Elliptic curves, number theory and cryptography

5. handin – Isogenies, Pairings

Aurore Guillevic and Diego F. Aranha

Aarhus University

April 21, 2022

Due date: May 6, 4pm GMT+2 (16:00 Aarhus time)

The corresponding lecture materials are at
`https://www.hyperelliptic.org/tanja/teaching/isogeny-school21/`
and Lorenz Panny materials at
`https://yx7.cc/docs/misc/isog_bristol_notes.pdf`

Question 1 is the follow-up of Question 5 in hand-in 4.

Question 1 (SIKE on a toy-example)**.** Let $p = 431$ and note that $p + 1 = 432 = 2^4 \cdot 3^3$. The curve $E_0 \colon y^2 = x^3 + x$ is a supersingular curve over $\mathbb{F}_p$ and has $p+1$ points. Consider the curve over $\mathbb{F}_{p^2}$ where it has $(p+1)^2$ points.

```
p = 431
Fp = GF(p)
A = Fp(0)
E0 = EllipticCurve(Fp, [1, 0])
E0.is_supersingular()
Fpz.<z> = Fp[]
Fp2.<i> = Fp.extension(z^2+1)
E0p2 = E0.base_extend(Fp2)
r2 = 2^4
r3 = 3^3
assert r2 * r3 == p+1
```

(a) Find a basis of the 2^4-torsion and a basis of the 3^3-torsion subgroups, *i.e.*, find points $P \in E(\mathbb{F}_p)$ and $Q \in E(\mathbb{F}_{p^2})$ of order 2^4 such that $\langle P \rangle \cap \langle Q \rangle = \mathcal{O}$ and points $R \in E(\mathbb{F}_p)$ and $S \in E(\mathbb{F}_{p^2})$ of order 3^3 such that $\langle R \rangle \cap \langle S \rangle = \mathcal{O}$.
Hint: You can check this as $[8]P \neq [8]Q$ and $[9]R \neq \pm[9]S$.
Hint: For the 3^3 torsion points, you can also use how the negative direction is defined for CSIDH to find the independent points.

(b) Alice and Bob.
Compute a generator $P_a \in E(\mathbb{F}_{p^2})$ for the kernel of Alice's isogeny, where $P_a = P + [a]Q$ and a is a random integer in $\{1, \ldots, 2^4 - 1\}$.
Compute a generator $P_b \in E(\mathbb{F}_{p^2})$ for the kernel of Bob's isogeny, where $P_b = R + [b]S$ and b is a random integer in $\{1, \ldots, 3^3 - 1\}$.
Check that P_a has order 2^4 and P_b has order 3^3 (without using `.order()`). If not, it means you have a problem with your basis, go back to Question (a).

(c) Isogenies of Alice and Bob.
With the function `phiA = E0p2.isogeny(Pa)`, compute Alice's isogeny ϕ_a and the isogenous curve E_a with `phiA.codomain()`.
Do the same for Bob with Bob's generator: compute Bob's isogeny ϕ_b and the isogenous curve E_b.

(d) Compute the image of P and Q under ϕ_b, then compute Alice's $\phi_b(P_a) = \phi_b(P) + [a]\phi_b(Q)$ in E_b.
Compute the image of R and S under ϕ_a, then compute Bob's $\phi_a(P_b) = \phi_a(R) + [b]\phi_a(S)$ in E_a.

(e) Compute the second part of the commutative diagram:
- compute an isogeny of kernel $\phi_b(P_a)$ from E_b, and the image curve E_{ba}.
- compute an isogeny of kernel $\phi_a(P_b)$ from E_a, and the image curve E_{ab}.

Check that the j-invariants of E_{ab} and E_{ba} are equal.

Solution 1. See the SageMath code.

In (a) it is important to cast P in $E(\mathbb{F}_{p^2})$ (the quadratic extension), with

```
E0p2(8*P2) != 8*Q2
E0p2(9*P3) != 9*Q3
```

In (d), it is important to note that Alice cannot compute Bob's kernel directly with the formula $\phi_a(P_b)$ because P_b is a secret value known of Bob only. On Bob's side, he cannot compute Alice's kernel generator $\phi_b(P_a)$ because Bob doesn't know P_a. That's why the more compicated formulas of (d) are required: Alice knows a, and Bob provides her with $\phi_b(P), \phi_b(Q)$ (because Alice does not know ϕ_b so she needs to receive the result from Bob). Then she can compute $\phi_b(P) + [a]\phi_b(Q)$ which is equal to $\phi_b(P + [a]Q) = \phi_b(P_a)$ thanks to the homomorphism property of ϕ_b. Alice gives $\phi_a(P), \phi_a(Q)$ to Bob so that he can compute $\phi_a(P) + [b]\phi_a(Q) = \phi_a(P + [b]Q) = \phi_a(P_b)$ on his side.

Question 2. The function `cocks_pinch(l, n, D)` is provided.

Use the Cocks-Pinch method to obtain a pairing-friendly curve of embedding degree $n = 6$ and $D = -3$ from a prime number ℓ of 256 bits.

Use the Cocks-Pinch method to obtain a pairing-friendly curve of embedding degree $n = 8$ and $D = -4$ from a prime number ℓ of 256 bits.

In both cases choose ℓ then run the method. Give ℓ, p, the curve trace t and the curve equation such that E has a subgroup of prime order ℓ and embedding degree n.

Hint: `random_prime(2**256)` returns a random-looking prime of 256 bits.

Solution 2. The SageMath code contains two solutions,

(1) With `random_prime(2**256)`,

(2) With a way to reproduce the random-lookingness of the prime, starting from the decimals of pi. A prime $\ell \equiv 1 \bmod 6$ and such that -3 is a square modulo ℓ from the decimals of pi is $\ell_1 =$31415926535897932384626433832795028841971693993751058209749445923078164062963.

A prime $\ell \equiv 1 \bmod 8$ and such that -1 is a square modulo ℓ from the decimals of pi is $\ell_2 =$ 314159265358979323846264338327950288419716939937510582097494459230781640639 69.

The parameters a, b of the respective elliptic curves are also required. In these two cases it is easy: the first curve has j-invariant 0 because $D = -3$, hence the curve equation is of the form $y^2 = x^3 + b$. The second curve has j-invariant 1728 because $D = -4$, and the curve equation is of the form $y^2 = x^3 + ax$.

Observation: The questions below refer to the slides in Week 10 and SAGE code from Week 11.

Question 3. Using the file `tate_pairing_supersingular_curve.py` from Brightspace, implement Joux's tripartite key agreement in SAGE. The file defines pairing groups for a Type-1 setting using a supersingular curve and a distortion map.

Solution 3. It's about playing the role of Alice, Bob and Charlie. All three sample a random element from $\mathbb{Z}/\ell\mathbb{Z}$ where ℓ is the prime order of the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, not the prime field characteristic p. Let us denote the random elements a, b, and c respectively. Then Alice computes $P_a = [a]P$, Bob $P_b = [b]P$ and Charlie $P_c = [c]P$ and exchange these values. From P_b and P_c, Alice computes $e(P_b, P_c)^a$ (in theory) but from an implementation perspective, we need to map P_c to $\mathbb{G}_2$ first. For that we use the *distorsion map* $\psi\colon (x, y) \mapsto (\omega x, y)$ provided in the SageMath code. Then Alice can compute $e(P_b, \psi(P_c))^a$. Some students chose to compute $e([a]P_b, \psi(P_c))$ and this is correct too, if the choice of $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ is so that the scalar multiplication $[a]P_b$ in $\mathbb{G}_1$ is faster than the exponentiation e^a in $\mathbb{G}_T$, it is appropriate to choose the second formula. Similarly, Bob computes $e(P_a, \psi(P_c))^b$ and Charlie computes $e(P_a, \psi(P_b))^c$. Finally we check that the three values are equal in $\mathbb{F}_{p^2}$.

Question 4. Convert the AKE protocol due to Sakai et al. that we studied in class to the Type-3 setting. Implement it in SAGE using the groups defined in the file `ate_pairing.py` from Brightspace. **Hint:** For hashing to the pairing groups, in this question you can hash to a scalar and multiply the scalar by a generator of the group. The next question will suggest something slightly more sophisticated.

The initial AKE protocol is the following.

Non-interactive identity-based AKE [Sakai et al, 2000]

Initialization: Central authority (PKG) generates master secret key $s \in (\mathbb{Z}/\ell\mathbb{Z})^*$.

Key generation: User with identity ID_i computes $P_i = h(\mathrm{ID}_i)$. PKG generates private key $S_i = [s]P_i$.

Key derivation: Users A and B compute shared key $e(S_A, P_B) = e(S_B, P_A)$.

Solution 4. With a hash function $H\colon \{0,1\}^* \to \mathbb{Z}/\ell\mathbb{Z}$ where ℓ is the order of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, P a generator of $\mathbb{G}_1$, Q of $\mathbb{G}_2$, we obtain
Initialization: Central authority (PKG) generates master secret key $s \in \mathbb{Z}/\ell\mathbb{Z}^*$
Key generation: User with identity ID_i computes $p_i = h(\mathrm{ID}_i)$ then $P_i = [p_i]P$ and $Q_i = [p_i]Q$
PKG generates private key $S_i = [s]P_i$
Key derivation: Users A and B compute shared key $e(S_A, Q_B) = e(S_B, Q_A)$

A possible attack (thanks to Mathias): What if a user j computes $h(\mathrm{ID}_j)$, then invert the value modulo ℓ, and get $[h(\mathrm{ID}_j)^{-1} \bmod \ell]P_j = [s]P$? Then the user j can compute the secret key of another user i with $[h(\mathrm{ID}_i)][1/h(\mathrm{ID}_j) \bmod \ell]P_j = P_i$.
A solution: asymetric protocol wih asymetric pairing (thanks to Marius).
Initialization: Central authority (PKG) generates master key $s \in \mathbb{Z}/\ell\mathbb{Z}^*$
Key generation: User A with identity ID_A computes $P_A = H_1(\mathrm{ID}_i)$ with a hash function H_1 from $\{0,1\}^*$ into $\mathbb{G}_1$. User B with identity ID_B computes $Q_B = H_2(\mathrm{ID}_B)$ with a hash function H_2 from $\{0,1\}^*$ into $\mathbb{G}_2$.
PKG generates private keys $S_A = [s]P_A$ and $S_B = [s]Q_B$
Key derivation: Users A and B compute shared key $e(S_A, Q_B) = e(P_A, S_B)$

Question 5. Convert the BLS short signature scheme that we studied in class to the Type-3 setting and implement it using file `ate_pairing.py` from Brightspace. What are the possible trade-offs in terms of public key and signature size? For simplicity, assume that signed messages are integers in the base field so you can encode them as points by encoding into the x-coordinate and incrementing it until a suitable y satisfying the elliptic curve is found. Some helpful code can be found below:

```
import hashlib

def encode_msg(M, E, A, B, cofactor):
    x = Fp(Integer(hashlib.sha256(M).hexdigest(), 16))
    rhs = x**3 + A*x + B
    while not(rhs.is_square()):
        x += 1
        rhs = x**3 + A*x + B
    y = sqrt(rhs)
    return cofactor * E(x, y)
```

Solution 5. See SageMath code.