SIBC: A Python-3 library for designing and implementing efficient isogeny-based protocols

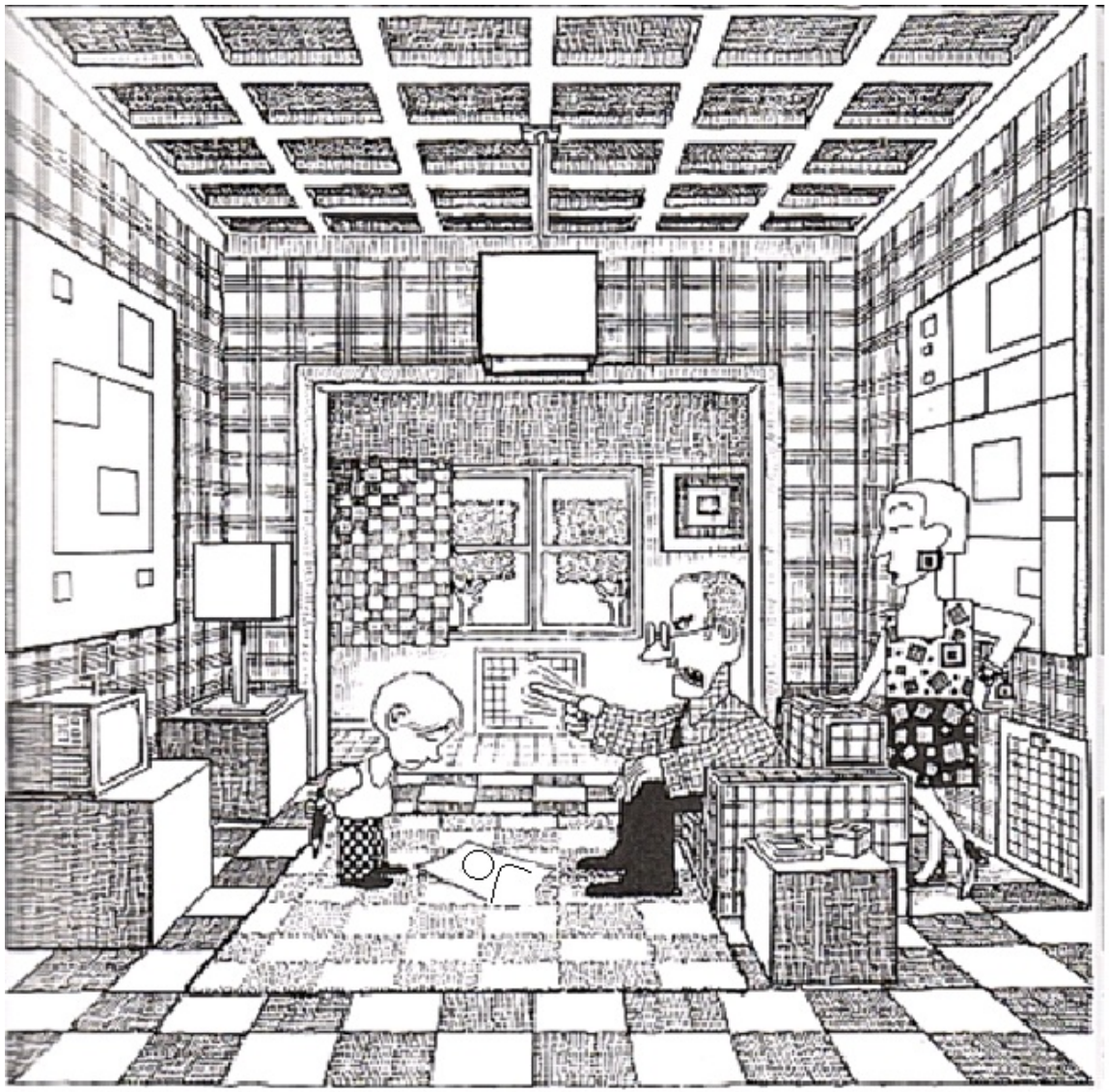
Gora Adj, Jesús-Javier Chi-Domínguez and
Francisco Rodríguez-Henríquez

Isogeny-based cryptography school, september.23.2021

The SIBC library

- Isogeny-Based Cryptographic constructions (SIBC) is a a Python-3 library that implements a variety of isogeny-based constructions and protocols
- SIBC supports Vélu and $\sqrt{}$élu for isogeny arithmetic, optimal strategies and other isogeny building blocks
- SIBC supports the CSIDH, B-SIDH, SIKE and *B-SIKE* key exchange protocols
- SIBC aims to provide fellow isogenistas with an agile design tool for constructing and testing new isogeny-based primitives, while keeping a constant-time execution of their procedures.
- SIBC is publicly available at: https://github.com/JJChiDguez/sibc/

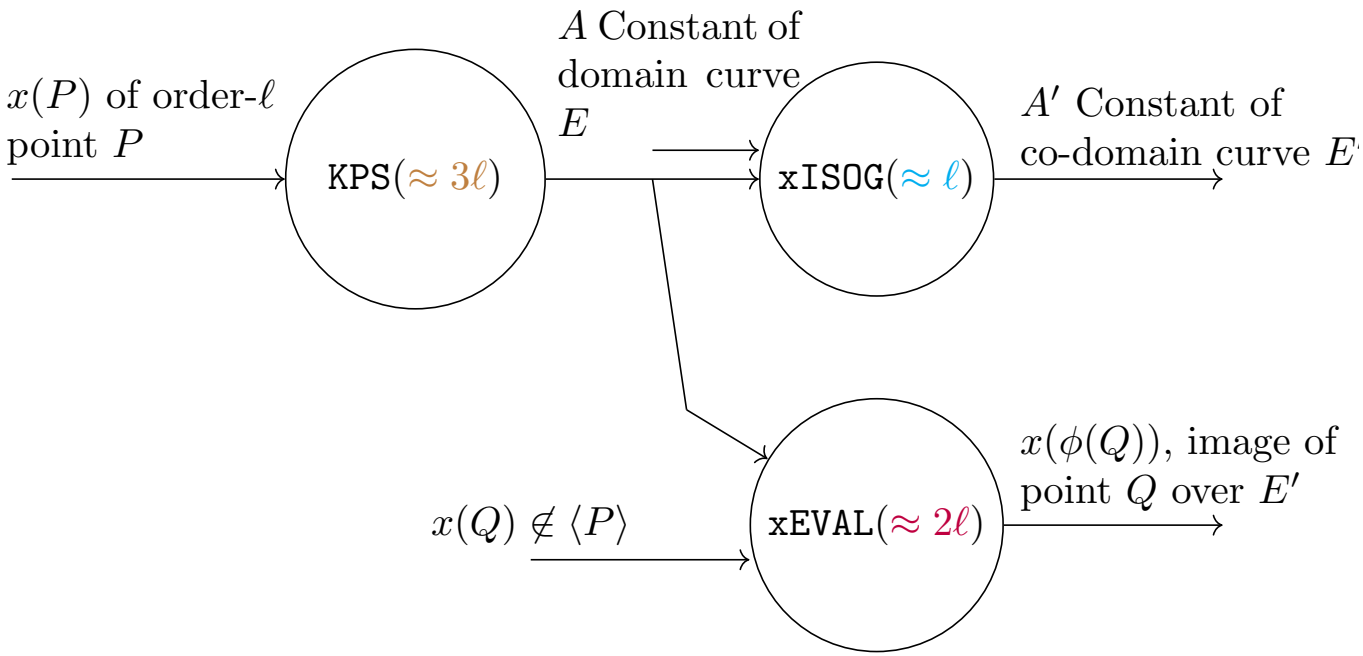
Context



Computing degree-ℓ isogenies using Vélu'

- Since July 1971, Vélu has been used to construct and evaluate degree-ℓ isogenies in the vast majority of isogeny-based cryptographic primitives.
- Vélu has a combined computational expense of about 6ℓ field operations to construct and evaluate degree-ℓ isogenies.

Computing degree-ℓ isogenies using Vélu



- Vélu uses three main blocks,
  - KPS [Sort of a pre-computation building block. Cost: $\approx (3\ell)\mathbf{M}$]
  - xISOG [Finds the image curve. Cost: $\approx (\ell)\mathbf{M}$]
  - xEVAL [Evaluate a point. Cost: $\approx (2\ell)\mathbf{M}$]

Computing degree-ℓ isogenies using $\sqrt{}$élu's formulas

- Recently, Bernstein, de Feo, Leroux and Smith presented in ANTS'2020 [BFLS ANTS'20], a new approach for computing degree-ℓ isogenies at a reduced cost of just $\tilde{O}(\sqrt{\ell})$ field operations.
- This improvement was obtained by observing that the polynomial product embedded in the isogeny computations could be speedup via a baby-step giant-step method

Computing degree-ℓ isogenies using $\sqrt{}$élu's formulas: KPS

Algorithm 1 KPS: Baby-step Giant-step method

Input: An elliptic curve $E_A/\mathbb{F}_q : y^2 = x^3 + Ax^2 + x$; $P \in E_A(\mathbb{F}_q)$ of odd prime order ℓ.
Output: $\mathcal{I} = \{x([i]P) \mid i \in I\}$, $\mathcal{J} = \{x([j]P) \mid j \in J\}$, and $\mathcal{K} = \{x([k]P) \mid k \in K\}$ such that (I, J) is an index system for S, and $K = S \backslash (I \pm J)$

1. $b \leftarrow \lfloor \sqrt{\ell - 1}/2 \rfloor$; $b' \leftarrow \lfloor (\ell - 1)/4b \rfloor$
2. $I \leftarrow \{2b(2i + 1) \mid 0 \leq i < b'\}$
3. $J \leftarrow \{2j + 1 \mid 0 \leq j < b\}$
4. $K \leftarrow S \backslash (I \pm J)$
5. $\mathcal{I} \leftarrow \{x([i]P) \mid i \in I\}$
6. $\mathcal{J} \leftarrow \{x([j]P) \mid j \in J\}$
7. $\mathcal{K} \leftarrow \{x([k]P) \mid k \in K\}$
8. **return** $\mathcal{I}, \mathcal{J}, \mathcal{K}$

- **Time cost**: $\approx 3b$ Point Additions $= 18b\mathbf{M}$
- **Memory cost**: $\leq 8b$ Field elements.

Computing degree-ℓ isogenies using $\sqrt{\text{élu}}$: xISOG

Algorithm 2 Computing xISOG

Input: An elliptic curve $E_A/\mathbb{F}_q : y^2 = x^3 + Ax^2 + x$; $P \in E_A(\mathbb{F}_q)$ of odd prime order ℓ; $\mathcal{I}, \mathcal{J}, \mathcal{K}$ from KPS.

Output: $A' \in \mathbb{F}_q$ such that $E_{A'}/\mathbb{F}_q : y^2 = x^3 + A'x^2 + x$ is the image curve of a separable isogeny with kernel $\langle P \rangle$.

1. $h_I \leftarrow \prod_{x_i \in \mathcal{I}}(Z - x_i)) \in \mathbb{F}_q[Z]$
2. $E_{0,J} \leftarrow \prod_{x_j \in \mathcal{J}} \left(F_0(Z, x_j) + F_1(Z, x_j) + F_2(Z, x_j)\right) \in \mathbb{F}_q[Z]$
3. $E_{1,J} \leftarrow \prod_{x_j \in \mathcal{J}} \left(F_0(Z, x_j) - F_1(Z, x_j) + F_2(Z, x_j)\right) \in \mathbb{F}_q[Z]$
4. $R_0 \leftarrow \mathtt{Res}_Z(h_I, E_{0,J}) \in \mathbb{F}_q$
5. $R_1 \leftarrow \mathtt{Res}_Z(h_I, E_{1,J}) \in \mathbb{F}_q$
6. $M_0 \leftarrow \prod_{x_k \in \mathcal{K}}(1 - x_k) \in \mathbb{F}_q$
7. $M_1 \leftarrow \prod_{x_k \in \mathcal{K}}(-1 - x_k) \in \mathbb{F}_q$
8. $d \leftarrow \left(\frac{A-2}{A+2}\right)^{\ell} \left(\frac{M_0 R_0}{M_1 R_1}\right)^8$
9. **return** $2\left(\frac{1+d}{1-d}\right)$

- **Time cost**:

$$\text{Cost}(b) = 19b^{\log_2(3)} + 4b\log_2(b) + \frac{77}{3}b + 3\log_2(b) + \frac{58}{3}.$$

- **Memory cost**: $\leq 3b\log_2 b$ field elements.[shared with xEVAL]

Computing degree-ℓ isogenies using $\sqrt{\text{élu}}$: xEVAL

Algorithm 3 Computing xEVAL

Input: An elliptic curve $E_A/\mathbb{F}_q : y^2 = x^3 + Ax^2 + x$; $P \in E_A(\mathbb{F}_q)$ of odd prime order ℓ; the x-coordinate $\alpha \neq 0$ of a point $Q \in E_A(\mathbb{F}_q)\backslash\langle P\rangle$; $\mathcal{I}$, $\mathcal{J}$, $\mathcal{K}$ from KPS.

Output: The x-coordinate of $\phi(Q)$, where ϕ is a separable isogeny with kernel $\langle P\rangle$.

1. $h_I \leftarrow \prod_{x_i \in \mathcal{I}}(Z - x_i)) \in \mathbb{F}_q[Z]$
2. $E_{0,J} \leftarrow \prod_{x_j \in \mathcal{J}} \left(F_0(Z, x_j)/\alpha^2 + F_1(Z, x_j)/\alpha + F_2(Z, x_j)\right) \in \mathbb{F}_q[Z]$
3. $E_{1,J} \leftarrow \prod_{x_j \in \mathcal{J}} \left(F_0(Z, x_j)\alpha^2 + F_1(Z, x_j)\alpha + F_2(Z, x_j)\right) \in \mathbb{F}_q[Z]$
4. $R_0 \leftarrow \mathtt{Res}_Z(h_I, E_{0,J}) \in \mathbb{F}_q$
5. $R_1 \leftarrow \mathtt{Res}_Z(h_I, E_{1,J}) \in \mathbb{F}_q$
6. $M_0 \leftarrow \prod_{x_k \in \mathcal{K}}(1/\alpha - x_k) \in \mathbb{F}_q$
7. $M_1 \leftarrow \prod_{x_k \in \mathcal{K}}(\alpha - x_k) \in \mathbb{F}_q$
8. **return** $(M_0 R_0)^2/(M_1 R_1)^2$

- **Time cost**:

$$\text{Cost}(b) = 19b^{\log_2(3)} + 4b\log_2(b) + \frac{77}{3}b + 3\log_2(b) + \frac{58}{3}\,.$$

- **Memory cost**: $\leq 3b\log_2 b$ field elements.[shared with xEVAL]

Two isogeny-based protocols

$\sqrt{}$élu impact on isogeny-based protocols

- What is the expected impact of $\sqrt{}$élu for SIDH or SIKE?

$\sqrt{}$élu impact on isogeny-based protocols

- What is the expected impact of $\sqrt{}$élu for SIDH or SIKE?
  None

$\sqrt{}$élu impact on isogeny-based protocols

- What is the expected impact of $\sqrt{}$élu for CSIDH?

$\sqrt{}$élu impact on isogeny-based protocols

- What is the expected impact of $\sqrt{}$élu for CSIDH?
  We reported significant accelerations for CSIDH-1792 using $\sqrt{}$élu in https://eprint.iacr.org/2020/1109
  Banegas-Bernstein-Campos-Chou-Lange-Meyer-Smith-Sotáková reported significant accelerations for CSIDH-512 CSIDH-1024 using $\sqrt{}$élu in https://eprint.iacr.org/2021/633

$\sqrt{}$élu impact on isogeny-based protocols

- What is the expected impact of $\sqrt{}$élu for B-SIDH?

$\sqrt{}$élu impact on isogeny-based protocols

- What is the expected impact of $\sqrt{}$élu for B-SIDH?
  Huge. B-SIDH is the big winner among the isogeny-based protocols

Overviewing the CSIDH [Castryck-Lange-Martindale-Panny-Renes Asiacrypt'18]

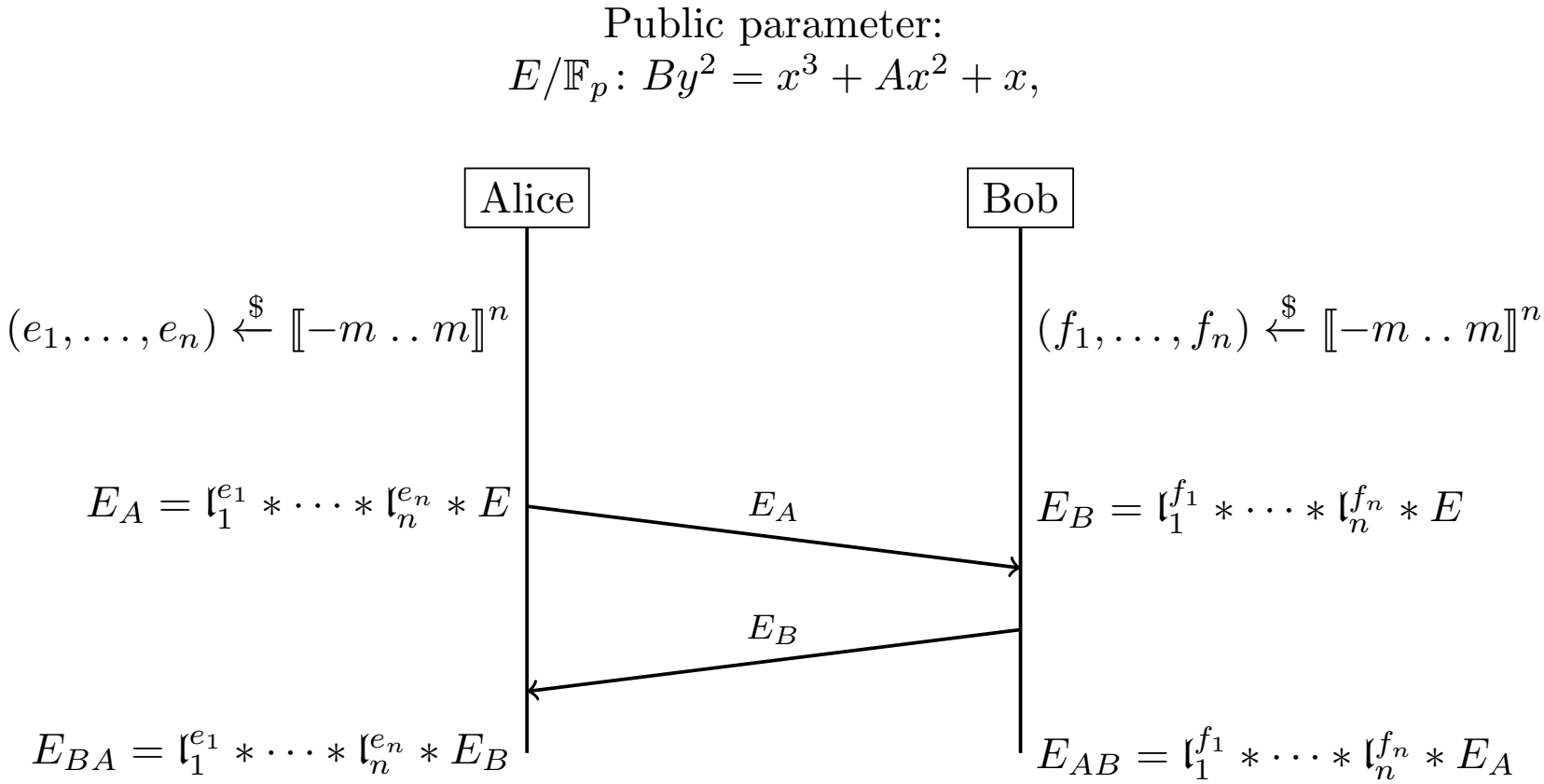


Figure: CSIDH key-exchange protocol

CSIDH works over a finite field $\mathbb{F}_p$, where p is a prime of the form

$$p := 4 \prod_{i=1}^{n} \ell_i - 1$$

Overviewing the CSIDH [Castryck-Lange-Martindale-Panny-Renes Asiacrypt'18]

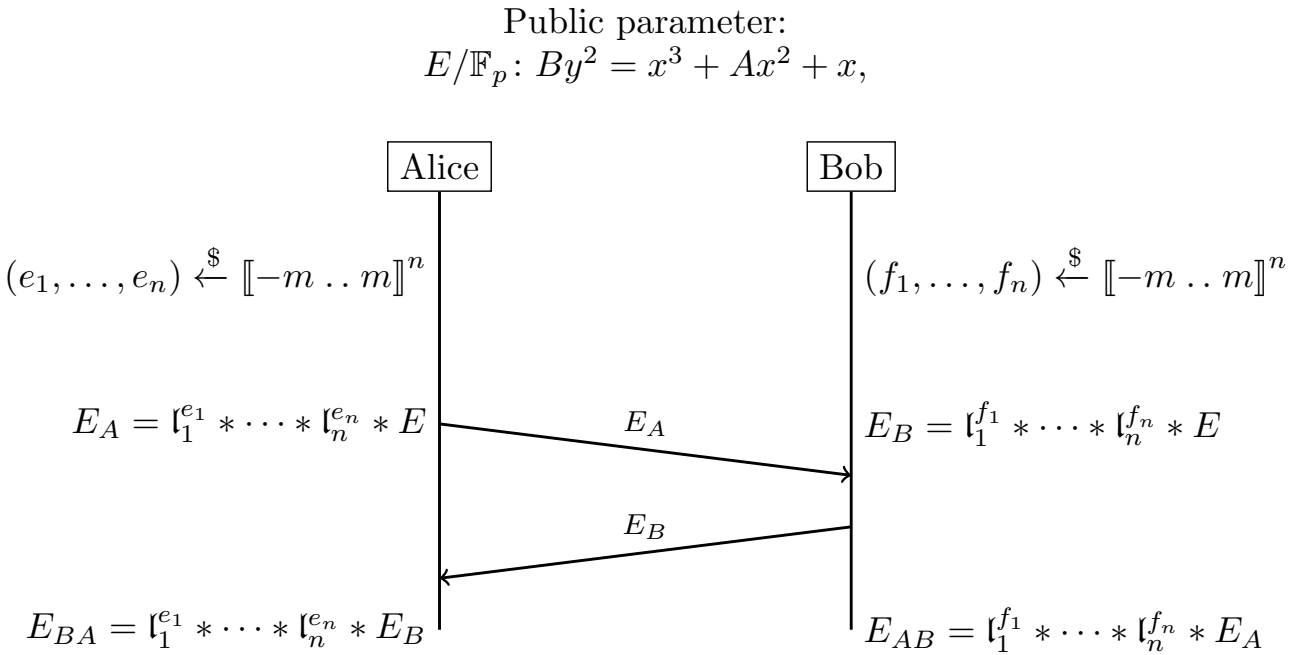


Figure: CSIDH key-exchange protocol

$$(p+1)/4 = 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 \cdot 59 \cdot 61 \cdot 67 \cdot 71 \cdot 73 \cdot 79 \cdot 83 \cdot \mathbf{89} \cdot 97 \cdot 101 \cdot 103 \cdot$$
$$107 \cdot 109 \cdot 113 \cdot 127 \cdot 131 \cdot 137 \cdot 139 \cdot 149 \cdot 151 \cdot 157 \cdot 163 \cdot 167 \cdot 173 \cdot 179 \cdot 181 \cdot 191 \cdot 193 \cdot 197 \cdot 199 \cdot 211 \cdot 223 \cdot$$
$$227 \cdot 229 \cdot 233 \cdot 239 \cdot 241 \cdot 251 \cdot 257 \cdot 263 \cdot 269 \cdot 271 \cdot 277 \cdot 281 \cdot 283 \cdot 293 \cdot 307 \cdot 311 \cdot 313 \cdot 317 \cdot 331 \cdot 337 \cdot 347 \cdot$$
$$349 \cdot 353 \cdot 359 \cdot 367 \cdot 373 \cdot \mathbf{587}$$

Overviewing the CSIDH [Castryck-Lange-Martindale-Panny-Renes Asiacrypt'18]

```
from sibc.csidh import CSIDH, default_parameters
csidh = CSIDH(**default_parameters)

# alice generates a key
alice_secret_key = csidh.secret_key()
alice_public_key = csidh.public_key(alice_secret_key)

# bob generates a key
bob_secret_key = csidh.secret_key()
bob_public_key = csidh.public_key(bob_secret_key)

# if either alice or bob use their secret key with the other's respective
# public key, the resulting shared secrets are the same
shared_secret_alice = csidh.dh(alice_secret_key, bob_public_key)
shared_secret_bob = csidh.dh(bob_secret_key, alice_public_key)

# Alice and bob produce an identical shared secret
assert shared_secret_alice == shared_secret_bob
```

Figure: Executing the CSIDH key exchange using SIBC

Overviewing the CSIDH [Castryck-Lange-Martindale-Panny-Renes Asiacrypt'18]

```
# CSIDH
# A single random instances of a key exchange
sibc -p p512 -f hvelu -a csidh -s df -e 10 csidh-main
sibc -p p512 -f svelu -a csidh -s df -e 10 -m csidh-main
sibc -p p512 -f tvelu -a csidh -s df -e 10 -t csidh-main
sibc -p p512 -f hvelu -a csidh -s df -e 10 -m -t csidh-main
```

Figure: Executing a random dummy-free CSIDH key exchange using a 512-bit prime p, a selection/combination of Vélu and Vélu sqrt for the isogeny computations and an integer range for the secret exponents between $[-10, 10]$.

Playing the B-SIDH [Costello Asiacrypt'20]

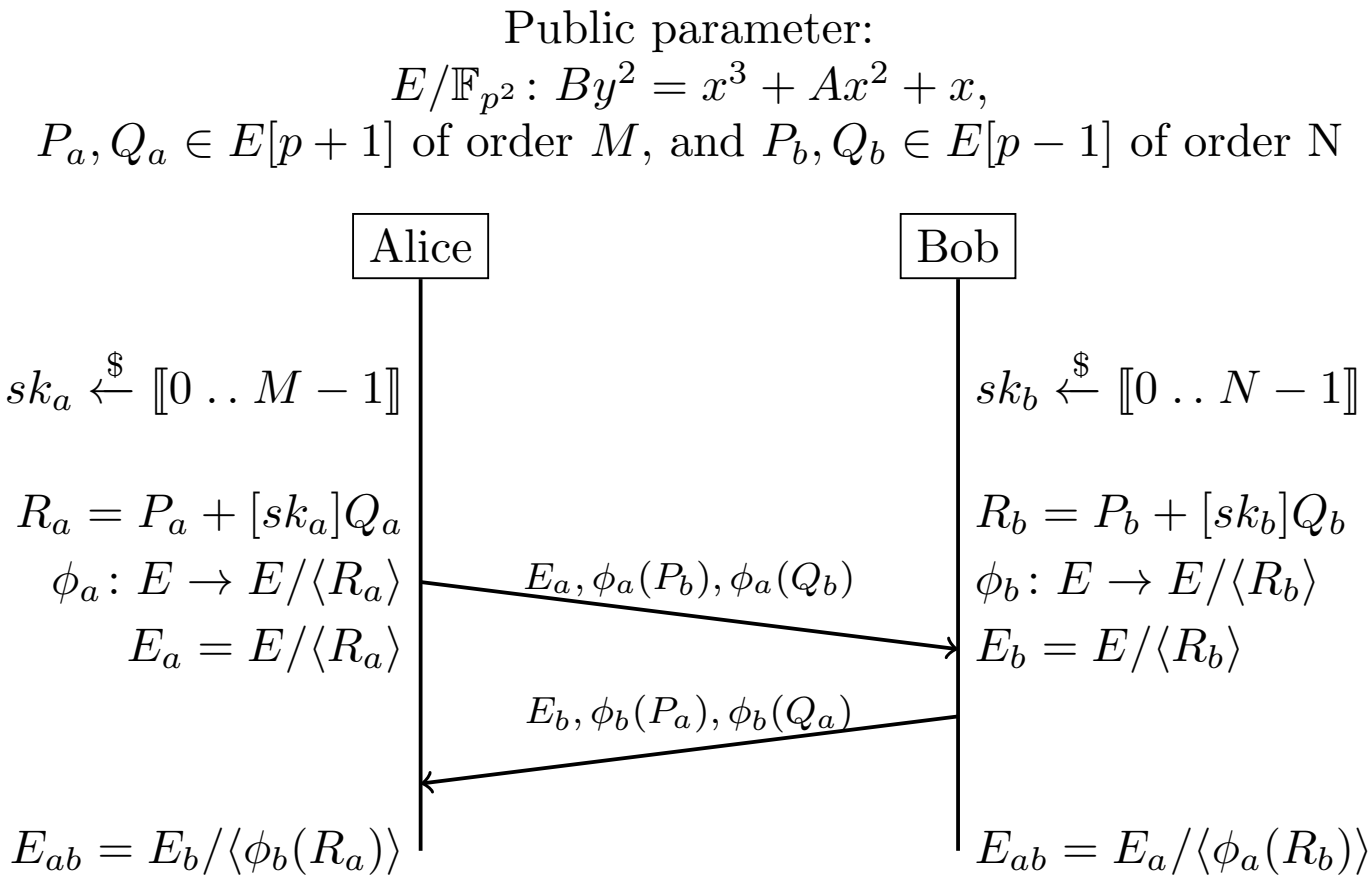


Figure: B-SIDH protocol for a prime p such that $M|(p+1)$ and $N|(p-1)$.

Alice and Bob work in the $(p+1)$- and $(p-1)$-torsion of a set of supersingular curves defined over $\mathbb{F}_{p^2}$ and the set of their quadratic twist, respectively.

Playing the B-SIDH [Costello Asiacrypt'20]

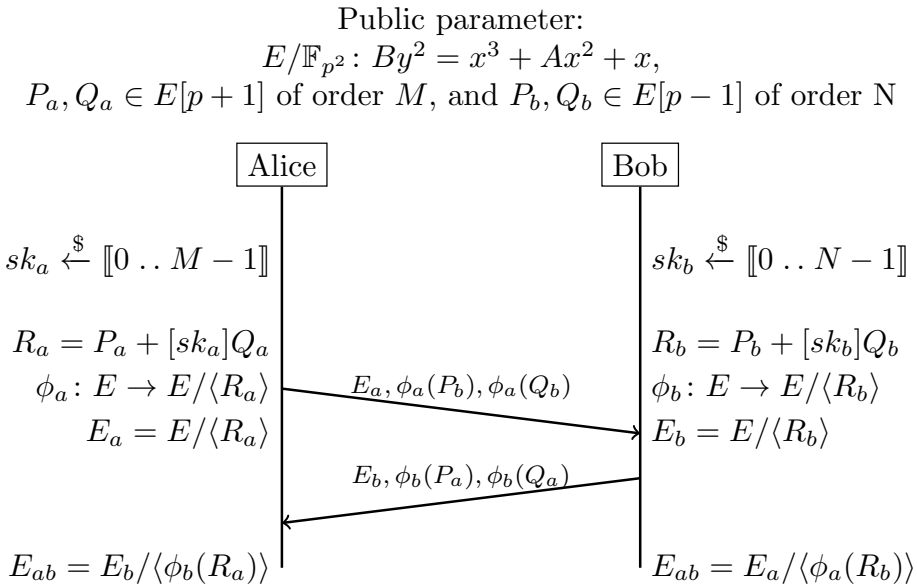


Figure: B-SIDH protocol for a prime p such that $M|(p+1)$ and $N|(p-1)$.

Prime example: **B-SIDHp237**:

$$p = \texttt{0x1B40F93CE52A207249237A4FF37425A798E914A74949FA343E8EA487FFFF}$$

$$M = 4^3 \cdot \left(4 \cdot 3^4 \cdot 17 \cdot 19 \cdot 31 \cdot 37 \cdot 53^2\right)^6,$$

$$N = 7 \cdot 13 \cdot 43 \cdot 73 \cdot 103 \cdot 269 \cdot 439 \cdot 881 \cdot 883 \cdot 1321 \cdot 5479 \cdot 9181 \cdot 12541 \cdot 15803 \cdot 20161 \cdot 24043 \cdot 34843 \cdot 48437 \cdot 62753 \cdot 72577.$$

Playing the B-SIDH [Costello Asiacrypt'20]

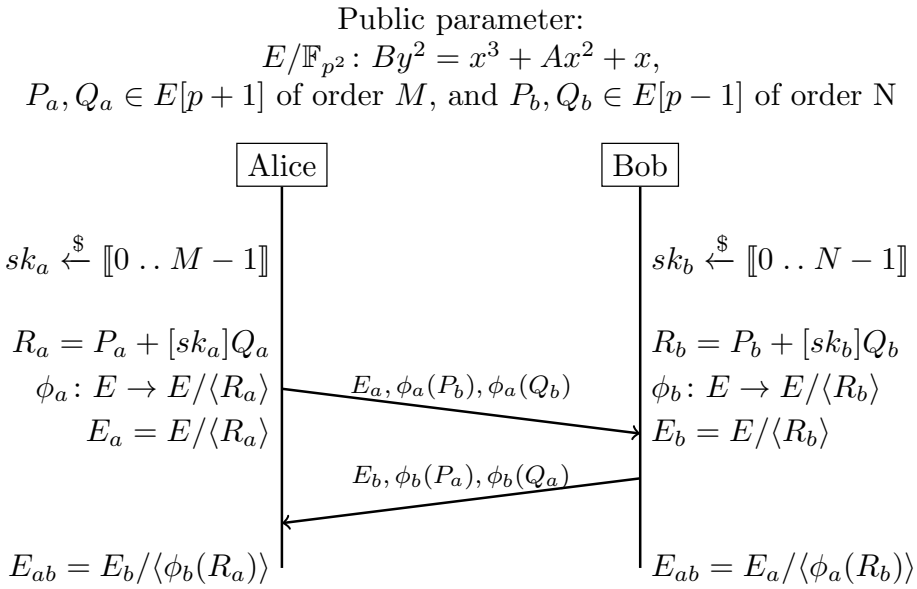


Figure: B-SIDH protocol for a prime p such that $M|(p+1)$ and $N|(p-1)$.

Prime example: **B-SIDHp253**:

$$p = \texttt{0x1935BECE108DC6C0AAD0712181BB1A414E6A8AAA6B510FC29826190FE7EDA80F},$$
$$M = 4^2 \cdot 3 \cdot 7^{16} \cdot 17^9 \cdot 31^8 \cdot 311 \cdot 571 \cdot 1321 \cdot 5119 \cdot 6011 \cdot 14207 \cdot 28477 \cdot 76667,$$
$$N = 11^{18} \cdot 19 \cdot 23^{13} \cdot 47 \cdot 79 \cdot 83 \cdot 89 \cdot 151 \cdot 3347 \cdot 17449 \cdot 33461 \cdot 51193.$$

Playing the B-SIDH [Costello Asiacrypt'20]

```
from sibc.bsidh import BSIDH, default_parameters
bsidh = BSIDH(**default_parameters)
sk_a, pk_a = bsidh.keygen_a()
sk_b, pk_b = bsidh.keygen_b()
ss_a, ss_b = bsidh.derive_a(sk_a, pk_b), bsidh.derive_b(sk_b, pk_a)
ss_a == ss_b
```

Figure: Executing the B-SIDH key exchange using SIBC

B-SIKE

```
from sibc.sidh import SIKE, default_parameters
sike = SIKE(**default_parameters)
s, sk3, pk3 = sike.KeyGen()
c, K = sike.Encaps(pk3)
K_ = sike.Decaps((s, sk3, pk3), c)
K == K_

sike503 = SIKE('montgomery', 'p503', False, False)
s, sk3, pk3 = sike503.KeyGen()
c, K = sike503.Encaps(pk3)
K_ = sike503.Decaps((s, sk3, pk3), c)
K == K_
```

Figure: We adopt the name B-SIKE for the combination of B-SIDH plus a key encapsulation mechanism. Executing a B-SIKE key exchange using SIBC and the prime $SIKE_p503$

Skylake Clock cycle timings for several key exchange isogeny-based protocols

Implementation	Protocol Instantiation	Mcycles
SIKE [NIST alternative candidate]	SIKEp434	22
Castryck *et al.* [Original CSIDH]	CSIDH-512 unprotected	4×155
Bernstein *et al.* [Original $\sqrt{}$élu]	CSIDH-512 unprotected	4×153
	CSIDH-1024 unprotected	4×760
Cervantes-Vázquez *et al.* [LC'19 CSIDH imp]	CSIDH-512	4×238
Chi-Domínguez *et al.* [CSIDH with strategies]	CSIDH-512	4×230
Hutchinson *et al.* [CSIDH with strategies]	CSIDH-512	4×229
Banegas *et al.* [CTIDH]	CTIDH-512	4×126
*This work (***estimated***)*	CSIDH-512	4×223
	B-SIDH-p253	119

Table: Skylake Clock cycle timings for a key exchange protocol for different instantiations of the SIDH, CSIDH, and B-SIDH protocols.

Programming exercises [from easiest to more involved]

0. Provide large primes with a bitlength greater than two thousand bits for implementing large instances of CSIDH
1. Modify SIBC to compute $\sqrt{}$élu using Toom-Cook instead of Karatsuba.
2. Use SIBC to accurately estimate the computational cost of a parallel implementation of $\sqrt{}$élu.
3. Use SIBC to accurately estimate the computational cost of a parallel implementation of B-SIDH and CSIDH using $\sqrt{}$élu and hvelu (as defined in the library).
4. Modify SIBC to implement CTIDH.
5. Modify SIBC to implement the Verifiable Delay function of De Feo-Masson-Petit-Sanso.
6. Modify SIBC to implement any isogeny-based digital signature scheme of your choice

Thanks