Supersingular isogenies Magma tutorial: ECC 2017

Craig Costello

craigco@microsoft.com

1. Choose a prime p between 50 and 100 and construct the field $\mathbb{F}_{p^2}$. Define the set `super_js` as the empty set. Now write a `for` loop that loops over all possible $j \in \mathbb{F}_{p^2}$; inside the loop, use Magma's `EllipticCurveFromjInvariant` find an elliptic curve E that represents the isomorphism class with that j-invariant. Use Magma's `IsSupersingular` function to test if E is supersingular, and if so, use the syntax `Include`($\sim$ `super_js`, j) to include j in the set (also print the group order for each such supersingular curve).

   (a) How many supersingular isomorphism classes are there? Does this agree with the theorem ($\#S_{p^2} = \lfloor p/12 \rfloor + b$ where $b \in \{0, 1, 2\}$) from the slides?
   (b) Find a p in the interval for each $b \in \{0, 1, 2\}$. You may need to use `IrreduciblePolynomial`(Fp2,2) to construct $\mathbb{F}_{p^2}$, depending on your prime p.
   (c) What is the group order for each supersingular curve. How does this relate to p?

2. Choose your favourite prime p between 100 and 200 such that $p \equiv 3 \bmod 4$.

   (a) Using Magma to assist you (and the code written in the previous question), write down the set S_{p^2}, i.e., the set of j-invariants corresponding to isomorphisms in the supersingular isogeny class.
   (b) Each curve in this class will have three points of exact order 2. Use them and Magma's `IsogenyFromKernel` function to draw (by hand) the 3-regular directed regular graph $X(S_{p^2}, 2)$, i.e., the 2-isogeny graph.

3. Now let's do some SIDH key exchange.

   (a) Write a loop to find your favourite SIDH-friendly prime $p = 2^{e_A} \cdot 3^{e_B} - 1$, where $2^{e_A} \approx 3^{e_B}$, e.g., where $\alpha = \log 3^{e_B} / \log 2^{e_A}$ is such that $0.9 < \alpha < 1.1$, and p is at least 500 bits. Define $E_0/\mathbb{F}_{p^2} : y^2 = x^3 + x$ and `assert` that this curve is supersingular. Print `AbelianGroup`(E_0) to see its group structure, and determine the maximum order of any point in E_0.
   (b) We want to find (and fix) two public points P_A and Q_A, such that both have order 2^{e_A} and such that they form a basis for $E_0[2^A]$. To do this, you can use $3^{e_B} *$ `Random`(E_0) to generate random points whose orders are at most 2^{e_A}, but think of a check to ensure that their orders are exactly 2^{e_A} (and don't use the `Order` function, this could take a while). Then, ensure they form a basis for $E_0[2^{e_A}]$, you will need to ensure that the Weil pairing $e(P_A, Q_A, 2^{e_A})$ has exact order 2^{e_A}.
   (c) Now do the analogous to find a basis P_B and Q_B for $E_0[3^{e_B}]$. You now have all the public parameters for SIDH.
   (d) Alice will choose a secret integer $s_A \in [0, 2^{e_A})$ that is a multiple of 2. Bob will choose a secret integer in $s_B = [0, 3^{e_B})$ that is a multiple of 3. Write code that does this. Use `Random`$(0, n)$ to generate a random integer in $[0, n]$. (Side question: why do they need to be multiples?)
   (e) Now comes the hard part(s). We want to write the functions `PublicKeyGen_Alice` and `PublicKeyGen_Bob`. Each will take in the corresponding secret key and the public parameters E_0, P_A, Q_A, P_B and Q_B.
      i. The first step of Alice's (resp. Bob's) public key generation is to compute $S_A = P_A + [s_A]Q_A$ (resp. $S_B = P_B + [s_B]Q_B$). Write code that does this.

ii. Alice's (resp. Bob's) main loop will compute $E_A = E_0/\langle S_A \rangle$ (resp. $E_B = E_0/\langle S_B \rangle$) using the method described in the slides. Write code that does this. In Alice's case, we will want to use 2-isogenies, by calling $E', \phi_2 :=$ `IsogenyFromKernel`$(E, x - x_2)$, where x_2 is the x-coordinate of the point P_2 of order 2 that generates the 2-isogeny $\phi_2 \colon E \to E'$. What does the kernel polynomial look like in Bob's case, i.e., using points of order 3?

iii. The main loop will terminate and output E_A, which is 2^{e_A}-isogenous to E_0, as well as the evaluation of $\phi_A \colon E_0 \to E_A$ at the points P_B and Q_B, i.e., $\phi_A(P_B)$ and $\phi_A(Q_B)$. These evaluations should be carried through the main loop in (ii). Alice's public key is E_A, $\phi_A(P_B)$, and $\phi_A(Q_B)$.

(f) Now we want to write the functions `SharedSecret_Alice` and `SharedSecret_Bob`. Each will take in the corresponding secret key and the other party's public key. The first step is again the scalar multiplication using the points and curve in the public key (and the secret key/scalar), and then the main loop is the same as in key generation. The only difference is you no longer need to pull the additional points through. This function should output the j-invariant (use `jInvariant`) of the final curve.

(g) Run some dummy key exchanges and test that $j(E_{AB}) = j(E_{BA})$.

4. Finally, pair up with someone else (or another group) and decide on some public parameters E_0, P_A, Q_A, P_B and Q_B. Decide who is Alice and who is Bob. Then, generate your public keys (locally, and privately), exchange them over email, and compute the SIDH shared secret. If they are the same, share a gentle fistbump.