Fantastic isogenies and how to find them

Week 6 of the Isogeny-based Cryptography School, originally planned in {2020, Bristol}

Lorenz Panny
Academia Sinica, Taipei, Taiwan

August 9, 2021

1 Different isogeny problems

Almost all constructions in isogeny-based cryptography rely on the (presumed) hardness of various incarnations of *isogeny problems*.[1] The most fundamental one is what I call the pure isogeny problem:

Problem 1. *Let k be a field. Given two elliptic curves E, E' over k that are isogenous over k, compute any isogeny $E \to E'$ defined over k.*

Throughout, we'll be focusing on finding isogenies between *supersingular* elliptic curves, as that's the most relevant setting for contemporary isogeny-based cryptography. By necessity, this implies that the base field will be $\mathbb{F}_{p^2}$, and sometimes even $\mathbb{F}_p$.

1.1 Representation questions

Before diving deeper into the details of how to compute isogenies, we should discuss what it even *means* to "compute" an isogeny.

By definition, an isogeny *is* a rational map; hence, to write down an isogeny of degree d a priori means to write down explicit polynomial expressions

$$\left(\frac{a_d x^d + a_{d-1}x^{d-1} + \cdots + a_1 x + a_0}{c_d x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0},\ y \cdot \frac{b_d x^d + b_{d-1}x^{d-1} + \cdots + b_1 x + b_0}{c'_d x^d + c'_{d-1}x^{d-1} + \cdots + c'_1 x + c'_0}\right)$$

for the isogeny. By definition of the degree, these polynomials have size linear in the degree d. Given that the expected minimal degree of an isogeny between (say) two random supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ is on the order of $\sqrt{p}$, interpreting the word "compute" in this way would vacuously imply that the supersingular isogeny problem is (unconditionally!) average-case exponentially hard, simply because the output is typically exponentially big in the size of the input (e.g., two Weierstraß equations over $\mathbb{F}_{p^2}$).

Luckily, there are other ways of writing down isogenies. The most important one is the technique we've been using all the time: Pick a large but *smooth* degree, and write the isogeny as a composition of small-degree isogenies (which may be stored using explicit polynomials). If d is sufficiently smooth (say, a power of a small prime), this reduces the size from $\Theta(d)$ to $\Theta(\log d)$ — an exponential improvement.

The other cases where we can write down an isogeny in logarithmic space are much less important for our current purposes, but let me mention them anyway for completeness. The first is simply when the isogeny consists of *sparse* polynomials, the only relevant example being the Frobenius isogeny $(x, y) \mapsto (x^p, y^p)$: Storing every coefficient of the polynomial $1 \cdot x^p + 0 \cdot x^{p-1} + 0 \cdot x^{p-2} + \cdots + 0 \cdot x^1 + 0 \cdot x^0$ requires space $\Theta(p)$, but omitting all the zeroes reduces this to $O(1)$. The other example are *sums* of compact isogenies between a fixed pair of curves, which can be stored as *symbolic linear combinations* of those other isogenies even when they have big non-smooth degree. This is obviously not very useful when discussing the isogeny problem since it requires prior knowledge of an isogeny, but it is important for working with endomorphism rings.

In reality, the output of the isogeny problem is always going to be a smooth-degree isogeny.

[1] A prominent exception is the isogeny VDF [4], which relies on the hardness of *evaluating* a known isogeny *faster* than expected.

1.2 Known degrees

The isogeny-finding problems arising from cryptography are most broadly categorized by the kind of information given about the degree of the secret isogeny. In SIKE, for instance, the given curves are known to be connected by an isogeny of degree $\ell^n \approx \sqrt{p}$. This corresponds to an exceptionally short path in the ℓ-isogeny graph: There are no more than $\frac{3}{2}\ell^n$ isogenies of degree ℓ^n emanating from any given curve, but there exist $p/12 + o(1)$ distinct supersingular elliptic curves overall.[2] Thus, isogenies of degree $\ell^n \approx \sqrt{p}$ cannot possibly reach more than a square-root fraction of all isogenous curves — an exponentially sparse subset! It should not be surprising that having this kind of intel about the input curves can allow us to find an isogeny much faster than in the general case. It therefore makes sense to define the degree-N isogeny problem:

Problem 2. *Let k be a field. Given a positive integer N and two elliptic curves E, E' over k that are N-isogenous over k, compute an isogeny $E \to E'$ defined over k of degree N.*

Interestingly, this problem can be both easier and harder than Problem 1: On one hand, we have additional information about the isogeny we are looking for, but on the other hand our set of outputs has been restricted. It usually depends on further details of the situation whether Problem 2 is easier or harder than Problem 1; see for example Exercise 5.1.

1.3 The SIDH situation

The core of the SIDH concept is publishing not only the codomain of a secret isogeny, but also its image on a few points: Alice's public data is a tuple $(E, \varphi(P), \varphi(Q))$ where $\varphi\colon E_0 \to E$ is a secret isogeny and P, Q are publicly known generators of an m-torsion subgroup $E_0[m]$.

If computing discrete logarithms in $E_0[m]$ is easy (such as when m is smooth, or when quantum), it is not hard to see that this data suffices to recover the action of φ on the entirety of $E_0[m]$: Write a given point $R \in E_0[m]$ as a $\mathbb{Z}$-linear combination $[a]P + [b]Q$ and compute $\varphi(R) = [a]\varphi(P) + [b]\varphi(Q)$ using the fact that φ is a group homomorphism. One way to interpret this is that $(P, Q, \varphi(P), \varphi(Q))$ encodes the *restriction* of φ to the subgroup $E_0[m]$. In a sense, it is a description of the *matrix* of the mapping $\varphi|_{E_0[m]}$ induced by φ on m-torsion subgroups.[3]

The bottom line is that the isogeny-finding problem in SIDH is a stricter version of Problem 2: In addition to the degree, we are given the restriction of the desired isogeny to a huge subgroup; in turn, on the output side there's the additional constraint that the isogeny maps P and Q correctly. The extra input may help attackers: Christophe's part on *torsion-point attacks*[4] this week shows how to exploit this information in some cases, which (un)fortunately do not include SIKE.

Problem 3. *Let k be a field. Given positive integers N, m and two elliptic curves E, E' over k such that $E[m] \subseteq E(k)$, and also given an efficient description of $\varphi|_{E[m]}$ where $\varphi\colon E \to E'$ is an isogeny of degree N, compute an isogeny matching these constraints.*

For SIKE, it seems the best we can do (according to current knowledge) is actually to simply ignore the extra information and solve a generic isogeny-finding problem. Interestingly, while breaking SIKE a priori does need "the" correct isogeny, it has been shown that finding *any* isogeny is sufficient [9, 5] when the endomorphism ring of E_0 is known. In other words, [9] reduces Problem 3 to either Problem 1 or 2 in the SIKE setting.

1.4 The CSIDH situation

It's not extremely obvious whether the security of the CSIDH group action actually is equivalent to an isogeny-finding problem. Clearly, a CSIDH private key defines an isogeny, but does every isogeny

[2] Recall that over $\mathbb{F}_{p^2}$, every supersingular ℓ-isogeny graph is connected.

[3] Recall that the m-torsion of an elliptic curve is isomorphic to $\mathbb{Z}/m \times \mathbb{Z}/m$ when m is not divisible by the characteristic.

[4] I'm not too happy with this name since it's really about exploiting knowledge of the restriction of a secret isogeny, rather than the points themselves. Moreover, over a finite field every point is torsion, so "torsion-point attack" is identical to "point attack" which again doesn't make it clear what that is about. Alas, apparently that's what we call them now.

Personally, I would suggest a name like "restricted-isogeny attack". (Not to be confused with "restricted isogeny attack".)

between two public CSIDH curves allow us to compute a corresponding CSIDH private key?[5] Indeed, there are some intricate details in the process of breaking CSIDH when only given *an* isogeny between the starting curve E and the public-key curve $[\mathfrak{a}]E$; see for example [2].

However, any isogeny *defined over* $\mathbb{F}_p$ of polynomially smooth degree[6] between two public CSIDH curves suffices to compute the action of the same private key on another curve: Just like for "real" CSIDH isogenies, every prime-degree step must be the result of acting by an ideal lying above the degree in the endomorphism ring $\mathcal{O}$, and we can thus determine the "direction" of each prime-degree step and recover a CSIDH-style exponent vector (with a potentially extended set of degrees ℓ_i).

2 Meet-in-the-middle

After all this build-up, the somewhat disappointing news is that the best algorithms we have to solve pure isogeny-finding problems (i.e., Problems 1 and 2) in the supersingular isogeny graph are essentially brute-force graph search methods.

The simplest among them is the *meet-in-the-middle* algorithm: It consists of growing two trees in some isogeny graph starting from the two target elliptic curves in a breadth-first manner until the trees, well... meet in the middle. At that point, the connecting path yields an isogeny.[7] The complexity of this algorithm depends a bit on the specifics, but in a graph with good mixing properties (such as any supersingular ℓ-isogeny graph over $\mathbb{F}_{p^2}$, or a CSIDH graph with sufficiently many degrees ℓ_i), we can heuristically apply the birthday "paradox": Modelling the vertices covered by the left and right tree as random, we should see a collision between those sets approximately when both trees have produced about $\sqrt{\#\text{vertices}}$ different vertices.

A more rigorous version of this argument indeed confirms that applying meet-in-the-middle in any ℓ-isogeny graph (for a small prime ℓ) solves Problem 1 for supersingular elliptic curves over $\mathbb{F}_{p^2}$ with time complexity $O(\sqrt{p})$.[8] For Problem 2, when the known degree is smooth and relatively small,[9] it is obviously a good idea to search for that known, particularly short, isogeny. That is, when attacking a SIKE key with $\ell = 2$, we should definitely be applying meet-in-the-middle in the 2-isogeny graph and not in the 7-isogeny graph. Indeed, ignoring the issue discussed in Section 2.1, this leads to an attack on SIKE with (still fully exponential) time complexity $O(p^{1/4})$.

For CSIDH, we can actually view the meet-in-the-middle algorithm as a group-action version of the good old baby-step giant-step algorithm: We can split the prime degrees used in CSIDH isogeny steps into two sets $\ell_1, ..., \ell_m$ and $\ell_{m+1}, ..., \ell_n$ of (close to) equal cardinality, which allows us to reformulate the equation $E_A = \left[\prod_{i=1}^{n} \mathfrak{l}_i^{e_i}\right] E_0$ as $\left[\prod_{i=m+1}^{n} \mathfrak{l}_i^{-e_i}\right] E_A = \left[\prod_{i=1}^{m} \mathfrak{l}_i^{e_i}\right] E_0$. Now, the left- and right-hand sides both come from a square-root-sized set, so we can make a big table of all values on the left-hand side, then search through all values on the right-hand side, until there's a match in the table. Modulo some minor details, this procedure boils down to a meet-in-the-middle search on the isogeny graph as before.

Another way to view the meet-in-the-middle procedure is as a *claw finding* algorithm: Given two functions $f\colon X \to S$ and $g\colon Y \to S$, a *claw* for f and g is simply a pair $(x, y) \in X \times Y$ such that $f(x) = g(y)$. Here, the sets X and Y contain "descriptions" of isogeny paths from E resp. E', and the functions f and g simply map the description of a path to its codomain (by computing the path).

2.1 Memory!

The meet-in-the-middle attack, when applied to cryptographically relevant parameter sets, has one colossal drawback: It uses an exponential amount of memory, and the time complexity is only valid assuming that every memory access is fast, independent of the address. In reality, this is false: Not only

[5] Note that an attacker is satisfied with an *equivalent key*, which not only includes alternative CSIDH exponent vectors, but could in principle be any efficient algorithm that allows them to compute the forward direction of the secret-key operation. In other words, attackers need an algorithm for $E \mapsto [\mathfrak{a}]E$, but they don't need any mathematically recognizable description of $[\mathfrak{a}]$.

[6] As discussed in Section 1.1, any other kind of isogeny is quite difficult to even write down.

[7] Recall that computing the dual of a given isogeny is practically always computationally easy.

[8] We always give the time complexity in base-field operations. When counting bit operations, the O turns into a $\widetilde{O}$, which means the term $\sqrt{p}$ in the O gets multiplied by a polynomial in $\log(p)$: Generally, we have $\widetilde{O}(\mathit{thing}) = \mathit{thing} \cdot (\log(\mathit{thing}))^{O(1)}$.

[9] Warning: The expected minimal degree of an isogeny between two random supersingular elliptic curves over $\mathbb{F}_{p^2}$ is on the order of $\sqrt{p}$. At the same time, the expected minimal degree of an ℓ-power isogeny (for any fixed ℓ) is on the order of p.

is commercially available hardware limited to plugging at most a couple terabytes of RAM into a single computer, but there are actually good reasons to believe that uniformly fast memory accesses are impossible for really large computations. Indeed, at some point, we simply cannot pack enough memory cells into the chunk of physical space close to the CPU, hence memory accesses are literally limited by the speed of light. For quite a while now, the biggest computers in the world have been distributed systems with non-uniform memory architectures: They contain many processing cores, each having direct access to some "local" memory, but accesses to non-local memory require transmitting the data from another processing core over a network. A much more down-to-earth example is the "swapping" of memory contents to disk used by standard computers when under memory pressure: Effectively, the hard drive is used as a secondary unit of main memory, with drastically worse latency than the volatile storage directly attached to (or even contained within) the CPU.

The bottom line is that computations using a lot of memory tend to either be entirely unrealizable (because there is not enough memory), or run dramatically slower than expected as they're spending most of the time waiting for the result of a memory access rather than doing useful computations.[10]

3 Collision finding

The standard way to circumvent the huge memory footprint of meet-in-the-middle type algorithms (in vask generality) is to reduce the search problem to a collision-search problem. For concreteness, we will discuss this in the context of solving Problem 2 to attack SIKE, but the method generalizes easily to all situations where a meet-in-the-middle graph search would work. Recall that we are given two ℓ^a-isogenous supersingular elliptic curves E_0, E_1 defined over $\mathbb{F}_{p^2}$, where $\ell^a \approx \sqrt{p}$.

Collision search is the following problem:

Problem 4. *Let S be a finite set. Given efficient algorithms to compute a function $f\colon S \to S$ and to sample from S uniformly at random, find a collision for f, that is, compute $x, y \in S$ such that $x \neq y$ but $f(x) = f(y)$.*

The standard algorithm to solve this problem is due to van Oorschot and Wiener [10]. In a nutshell, their technique increases the theoretical time complexity from the $O(\sqrt{\#S})$ of straightforward birthday collision search, but it comes with two major benefits in return: The algorithm parallelizes perfectly (i.e., the speedup from using m processors is close to a factor of m), and it works well even when the amount of available memory is quite limited.

Further reading for this section: [10, 1, 3].

3.1 Isogenies from collisions

To reduce isogeny finding to collision search, we have to construct a suitable set S and function f. For this to work, S should somehow describe isogeny paths from either E_0 or E_1, and f should return something related to the codomain of those paths, such that a collision has a chance to be coming from colliding isogeny codomains — which yields a solution of the isogeny problem as before.

The map $f\colon S \to S$ will be composed of two parts: First, a map $g\colon S \to J$ which uses its input as a set of "directions" to perform an isogeny walk and returns the codomain j-invariant, and then a (not necessarily cryptographic) hash function h mapping J back to the set S.

SIDH describes isogeny walks by kernels, and we will do the same: Thus, let

$$S_{i,k} := \left\{\text{cyclic subgroups } H \leq E_i[\ell^k] \text{ of order } \ell^k\right\}.$$

Note that in the actual computations, we do not really store the entire (exponentially big) subgroup. Instead, much like what's done in SIDH, we fix a basis (P, Q) of $E_i[\ell^k]$ and use vectors $(u, v) \in (\mathbb{Z}/\ell^k)^2$ to represent the subgroup $\langle [u]P + [v]Q\rangle$ of $E_i[\ell^k]$.[11]

[10] That said, just like classical mechanics works extremely well in some restricted parameter regime, the RAM model works very well to describe the workings of "small" computers (although the complicated cache hierarchies used in modern processors have already eroded the truth of this statement somewhat).

[11] Restricting to $(u, v) \in \left(\{1\} \times \mathbb{Z}/\ell^k\right) \cup \left(\ell\mathbb{Z}/\ell^k \times \{1\}\right)$ turns the map $(u, v) \mapsto \langle [u]P + [v]Q\rangle \in S_{i,k}$ into a bijection.

Now, to define S and g, we simply pick an integer $k \approx a/2$ and let

$$\begin{aligned} S &:= \big(\{0\} \times S_{0,k}\big) \dot\cup \big(\{1\} \times S_{1,a-k}\big) \\ g\colon &\quad (i, H) \longmapsto j(E_i/H)\,. \end{aligned}$$

Notice that finding a collision of the form $g(0, H) = g(1, H')$ immediately solves the isogeny problem.

However, we still need to define a function h that maps the output of g back to S. For this, we can simply apply any sufficiently random-looking hash function to a given j-invariant and derive integers $i, (u, v)$ from the output, which as described above defines an element of S. While composing with h is required to make the collision-finding subroutine work, the sad news is that we've introduced spurious collisions by doing so: Since h maps a set of size $\approx p$ to a set of size $\approx p^{1/4}$, most of the collisions of the composition $f = h \circ g$ now actually come from h rather than from g, and the latter are the useful ones. Nevertheless, producing *lots* of collisions should at some point lead to a so-called *golden collision*: one that solves the isogeny problem we were originally facing.[12]

3.2 The van Oorschot–Wiener algorithm

Now that we've reduced isogeny finding to collision search, all that remains to do is to run the actual collision-finding algorithm. I will not discuss the full details of this algorithm here for at least two reasons: First, none of it is specific to the isogeny application. Second, the original paper [10] already explains everything very well (including the application to meet-in-the-middle problems and golden collision finding), and it is probably useless for me to repeat everything once more.

Recall that vOW is primarily a *parallel* algorithm. Thus, the kind of architecture we are assuming is a machine with lots of independent processor cores capable of computing f, and a large block of shared memory "in the middle" that's accessible by all cores. (In a real cluster, the shared memory is probably emulated by a network protocol rather than physically connected to each core, but this only changes the optimal parameterization of the algorithm rather than the fundamental concept.)

- The core object underlying the vOW algorithm is that of a *distinguished point*. These are elements of S which were declared to be distinguished, but they are not inherently special in any way. The usual example is that distinguished points are simply those elements whose binary representation ends in a certain number of zero bits.
- Now, the main operation of the algorithm is generating *trails*: Each processor repeatedly produces sequences of elements of S by sampling an element of S at random and repeatedly applying f until a distinguished point is found. It then stores the distinguished point along with the initial element of the trail in memory.
- If the memory already contained another trail ending in the same distinguished point, the processor takes both initial elements and repeats the forward calculation of both trails. As the ends of the trails are the same, the trails must merge at some point, which yields a collision unless the merge happens already at the start of the shorter trail (i.e., the shorter trail is a subsequence of the longer trail).
- Whenever a collision of f is found, the golden-collision check is performed, which in our setting simply means running the input through g instead of $f = h \circ g$. Usually, the collision will have happened in h, so the collision is not golden and the algorithm continues. Occasionally though, we should get lucky and collide already in g, which means we're done.

Warning. The trails leading to distinguished points that are produced in the van Oorschot–Wiener algorithm are absolutely not the same thing as paths in the isogeny graph.

It is not very hard to imagine that the most important configuration option of the algorithm is the number of distinguished points in S: If too few points are distinguished, trails will be extremely long and we barely get any distinguished points, and if too many points are distinguished we run into trouble

[12] Finding lots of collisions diverts slightly from the statement of Problem 4, but the vOW algorithm does not judge nor care and simply keeps spitting out random collisions for quite literally ever. (Though see [7, §1].)

trying to store enough of them to find any collisions. This is usually phrased in terms of the density θ, so that with $\theta = 1$ every point is distinguished and with $\theta = 0$ none are.

In the original paper, van Oorschot and Wiener determine experimentally that a good value of θ is $\theta \approx 2.25\sqrt{w/n}$, where $n = |S|$ and w is the number of distinguished points that fit into memory. The resulting number of evaluations of f before a golden collision is found is then $\approx 2.5\sqrt{n^3/w}$, and as mentioned before, this computation can be parallelized perfectly.

In our isogeny setting, we have $n \approx \ell^{a/2} \approx (\sqrt{p})^{1/2} = p^{1/4}$. Thus, for fixed amounts of memory and processors, the asymptotic complexity of vOW to attack a SIKE-style isogeny problem is $O(p^{3/8})$. This is clearly higher than the $O(p^{1/4})$ of meet-in-the-middle, but this comparison is nonsensical since the latter does not even work asymptotically without hoarding exponentially growing amounts of memory. Thus, vOW is generally considered to be the best attack against SIKE on cryptographically-sized parameters, and rightfully so.

There is also a potential problem we haven't discussed: The collisions obtained from the algorithm are not uniformly random, and it's possible for the function h to conspire against us and make the golden collision appear only with very low probability (or never). Thus, in a golden-collision setting such as when searching isogenies using vOW, we should periodically (in fact, quite frequently) throw away everything we've done and pick a new hash function h. One way of doing this is to pick a random value each time that then gets hashed along with the j-invariant.

4 Deuring's correspondence

In the supersingular case over $\mathbb{F}_{p^2}$, all three Problems 1–3 reduce to computing endomorphism rings, which in graph terms reduces to finding a couple of (independent) cycles. [11, 9, 6]

References

[1] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes and Francisco Rodríguez-Henríquez. "On the Cost of Computing Isogenies Between Supersingular Elliptic Curves". In: *SAC*. Vol. 11349. Lecture Notes in Computer Science. Springer, 2018, pp. 322–343. URL: https://ia.cr/2018/313.

[2] Wouter Castryck, Lorenz Panny and Frederik Vercauteren. "Rational Isogenies from Irrational Endomorphisms". In: *EUROCRYPT (2)*. Vol. 12106. Lecture Notes in Computer Science. Springer, 2020, pp. 523–548. URL: https://ia.cr/2019/1202.

[3] Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes and Fernando Virdia. "Improved Classical Cryptanalysis of SIKE in Practice". In: *Public Key Cryptography (2)*. Vol. 12111. Lecture Notes in Computer Science. Springer, 2020, pp. 505–534. URL: https://ia.cr/2019/298.

[4] Luca De Feo, Simon Masson, Christophe Petit and Antonio Sanso. "Verifiable Delay Functions from Supersingular Isogenies and Pairings". In: *ASIACRYPT (1)*. Vol. 11921. Lecture Notes in Computer Science. Springer, 2019, pp. 248–277. URL: https://ia.cr/2019/166.

[5] Steven D. Galbraith, Christophe Petit, Barak Shani and Yan Bo Ti. "On the Security of Supersingular Isogeny Cryptosystems". In: *ASIACRYPT (1)*. Vol. 10031. Lecture Notes in Computer Science. Springer, 2016, pp. 63–91. URL: https://ia.cr/2016/859.

[6] David Kohel, Kristin Lauter, Christophe Petit and Jean-Pierre Tignol. "On the quaternion ℓ-isogeny path problem". In: *LMS Journal of Computation and Mathematics* 17 (2014), pp. 418–432. URL: https://ia.cr/2014/505.

[7] Lorenz Panny. *Stellingen accompanying the thesis Cryptography on Isogeny Graphs by Lorenz Panny*. 2021. URL: https://yx7.cc/docs/phd/stellingen.pdf.

[8] The Sage Developers. *SageMath, the Sage Mathematics Software System*.

[9] Boris Fouotsa Tako, Péter Kutas and Simon-Philipp Merz. *On the Isogeny Problem with Torsion Point Information*. IACR Cryptology ePrint Archive 2021/153. 2021. URL: https://ia.cr/2021/153.

[10] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Cryptanalytic Applications". In: *Journal of Cryptology* 12.1 (1999), pp. 1–28.

[11] Benjamin Wesolowski. *The supersingular isogeny path and endomorphism ring problems are equivalent*. IACR Cryptology ePrint Archive 2021/919. 2021. URL: https://ia.cr/2021/919.

5 Exercises

These are intended to be solved with the assistance of a computer-algebra system such as SageMath [8].

5.1 Harder or easier or both?

5.1.1

Show that Problem 1 for supersingular curves over $\mathbb{F}_{p^2}$ is polynomial-time reducible to Problem 2.
Hint: Each ℓ-isogeny graph in this setting is an expander, i.e., there are paths of logarithmic length between all pairs of nodes.

5.1.2

Find an example where Problem 1 is exponentially easier than Problem 2.

5.1.3

Is Problem 3 hard when N equals m?

5.2 Meet me halfway, $\mathbb{F}_{p^2}$ edition

Let $p = 4084079$ and let $\mathrm{i}^2 = -1 \in \mathbb{F}_{p^2}$. Define the curves

$$E_0 \colon y^2 = x^3 + x$$
$$E_1 \colon y^2 = x^3 + (995970 + 2085238\mathrm{i})x + (287520 + 1073219\mathrm{i})$$

over $\mathbb{F}_{p^2}$. Compute an isogeny $E_0 \to E_1$.

5.3 Meet me halfway, $\mathbb{F}_p$ edition

Let $p = 4084079$. Define the curves

$$E_0 \colon y^2 = x^3 + x$$
$$E_1 \colon y^2 = x^3 + 2060675x + 1790700$$

over $\mathbb{F}_p$. Compute an isogeny $E_0 \to E_1$ defined over $\mathbb{F}_p$.

5.4

In an attempt, we may speculate, to advertise SIKE to the world as a very secure and conservative cryptosystem, a particular well-known US technology company has recently announced non-negligible cash prizes for breaking toy instances of SIKE. They are appropriately named \$IKEp182 and \$IKEp217. See `https://github.com/microsoft/SIKE-challenges` for the challenge instances.

Implement the function g from Section 3.1 for the smaller instantiation (\$IKEp182). Estimate how many core hours it would take to break the challenge using your implementation on a reasonably-sized machine, and decide for yourself if it's (a) feasible at all, and (b) profitable. Good luck!

Money can't buy me SIKE private keys. — The Beatles, probably