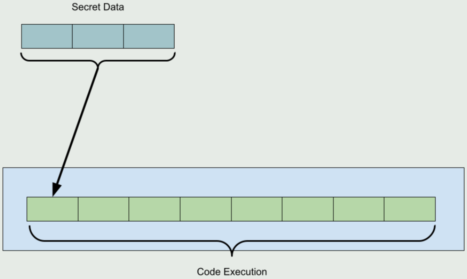
Valgrind

Checking for constant-time

- We "poison" the secret data, that is, we put an undefined value;
- *valgrind* will check if the undefined data corrupts branches or indices.

How to use *valgrind* to check sensitive data

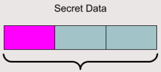
Correct flow without “poisoning”:



How to use *valgrind* to check sensitive data

Poisoning secret data

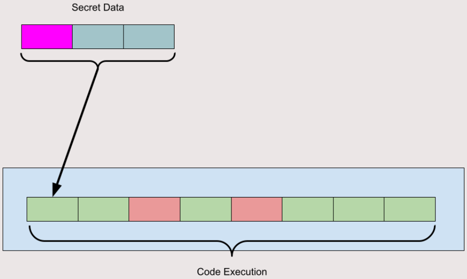
we poison the secret data with “undefined” value



How to use *valgrind* to check sensitive data

Checking for inconsistencies

We check where "undefined" value impacts in the code execution



high-ctidh

we used *valgrind* to check if poisoning the secret data generates leaks of sensitive information such as timing.

Speedups, comparison to previous works

pub	priv	DH	Mcyc	**M**	**S**	**a**	$1, 1, 0$	$1, 0.8, 0.05$	
512	220	1	89.11	228780	82165	346798	310945	311852	new
512	220	1	190.92	447000	128000	626000	575000	580700	[CCJR20]
512	220	2	93.23	238538	87154	361964	325692	326359	new
512	256	1	125.53	321207	116798	482311	438006	438762	new
512	256	1	—	624000	165000	893000	789000	800650	[ACR20]
512	256	2	129.64	330966	121787	497476	452752	453269	new
512	256	2	218.42	665876	189377	691231	855253	851939	[CDRH20]
512	256	2	238.51	632444	209310	704576	841754	835121	[HLKA20]
512	256	2	239.00	657000	210000	691000	867000	859550	[CCC+19]
512	256	2	—	732966	243838	680801	976804	962076	[OAYT19]
512	256	2	395.00	1054000	410000	1053000	1464000	1434650	[MCR19]
1024	256	1	469.52	287739	87944	486764	375683	382432	new
1024	256	1	—	552000	133000	924000	685000	704600	[ACR20]
1024	256	2	511.19	310154	99371	521400	409525	415721	new

Table: **pub**: size of p; **priv**: size of the keyspace; **DH** 1: group action evaluation, **DH** 2: group action evaluation and public key validation; **Mcyc** millions of cycles on a 3GHz Intel Xeon E3-1220 v5 (Skylake) CPU with Turbo Boost disabled; "**M**" multiplications; "**S**" squarings; "**a**" additions; "$1, 1, 0$" and "$1, 0.8, 0.05$" combinations of **M**, **S**, and **a**.

Summary

CTIDH

- New keyspace for CSIDH,
- New constant-time algorithm to evaluate the group action in CSIDH,
- Formalization of atomic blocks to compute the isogeny group action,
- constant-time verification using `valgrind`,
- speed records,

Find the article and the code at

https://ctidh.isogeny.org/

References I

Gora Adj, Jesús-Javier Chi-Domínguez, and Francisco Rodríguez-Henríquez.
On new Vélu's formulae and their applications to CSIDH and B-SIDH constant-time implementations, 2020.
https://eprint.iacr.org/2020/1109.

Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith.
Faster computation of isogenies of large prime degree, 2020.
https://eprint.iacr.org/2020/341.

Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny.
Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies.
2019.
https://eprint.iacr.org/2018/1059.

References II

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, 2019.
https://eprint.iacr.org/2019/837.

Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez.
The SQALE of CSIDH: square-root Vélu quantum-resistant isogeny action with low exponents, 2020.
https://eprint.iacr.org/2020/1520.

Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez.
Optimal strategies for CSIDH, 2020.
https://eprint.iacr.org/2020/417.

References III

Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes.
CSIDH: an efficient post-quantum commutative group action, 2018.
https://eprint.iacr.org/2018/383.

Aaron Hutchinson, Jason T. LeGrow, Brian Koziel, and Reza Azarderakhsh.
Further optimizations of CSIDH: A systematic approach to efficient strategies, permutations, and bound vectors, 2020.
https://eprint.iacr.org/2019/1121.

Michael Meyer, Fabio Campos, and Steffen Reith.
On Lions and Elligators: An efficient constant-time implementation of CSIDH, 2019.
https://eprint.iacr.org/2018/1198.

References IV

Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi.
(Short paper) A faster constant-time algorithm of CSIDH keeping two points, 2019.
https://eprint.iacr.org/2019/353.

Jacques Vélu.
Isogénies entre courbes elliptiques, 1971.
https://gallica.bnf.fr/ark:/12148/cb34416987n/date.