Breaking the $IKEp182 Challenge*

Aleksei Udovenko[1] and Giuseppe Vitto[2]

[1] CryptoExperts, Paris, France
aleksei@affine.group
[2] SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg
giuseppe.vitto@uni.lu

October 21, 2021

Abstract. We report a break of the $IKEp182 challenge using a meet-in-the-middle attack strategy improved with multiple SIKE-specific optimizations. The attack was executed on the HPC cluster of the University of Luxembourg and required less than 10 core-years and 256TiB of high-performance network storage (GPFS). Different trade-offs allow execution of the attack with similar time complexity and reduced storage requirements of only about 70TiB.

Keywords: Isogenies · Cryptanalysis · SIDH · SIKE

1 Introduction

Under the threat of quantum computers appearing in the near future, public-key cryptography has to evolve to keep modern communication protocols secure. To foster the evolution, NIST organizes a competition for Post-Quantum Cryptography Standardization (PQC) [Nat22]. SIKE [JAC+20] (Supersingular Isogeny Key Encapsulation) is one of the alternate candidates of the ongoing 3rd round. It is based on the SIDH protocol (Supersingular Isogeny Diffie-Hellman) developed by De Feo and Jao [JD11], following and improving the ideas of the constructions proposed by Rostovtsev and Stolbunov [RS06,Sto10]. *Isogeny*-based cryptography only recently started to develop rapidly.

In particular, for a specially shaped prime p, the security of SIKE relies on the hardness of finding an isogeny between two given supersingular elliptic curves defined over the finite field $\mathbb{F}_{p^2}$. The classic meet-in-the-middle attack (MitM, also known as bidirectional search), applied in the isogeny setting by Galbraith [Gal99], requires $\mathcal{O}(p^{1/4})$ time and memory/storage. Adj, Cervantes-Vazquez, Chi-Domínguez, Menezes and Rodríguez-Henríquez [ACC+19] observed that large amounts of storage are likely impossible to be achieved in practice due to fundamental physical constraints. They thus applied the classic low-memory van Oorschot-Wiener (vOW) golden collision search [vW99] to the isogeny setting by using less memory at the expense of more time, and conjectured that this attack represents the main threat to SIKE. Improved analysis of the application of van Oorschot-Wiener to SIKE with further optimizations was given by Costello, Longa, Naehrig, Renes and Virdia [CLN+20]. Based on this analyses, Longa, Wang and Szefer [LWS21] estimated the real costs of mounting such attack at various security levels, concluded that previous security estimates were conservative, and proposed to revise parameters in order to improve efficiency. For example, they propose to replace SIKEp434 with SIKEp377, which is 40% faster, while still targeting to satisfy NIST Level 1 security requirements.

*The work of Giuseppe Vitto was supported by the Luxembourg National Research Fund (FNR) project FinCrypt (C17/IS/11684537). The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg [VBCG14] – see https://hpc.uni.lu.

In order to motivate security analysis of SIKE, Microsoft recently published two challenges [Mic21b] with reduced-size instances of SIKE: $IKEp182 and $IKEp217, with bounties of $5000 and $50 000, respectively. In this report, we describe how we managed to break the first instance using the HPC facilities of the University of Luxembourg [VBCG14]. While the classic security of the instance via the meet-in-the-middle attack is only about 45 bits, such amount of memory (2^{45} storage units $\geq$ 256TiB) is not trivial to manage efficiently. Nonetheless, we chose to stick to MitM instead of vOW due to the large overheads introduced by the latter, where, for example, a single step requires computing expensive isogenies (which can instead be amortized in MitM), and large penalties are paid to reduce the memory usage. Our implementation is mainly written in SageMath [The21] and C++, using parts of the SIDH library by Microsoft Research [Mic21a].

1.1 Our Approach

At the high level, we used the classic meet-in-the-middle approach for solving the isogeny path problem, in which the hardness of SIKE lies. We developed/applied several optimizations:

(2-bit leak from the knowledge of the final curve)
: In [CLN+20], it was noted that the final curve (i.e., the image of the initial curve through a secret 2^e-isogeny) fully leaks the last 4-isogeny. This effectively reduces the set of j-invariants that can be reached from the final curve by a factor of 4. In addition, we show how to express this reduced set in the same form as the set of j-invariants reached from the initial curve. This simplifies conceptually the MitM application to SIKE, by unifying the representation of sets arising from the initial and the final curves. In the case of $IKEp182, both sets have 2^{44} elements, after applying this and the next optimization.

(1-bit conjugation-based reduction)
: In SIKE, the initial Montgomery curve is $y^2 = x^3 + 6x^2 + x$, and by being defined over $\mathbb{F}_p$, all the curves 2^e-isogenous over $\mathbb{F}_{p^2}$ to it (through SIKE isogenies), have j-invariants which can be grouped in conjugate pairs. It is thus sufficient to search for a collision of e.g. the real part of the j-invariants in the middle, to effectively halve the size of the set arising from the initial curve. Recovering the full colliding j-invariant from such partial collision is easy due to the fact that paths to conjugate elements are element-wise conjugates. This technique was discovered and applied in the vOW setting in [CLN+20].

(Efficient tree exploration and optimal strategy)
: A direct application of meet-in-the-middle with the (optimized) arithmetic from SIKE, would recompute a lot of intermediate steps repeatedly (simply speaking, computing each entry in the middle would require following a full path from the root of a full binary tree to its leaf). However, these computations are not fully identical and can not be avoided by simply storing some intermediate values. We show how to explore the tree more efficiently, and adapt the optimal isogeny evaluation strategy of [DJP11] to this case.

(Disk-based storage and sorting)
: It is much more feasible to obtain and use a large amount of disk-based storage, than a similar amount of RAM memory. However, the classic meet-in-the-middle formulation uses a (hash-)table where the majority of queries follow a random access pattern, most suitable for RAM. When disk storage is used, latency represents the bottleneck of using hash-tables, and limits the application of parallelization. To counter this, we follow an alternative approach to implement the MitM attack: we

generate the two large j-invariants sets arising from the starting and the final curves, and we intersect them using *sorting* and *merging* techniques, which, instead, mostly require a sequential access pattern.

(Storage-collision trade-off and compression)
Truncating intermediate entries (j-invariants representations) permits to reduce storage requirements at the cost of allowing false-positive collisions. By omitting all the auxiliary information (e.g. the path in the set to an entry), we can reduce the storage further at the cost of an extra recomputation step, where the two sets are recomputed (fully memoryless and in parallel) in order to retrieve the relevant auxiliary information for collisions found in the previous step. Furthermore, the resulting sets become *dense* due to the truncation of entries, and can be compressed (when sorted) by storing the differences between successive elements. In our case, we used 64-bit entries, which already at 32GiB of sorted data (2^{32} truncated entries) have the expected difference of about 32 bits. This reduces the total storage requirements down to approximately $2^{44} \times 2 \times 4$ bytes = 128 TiB.

2 Preliminaries

2.1 The Supersingular Isogeny Graph

Definition 1. An isogeny of elliptic curves $\phi : E \to E'$ defined over $\mathbb{F}_q$ is a surjective morphism of curves that induces a group homomorphism $E(\overline{\mathbb{F}}_q) \to E'(\overline{\mathbb{F}}_q)$. When such map exists, E and E' are said to be isogenous over $\mathbb{F}_q$.

An isogeny of elliptic curves $\phi : E \to E'$ defined over $\mathbb{F}_q$ can be represented as a non-constant rational map fixing the identity, i.e.,

$$\phi : \quad (x, y) \mapsto \left(\frac{a(x)}{c(x)}, \frac{b(x)}{d(x)} y \right)$$

with $a(x), b(x), c(x), d(x) \in \mathbb{F}_q[x]$ and $\gcd(a(x), c(x)) = \gcd(b(x), d(x)) = 1$. The *degree* of ϕ is defined as $\max(\deg a(x), \deg c(x))$ and ϕ is said to be *separable* if $\left(\frac{a(x)}{c(x)}\right)' \neq 0$. For every separable degree-d isogeny $\phi : E \to E'$, there exists a dual degree-d isogeny $\hat{\phi} : E' \to E$ so that the maps $\phi \circ \hat{\phi} = [d]_E$ and $\hat{\phi} \circ \phi = [d]_{E'}$ are the multiplication-by-d endomorphisms on E and E', respectively.

If d is composite, it is possible to decompose a degree-d isogeny, or simply a d-isogeny, into a composition of isogenies of prime order. We note that this property allows, in practice, to compute efficiently high (smooth) degree isogenies. More precisely, if $d = p_0^{e_0} \cdot \ldots \cdot p_n^{e_n}$ and ϕ is a d-isogeny, then there exists p_i-isogenies $\phi_j^{p_i}$, with $i \in [0, n]$ so that

$$\phi = \phi_1^{p_0} \circ \ldots \circ \phi_{e_0}^{p_0} \circ \ldots \circ \phi_1^{p_n} \circ \ldots \circ \phi_{e_n}^{p_n}$$

It is well known that separable isogenies $\phi : E(\mathbb{F}_q) \to E'(\overline{\mathbb{F}}_p)$ (up to isomorphism) are in bijections with subgroups G of $E(\overline{\mathbb{F}}_p)$ so that $\ker(\phi) = G$ and ϕ is a $|G|$-isogeny: in such case, the curve E' is isomorphic to the group quotient E/G.

In the following, we will consider only separable isogenies over Montgomery elliptic curves.

Definition 2 (Montgomery Elliptic Curves)**.** An elliptic curve over a finite field $\mathbb{F}_q$ is said Montgomery of parameters $A, B \in \mathbb{F}_q$ if it has equation $E_{A,B} : By^2 = x^3 + Ax^2 + x$ with $B(A^2 - 4) \neq 0$.

The j-invariant of a Montgomery elliptic curve $E_{A,B}(\mathbb{F}_p)$ is equal to $j(E_{A,B}) = \frac{256(A^2-3)^3}{A^2-4}$. Hence, the $\overline{\mathbb{F}}_p$-isomorphism class[1] of $E_{A,B}(\mathbb{F}_p)$ depends only on A^2.

[1] Two elliptic curves are $\overline{\mathbb{F}}_p$-isomorphic if they have the same j-invariant.

Through a simple change of variables, Montgomery curves $E_{A,B}$ are isomorphic over $\mathbb{F}_q$ to only one of the following two quadratic twists

$$E_{A,B} \simeq \begin{cases} y^2 = x^3 + Ax^2 + x & \text{if } B \in \mathbb{F}_q \text{ is a square} \\ uy^2 = x^3 + Ax^2 + x & \text{otherwise} \end{cases}$$

where u is a non-square of $\mathbb{F}_q$.

Supersingular elliptic curves (i.e. curves with trace congruent to 0 mod p), have their j-invariant defined over $\mathbb{F}_{p^2}$ [Sil09, V.3 - Theorem 3.1.a]: in fact, any supersingular curve is isomorphic to an elliptic curve defined over $\mathbb{F}_{p^2}$ and we can thus consider only supersingular curves over $\mathbb{F}_{p^2}$. Moreover, the property of being supersingular is invariant under isogeny, and is induced by curves' j-invariants: if there is a supersingular curve with j-invariant equal to j, then j is said to be a *supersingular j-invariant* and all curves having j as j-invariant are supersingular too.

Definition 3 (Supersingular Isogeny Graph)**.** For p, ℓ distinct primes, the degree-ℓ supersingular isogeny graph over $\mathbb{F}_{p^2}$ is the graph where vertexes are curves' representatives of $\mathbb{F}_{p^2}$-isomorphism classes, and two vertexes are connected by an (undirected) edge if and only if there exists a separable ℓ-isogeny between them.

By Hasse's bound, supersingular curves E over $\mathbb{F}_{p^2}$ have $\#E(\mathbb{F}_{p^2}) = p^2 + 1 - t$ number of points, where the trace t can be equal only to $0, \pm p, \pm 2p$. Since, by Tate's Isogeny theorem [Tat66], two curves are isogenous over $\mathbb{F}_q$ if and only if have the same number of points over $\mathbb{F}_q$, it follows that separable ℓ-isogenies over $\mathbb{F}_{p^2}$ partition the ℓ-degree Supersingular Isogeny graph over $\mathbb{F}_{p^2}$ into multiple connected subgraphs, each connecting curves' representatives of same trace.

In [AAM19, Theorem 6] is proved that the two subgraphs associated to traces $2p$ and $-2p$ are isomorphic, which in turn are isomorphic to the ℓ-degree supersingular isogeny graph built considering $\overline{\mathbb{F}}_p$-isomorphism classes instead. This fact suggests that we can equivalently (in terms of security) work in any of these two $O(p)$ size subgraphs induced by supersingular curves with traces in $\pm 2p$ (equivalently, by curves of cardinality $(p \pm 1)^2$), moving between neighbour representatives using ℓ-isogenies. Interestingly, from the fact that for a curve $E(\mathbb{F}_{p^2})$ and an $\ell \nmid p$, we have $E[\ell] \simeq \mathbb{Z}_\ell \times \mathbb{Z}_\ell$, it follows that supersingular curves E belonging to classes in these two subgraphs, decomposes as, depending on E cardinality, $E \simeq \mathbb{Z}_{p\pm1} \times \mathbb{Z}_{p\pm1}$. In particular, curves coincide with their $(p \pm 1)$-torsions, which imply that the latter are $\mathbb{F}_{p^2}$-rational.

By adopting Montgomery curves, it is possible to further simplify this setting. If, for a supersingular j-invariant j_0, we have $j_0 = j(E_{A,1})$, then clearly its quadratic twist satisfies $j_0 = j(E_{A,u})$. However, if we exclusively use the efficient Montgomery x-coordinate only arithmetic (which employs the curve A-coefficient only), the twist selected becomes irrelevant[2], since it affects only the y-coordinate, and we can then represent Montgomery curves $E_{A,B}$ simply as E_A.

It follows that, in practice, by using Montgomery curves and x-coordinate arithmetic only, the two isomorphic supersingular isogeny subgraphs corresponding to the traces $\pm 2p$, coincide, and vertexes can be denoted with just supersingular j-invariants rather than with isomorphism class curve representative.

Definition 4 (Supersingular Isogeny Graph - Revisited)**.** For p, ℓ distinct primes, the degree-ℓ supersingular isogeny graph over $\mathbb{F}_{p^2}$ is the graph where vertexes are supersingular j-invariants, and two vertexes (j_1, j_2) are connected by an (undirected) edge if and only if there exists a separable ℓ-isogeny between two Montgomery curves E_{A_1} and E_{A_2} so that $j_1 = j(E_{A_1})$ and $j_2 = j(E_{A_2})$.

[2] In [Cos20] it was noted that, in fact, it is possible to build isogeny-based schemes over Montgomery curves which are *"twist-agnostic"*, that is can work independently of curves' quadratic twist chosen.

Definition 5 ($\mathcal{J}_A^d$-set)**.** Let E_A be a supersingular Montgomery elliptic curve over $\mathbb{F}_{p^2}$ and let d be so that $\#E_A(\mathbb{F}_{p^2}) = d \cdot r$ with $\gcd(d, r) = 1$. The $\mathcal{J}_A^d$-set is then

$$\mathcal{J}_A^d = \left\{ j(E_{A'}) \in \mathbb{F}_{p^2} \mid \exists \text{ a separable } d\text{-isogeny } \phi : E_A \to E_{A'} \right\}$$

i.e., the set of j-invariants of curves d-isogenous to E_A.

We note that due to the existence of dual isogenies, edges in an isogeny graph are undirected. In the case of supersingular curves, the degree-ℓ isogeny graph over $\mathbb{F}_{p^2}$ has approximately $\lfloor \frac{p+1}{12} \rfloor$ vertexes [Sch87, Theorem 4.6] and each vertex has exactly $\ell + 1$ neighbours (counting multiplicities), with edges corresponding to an isogeny with kernel being a distinct order-ℓ subgroup of the torsion $\mathbb{Z}_\ell \times \mathbb{Z}_\ell$. In other words, the degree-ℓ supersingular isogeny graph is a connected $(\ell + 1)$-regular graph which results to be Ramanujan (see [Piz90, Piz98]). It follows that random length-e walks in the supersingular isogeny graphs correspond to ℓ^e-isogenies between supersingular elliptic curves.

By exploiting the correspondence between order-ℓ kernels and ℓ-isogenies, a walk in the supersingular isogeny graph starting from a curve E, can be expressed in terms of a linear combination of two independent generators of the torsion $E[\ell]$.

Definition 6 (Walk)**.** Let E_0 be a supersingular elliptic curve over $\mathbb{F}_{p^2}$, ℓ a prime distinct from p and let (P_0, Q_0) be two independent generators of $E_0[\ell^e] = \mathbb{Z}_{\ell^e} \times \mathbb{Z}_{\ell^e}$. Two values $a, b \in \mathbb{Z}_{\ell^e}$ not simultaneously divisible by ℓ, define a separable ℓ^e-isogeny $\phi = \phi_{e-1} \circ \ldots \circ \phi_0 : E_0 \to E_e$ over $\mathbb{F}_{p^2}$ (i.e., a walk in the supersingular isogeny graph), where, for $i \in [0, e-1]$, $\phi_i : E_i \to E_{i+1}$ is an ℓ-isogeny with $\ker(\phi_i) = \langle [\ell^{e-1-i}] \cdot ([a]P_i + [b]Q_i) \rangle$ and $(P_{i+1}, Q_{i+1}) = (\, \phi_i(P_i), \phi_i(Q_i)\,)$. We will often refer to such ϕ as the isogeny arising from $[a]P + [b]Q$.

Remark 1. If $\ell \nmid a$, then $\langle [a]P + [b]Q \rangle = \langle P + [s]Q \rangle$, with $s = a^{-1}b \in \mathbb{Z}_{\ell^e}$, and such subgroups give rise to ℓ^e distinct isogenies. If instead $a = \ell \cdot c$, kernels can be written as $\langle [s\ell]P + Q \rangle$, with $s = b^{-1}c \in \mathbb{Z}_{\ell^e}$ and there exists at most ℓ^{e-1} such distinct subgroups. This brings the total number of walks that can be traversed from a starting curve E_0 to $\ell^{e-1}(\ell+1)$, which in turn correspond to all walks obtained by iteratively exploring all $\ell+1$ neighbours of E_0 up to depth e (with no backtracking). Kernels of the form $\langle P + [s]Q \rangle$, with $s \in \mathbb{Z}_{\ell^e}$, will be the ones employed by SIKE (Subsection 2.2): we note that this choice restricts the possible isogeny-paths that can be walked, since only ℓ out of $\ell+1$ neighbours of E_0 can be explored.

The main observation that ensures correctness of Definition 6 is that the order of $\phi_i([a]P_i + [b]Q_i)$ decreases by ℓ with respect to the order of $[a]P_i + [b]Q_i$: indeed, from $\ker(\phi_i) = \langle [\ell^{e-1-i}] \cdot ([a]P_i + [b]Q_i) \rangle$ we must have $[\ell^{e-1-i}] \cdot \phi_i([a]P_i + [b]Q_i) = \mathcal{O}_{E_{i+1}}$ and since P_0, Q_0 both have order ℓ^e, by induction, we can conclude that $\phi_i([a]P_i + [b]Q_i)$ has order ℓ^{e-1-i}.

The difficulty to obtain the scalar s from two curves E and E' isogenous through the ℓ^e-isogeny arising from $P + [s]Q$, is one of the different (formulations of) problems which are believed to be hard in the supersingular isogeny setting.

Problem 1 (Path-finding)**.** *Given two supersingular Montgomery curves E_A and $E_{A'}$ over $\mathbb{F}_{p^2}$ so that, for an $\ell \nmid p$ prime and $e > 0$, there exists a separable ℓ^e-isogeny $\phi : E_A \to E_{A'}$ over $\mathbb{F}_{p^2}$ (equivalently $j(E_{A'}) \in \mathcal{J}_A^{\ell^e}$), find a sequence of groups $\{K_i\}_{i \in [1,e]}$ such that*

- *ϕ_i is a separable ℓ-isogeny defined over $\mathbb{F}_{p^2}$ with $\ker(\phi_i) = K_i$;*
- *$\phi = \phi_1 \circ \ldots \circ \phi_e$ up to isomorphism.*

We note that a solution to Problem 1 for an ℓ^e-isogeny arising from $P + [s]Q$ can be efficiently mapped bit-by-bit to the corresponding generating secret value s.

2.2 The SIKE Protocol

Supersingular Isogeny Key Encapsulation (SIKE) [JAC+20] is a post-quantum key encapsulation mechanism (KEM) based on the difficulty to find a length-e path between two ℓ^e-isogenous elliptic curves (Problem 1). It is based on the Supersingular Isogeny Diffie-Hellman (SIDH) [JD11] key exchange.

In SIKE, p has the form $p = 2^{e_A}3^{e_B} - 1$ with $2^{e_A} \approx 3^{e_B}$ and the working field is set to be $\mathbb{F}_{p^2} = \mathbb{F}_p(i) = \mathbb{F}_p[x]/(x^2+1)$. The parameters e_A and e_B are chosen so that the Montgomery curve $E = E_6$ over $\mathbb{F}_{p^2}$ is supersingular with $(p+1)^2$ rational points and torsions $E[\ell_A^{e_A}] = \mathbb{Z}_{\ell_A^{e_A}} \times \mathbb{Z}_{\ell_A^{e_A}} = \langle P_A, Q_A \rangle$ and $E[\ell_B^{e_B}] = \mathbb{Z}_{\ell_B^{e_B}} \times \mathbb{Z}_{\ell_B^{e_B}} = \langle P_B, Q_B \rangle$. To avoid some technicalities introduced by adopting efficient 2-isogeny computation formulas, the order-2 point $(0,0)$ is not allowed to be into any 2-isogeny kernel in a path arising from $P_A + [s]Q_A$ with $s \in \mathbb{Z}_{2^{e_A}}$: thanks to a result of Renes [Ren18, Corollary 2], this is guaranteed by choosing the generators P_A, Q_A of the torsion $E[2^{e_A}]$ so that $[2^{e_A-1}]Q_A = (0,0)$.

Once the public parameters $(p, E(\mathbb{F}_{p^2}), P_A, Q_A, P_B, Q_B)$ are generated, two parties, Alice and Bob, can agree on a common secret as follows:

- Alice picks secret $s_A \leftarrow_\$ \mathbb{Z}_{2^{e_A}}$ and computes the 2^{e_A}-isogeny $\phi_A : E \to E_A$ arising from $\langle P_A + [s_A] \cdot Q_A \rangle$. She then sends to Bob E_A and the points $\phi_A(P_B)$, $\phi_A(Q_B)$.
- Bob picks secret $s_B \leftarrow_\$ \mathbb{Z}_{3^{e_B}}$ and computes the 3^{e_B}-isogeny $\phi_B : E \to E_B$ arising from $\langle P_B + [s_b] \cdot Q_B \rangle$. He then sends to Alice E_B and the points $\phi_B(P_A)$, $\phi_B(Q_A)$.
- Alice computes the 2^{e_A}-isogeny $\phi_{\tilde{A}} : E_B \to E_{BA}$ arising from $\langle \phi_B(P_A) + [s_A] \cdot \phi_B(Q_A) \rangle$ and sets the common secret to $j(E_{BA})$.
- Bob computes the 3^{e_B}-isogeny $\phi_{\tilde{B}} : E_A \to E_{AB}$ arising from $\langle \phi_A(P_B) + [s_B] \cdot \phi_A(Q_B) \rangle$ and sets the common secret to $j(E_{AB})$.

It easy to see that since separable isogenies correspond to curve quotients, in this setting they commute, and so $j(E_{BA}) = j(E_{AB})$. For more details and proof of correctness of the above protocol we refer to [JD11, JAC+20].

2.3 Efficient Isogeny Computation

In this section we provide an overview of how isogenies, and thus walks in the isogeny graph, can be practically and efficiently computed.

We will focus on ℓ-isogenies with $\ell = 2, 3$, relevant for SIKE and for our attacks. Proofs that the following formulas define isogenies can be found, for example, in [CH17, Ren18].

Proposition 1 (2-isogeny)**.** *Let $E_{A,B}$ be a Montgomery supersingular elliptic curve over $\mathbb{F}_{p^2}$ with $p \neq 2$ and let $R = (x_R, y_R) \in E(\mathbb{F}_{p^2})$ be an order 2 point not equal to $(0,0)$. Then*

$$\begin{aligned} \phi : E_{A,B} &\longrightarrow E_{A',B'} \\ (x,y) &\longmapsto (f(x), yf'(x)) \end{aligned}$$

with

$$f(x) = x\frac{x \cdot x_r - 1}{x - x_r}$$

is a separable 2-isogeny between Montgomery elliptic curves with $\ker(\phi) = \langle R \rangle$ and $(A', B') = (2(1 - 2x_r^2), Bx_r)$.

Remark 2. The 2-isogeny defined in Proposition 1 fixes the point $(0,0)$, and thus cannot belong to its kernel.

Proposition 2 (3-isogeny)**.** *Let $E_{A,B}$ be a Montgomery supersingular elliptic curve over $\mathbb{F}_{p^2}$ with $p \neq 2$ and let $R = (x_R, y_R) \in E(\mathbb{F}_{p^2})$ be an order 3 point. Then*

$$\begin{aligned} \phi : E_{A,B} &\longrightarrow E_{A',B'} \\ (x, y) &\longmapsto (f(x), yf'(x)) \end{aligned}$$

with

$$f(x) = x\frac{x \cdot (x_R - 1)^2}{(x - x_R)^2}$$

is a separable 3-isogeny between Montgomery elliptic curves with $\ker(\phi) = R$ *and* $(A', B') = (-6x_R^3 + Ax_R^2 + 6x_R, Bx_R^2)$.

Walk structure induced by SIKE 2-isogenies The structure induced by 2-isogeny formulas adopted by SIKE is of relevance for the attack we will outline in next sections.

It is easy to see that in the $\mathbb{F}_{p^2}$-isomorphism class of a supersingular j-invariant j_0, we have (at most) 6 distinct Montgomery curves: if $\pm A$ satisfy the equation $j_0 = \frac{256(x^2-3)^3}{x^2-4}$, then also

$$\pm B = \frac{3\tilde{x} + A}{\sqrt{\tilde{x}^2 - 1}} \qquad \pm C = \frac{3\tilde{z} + A}{\sqrt{\tilde{z}^2 - 1}}$$

do, where $\tilde{x}, \tilde{z} = 1/\tilde{x}$ are roots of $x^2 + Ax + 1 = 0$.

When these 6 coefficients are all distinct, a 2-isogeny as in Proposition 1 can walk in the supersingular isogeny graph to only 2 of the possible 3 neighbour j-invariants j_1, j_2, j_3, and whose values depend on the A-coefficient of the curve to which we are applying the isogeny.

As already noted in Remark 2, by using SIKE 2-isogenies, we cannot have $\langle(0, 0)\rangle$ as kernel: this practically correspond to the fact that if a curve E_B, with $j(E_B) = j_0$, is pushed through a 2-isogeny to $E_{A'}$, then $E_{A'}$ will not be pushed back to E_B by any of the 2-isogeny induced by an order-2 subgroup of $E_{A'}$ distinct from $\langle(0, 0)\rangle$.

In the $\mathbb{F}_{p^2}$-isomorphism class of $E_{A'}$, however, there will be 4 curves $E_{\pm B'}, E_{\pm C'}$ which can be pushed back to a curve in the isomorphism class of E_B (i.e., j_0), but not to the curve E_B itself, because, otherwise, there will be a 2-isogeny that will move E_B back to $E_{A'}$, a circumstance prevented by not allowing $\langle(0, 0)\rangle$ to be an isogeny kernel.

It follows that each of the 4 curves $E_{\pm B'}, E_{\pm C'}$ can be pushed to only one of the two isomorphic curves $E_{\pm A}$[3], which will eventually be pushed further to nodes j_3, j_2 distinct from $j(E_{A'}) = j(E_{\pm B'}) = j(E_{\pm C'}) = j_1$.

This example is illustrated (with same notation) in Figure 1.

3 The Meet-in-the-Middle Attack for Solving the Isogeny Path Problem

3.1 High-level Description

In this section we will provide an overview of the meet-in-the-middle attack to solve the path-finding Problem 1. In terms of path search on a graph between two nodes, this approach is also known as *bidirectional search*.

In order to find a path of length e between two curves E_A and E_B in the supersingular isogeny graph (i.e., an ℓ^e-isogeny between E_A and E_B), an attacker can explore all length-$\lfloor e/2 \rfloor$ paths starting from E_A and all length-$\lceil e/2 \rceil$ paths starting from E_B (intuitively, this corresponds to exploring the subgraph *spheres* centered in E_A and E_B with radius $\lfloor e/2 \rfloor$ and $\lceil e/2 \rceil$, respectively) looking for a non-trivial intersection: since isogenies are

[3]Since, in SIKE, $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$, the map $(x, y) \mapsto (-x, iy)$ defines an isomorphism between E_* and E_{-*}.

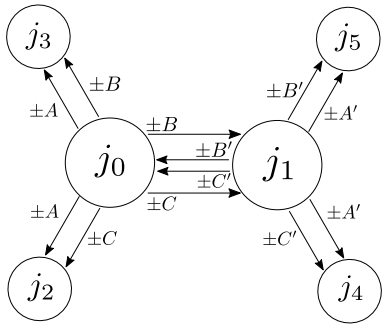


Figure 1: The different $j-$invariants reached by pushing curve's A-coefficients through 2-isogenies defined by Proposition 1. Here $\pm A, \pm B, \pm C$ are the 6 roots satisfying $j_0 = \frac{(x^2-3)^3}{(x^2-4)}$ (resp. $\pm A', \pm B', \pm C'$ and j_1), and define 6 Montgomery curves isomorphic over $\mathbb{F}_{p^2}$. The two edges associated to a certain coefficient represent isogenies with kernels order-2 subgroups not equal to $\langle (0,0) \rangle$.

defined up to isomorphisms, we can identify the curve(s) *in-the-middle* by computing their j-invariants.

The full path can then reconstructed either by iteratively applying the same attack on the two found half-length sub-paths, or by simply storing the paths starting in E_A or E_B associated with j-invariant in the middle and concatenate them once a collision is found.

Problem 2 (Meet-in-the-Middle)**.** *Given two ℓ^e-isogenous curves $E_A(\mathbb{F}_{p^2})$ and $E_B(\mathbb{F}_{p^2})$ for some $\ell \nmid p$ prime and $e > 0$, the Meet-in-the-Middle (MitM) problem asks to find the intersection*

$$\mathcal{J}_A^{\ell^{\lfloor e/2 \rfloor}} \cap \mathcal{J}_B^{\ell^{\lceil e/2 \rceil}}$$

3.2 Application to SIKE

In SIKE, MitM can be applied to attack either Alice's or Bob's public key: indeed, from Alice's public key we can easily recompute the curve E_A that is 2^{e_A}-isogenous to the starting curve E, and, similarly, Bob's public key reveals the curve E_B that is 3^{e_B}-isogenous to the starting curve E. Explicitly finding the secret isogeny $\phi_A : E \to E_A$ or $\phi_B : E \to E_B$, allows the attacker to reapply it to the other party's public key to ultimately obtain the shared secret key.

As already noted in Remark 1, in SIKE not all $(\ell+1)\ell^{e-1}$ isogenies are possible, because isogeny kernels are restricted to the shape $\langle P + [s]Q \rangle$, which excludes in the first ℓ-isogeny step the kernel $\langle [\ell^{e-1}]Q \rangle$, leaving only ℓ^e isogenies.

In Subsection 4.1, we show that the isogeny formulas of Subsection 2.3 can be used to walk from the curve E_A towards the starting curve E, by moving to an isomorphic curve $E_{A'}$ and defining kernels as $P' + [s']Q'$ with $\langle P', Q' \rangle = E_{A'}[\ell^e]$.

This refines the meet-in-the-middle Problem 2 into generating and intersecting the leaves of the two "trees" of j-invariants spanned by *walks* from the bases $(P, Q) \in E(\mathbb{F}_{p^2})$ and $(P', Q') \in E_{A'}(\mathbb{F}_{p^2})$. The meet-in-the-middle trees structure for $\ell = 2$ is illustrated in Figure 2.

Definition 7 (SIKE-tree)**.** Given a curve E defined over $\mathbb{F}_{p^2}$ and a basis (P, Q) for its torsion $E[\ell^e]$, the *tree* spanned by (P, Q) of depth $d \leq e$ is the directed graph consisting of all length-d walks from E arising from $[\ell^{e-d}] \cdot (P + [s]Q)$ with $s \in \mathbb{Z}_{\ell^e}$, i.e. all length-$d$ walks from E excluding those arising from $[\ell \cdot a]P + [b]Q$ for any $a, b \in \mathbb{Z}_{\ell^e}$.

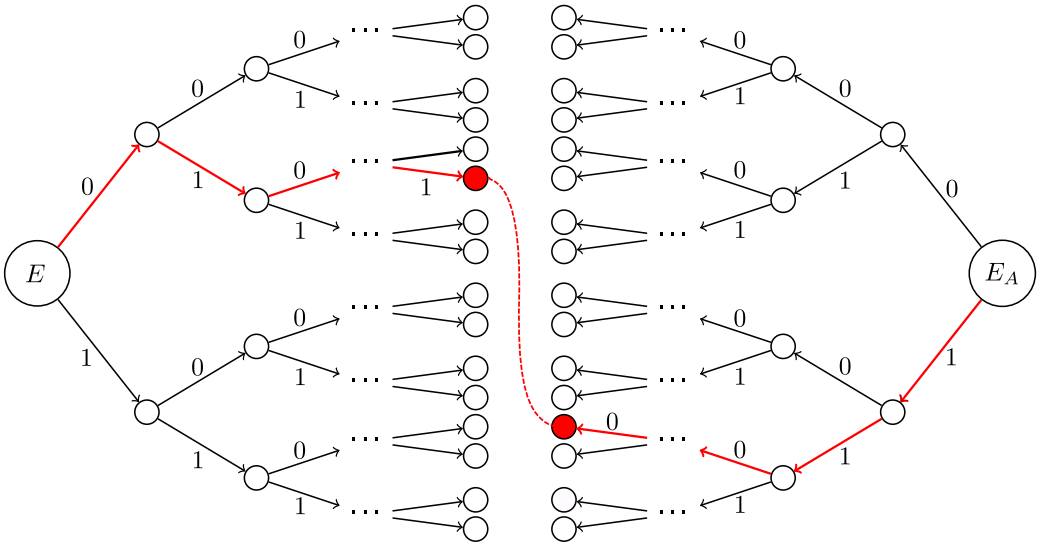


Figure 2: Example of 2-isogeny trees starting from the two 2^e-isogenous curves E and E_A. Red nodes in the middle denote curves with same j-invariant, whose respective path in the tree (in red) connect E_A to E_B. Edge labels are assigned arbitrarily in order to identify the paths.

Remark 3. Since two different kernels may lead to the same image curve, the graph spanned by the ℓ^e-torsion generators (P, Q) may not always correspond to a tree. However, for simplicity of analysis, we assume this does not happen, although such cases do not pose a problem in practice.

Remark 4. In SIKE, a party computes a full ℓ^e-isogeny using an ℓ^e-torsion basis (P, Q). In other circumstances, like in tree computation or in the meet-in-the-middle attack, we need to compute only the initial part of such full walks: thus, to keep Definition 6 consistent, such full torsion basis needs to be re-scaled, so that the path length matches the desired one.

For a walk of length i, the re-scaling is done as

$$(P', Q') = ([\ell^{e-i}]P, [\ell^{e-i}]Q).$$

so that all length-i walks arising from $P' + [t]Q'$ with $t \in [0, \ell^i]$ will match the first i steps of length-e walks arising from $P + [s]Q$ with $s \in [0, \ell^e]$.

It follows that, to succeed in a meet-in-the-middle attack, it is crucial to be able to generate trees (more precisely, their leaves) from curves.

Problem 3 (Tree generation)**.** *Given a supersingular curve E defined over $\mathbb{F}_{p^2}$ and an ℓ^e-torsion basis (P, Q) for it, compute the set of j-invariants of curves appearing as leaves in the depth $d \leq e$ tree spanned by (P, Q).*

3.3 Tree Generation Strategy

In this section, we address how it is possible to generate leaves of the tree spanned by some torsion generators.

If, in SIKE, a MitM attack relative to one party, e.g. Alice with her full torsion $E[\ell^{e_A}]$, succeeds, then the shared secret can be efficiently computed from the data she exchanged with Bob. It thus suffices to run the attack only on one full torsion and, for the sake of

simplicity, we will hereafter address only the case $\ell = 2$, i.e. attack $E[2^{e_A}]$. In this setting, we can take advantage of efficient 2-isogeny computation formulas and simpler formulation of tree generation and exploration, although we remark that the following discussion can be generalized to the $E[3^{e_B}]$ torsion as well.

A straightforward approach for generating a tree, is to enumerate all possible $s \in [0, 2^e - 1]$ and compute the respective isogeny's image curve, similarly as done in SIKE for a given private key s. In fact, such walk computation is performed as a *single step* in the low-memory van Oorschot-Wiener collision search applied to SIKE [ACC+19, CLN+20, LWS21]. However, many intermediate curves will be visited multiple times for different values s. More precisely, if two different s_0 and s_1 share the same k least significant bits of their binary representations, then the first k steps in the walks arising from $P + [s_0]Q$ and $P + [s_1]Q$ will be (partially) identical. To better understand the complexity of such naive approach, a depth-e tree has $2^{e+1} - 1$ nodes and $2^{e+1} - 2$ edges, while here we would walk through $e2^e$ edges, a logarithmic slowdown (in the tree size) with respect to other tree exploration techniques. In addition, lower-depth edges are typically more expensive to compute due to the larger scalar multiplication required to obtain an order-2 point (e.g. in SIKE isogeny evaluation algorithms), increasing the performance gap further.

We also note that, although s_0 and s_1 share the first k bits, the initial kernel generator points pushed through the two walks differ, and therefore the computations done on the shared sub-walks are not fully identical and cannot be trivially avoided by caching.

A better and more natural way to explore the tree is through a *depth-first traversal*, which also avoids a large memory footprint.

To better explain how it works, we will label tree nodes with the A_i coefficients of the corresponding curve E_{A_i}. Given a path from the starting node A_0 to the current node A_i, made of composition of 2-isogenies, we explore the node A_i by generating its 2 children nodes, each identified by an order-2 point on E_{A_i} with nonzero x-coordinate (as we noted in Subsection 2.2, the remaining third order-2 point corresponding to $(0, 0)$ is automatically excluded and this corresponds to the edge pointing backwards from a node towards the root). However, to the best of our knowledge, all known generic ways to iteratively compute coordinates of these order-2 kernel generators in a walk, require extracting an expensive square root in $\mathbb{F}_{p^2}$.

Our goal is to avoid this heavy operation, while maintaining 2 distinct order-2 kernels at each step (one for each possible child direction) on the way through the tree exploration. In addition, we want to take advantage of the efficient formulas for 2-isogenies, so we need to ensure that $(0, 0)$ never appears as one of the kernel generators, similarly as in [Ren18, Corollary 2].

Approach outline Our solution is based on a modified isogeny evaluation algorithm from SIKE. For a depth d node A_i, we store a basis P', Q' of $E_{A_i}[2^{e-d}]$, with the constraint that $[2^{e-d-1}]Q' = (0, 0)$. The children nodes are then reached by the two isogenies corresponding to the order-2 points $[2^{e-d-1}]P'$ and $[2^{e-d-1}](P' + Q')$, respectively, for which it is ensured that none of them equals $(0, 0)$, thus allowing efficient SIKE arithmetic implementation. As scalar multiplication by a large power of 2 is expensive (it is comparable to the field square root cost), we offset such operations by selectively pushing through isogenies few intermediate points, as done in SIKE, with the additional effort of ensuring they form a good basis. Finally, the optimal strategy - a trade-off between the number of point doublings and isogeny evaluations - can be computed using dynamic programming, similarly to how it was done in the SIDH paper by Jao, De Feo and Plût [JD11, DJP11], in order to compute the full walk up to a certain depth. Our general approach is easily parallelizable by distributing subtree generation tasks among available workers.

3.3.1 Maintaining Torsion Basis for Efficient Isogeny Computations

We now describe a method which allows us to maintain, during path traversals, a basis suitable for the efficient arithmetic formulas used by SIKE, i.e., the ones detailed in Subsection 2.3.

Proposition 3. *Let $A \in \mathbb{F}_{p^2}$ and $e \geq 2$. Let $P, Q \in E_A(\mathbb{F}_{p^2})$ be a basis of $E_A[2^e]$ with $[2^{e-1}]Q = (0,0)$. Then, for a 2-isogeny $\phi : E_A \to E_{A'}$ arising from $[2^{e-1}](P + [s]Q)$ with $s \in [0, 2^e - 1]$,*

1. *if $\ker\phi = \langle [2^{e-1}]P \rangle$, then $P', Q' \in E_{A'}(\mathbb{F}_{p^2})$ is a basis of $E_{A'}[2^{e-1}]$ with $[2^{e-2}]Q' = (0,0)$, where*

$$\begin{aligned} P' &= \phi(P), \\ Q' &= \phi([2]Q); \end{aligned}$$

2. *if $\ker\phi = \langle [2^{e-1}](P+Q) \rangle$, then $P', Q' \in E_{A'}(\mathbb{F}_{p^2})$ is a basis of $E_{A'}[2^{e-1}]$ with $[2^{e-2}]Q' = (0,0)$, where*

$$\begin{aligned} P' &= \phi(P+Q), \\ Q' &= \phi([2]P). \end{aligned}$$

Proof. Since P, Q are distinct generators and both have order 2^e, it follows that the 3 order 2 points $[2^{e-1}]P, [2^{e-1}]Q, [2^{e-1}](P+Q)$ generates the $2+1$ distinct subgroups of $E[2] = \mathbb{Z}_2 \times \mathbb{Z}_2$. Since $[2^{e-1}]Q = (0,0)$, the order-2 point $[2^{e-1}](P + [s]Q)$ appearing as a kernel for ϕ can only be equal to either $[2^{e-1}]P$ or $[2^{e-1}](P+Q)$. If $\ker\phi = \langle [2^{e-1}]P \rangle$ we immediately have $\phi([2^{e-1}]P) = \mathcal{O}_{E_{A'}} = [2^{e-1}]P'$ and since $\phi([2^{e-2}]P) \neq \mathcal{O}_{E_{A'}}$, $P' = \phi(P)$ must then be a generator of $E[2^{e-2}]$. Since 2-isogenies formulas arising from $P + [s]Q$ have the property to fix the point $(0,0)$ (see Remark 2), we then have $\phi([2]Q)$ has order 2^{e-1} and is such that $[2^{e-2}]\phi([2]Q) = \phi((0,0)) = (0,0)$.

Similarly, if $\ker\phi = \langle [2^{e-1}](P+Q) \rangle$, then $P' = \phi(P+Q)$ has order 2^{e-1}. It follows that $\phi([2^{e-1}]P) + \phi([2^{e-1}]Q) = \mathcal{O}_{E_{A'}}$, i.e. $[2^{e-2}]Q' = \phi([2^{e-1}]P) = -\phi([2^{e-1}]Q) = (0,0)$.

For P' and Q' to form a basis, we further need to show that $\langle P' \rangle \cap \langle Q' \rangle = \mathcal{O}_{E_{A'}}$. Let us assume, by contradiction, that there exists a non-trivial $R \in \langle P' \rangle \cap \langle Q' \rangle$: we then have, for certain $s, t \neq 0$, that $R = [s]P' = [t]Q'$ and thus $[s]P' - [t]Q' = \mathcal{O}_{E_{A'}}$. If $\ker\phi = \langle [2^{e-1}]P \rangle$, we then have that $[s]P - [t]Q$ is in $\ker\phi$ and thus $[2^{e-1}]P = [s]P - [2t]Q$. Since P, Q form a basis for $E_A[2^e]$, this in turn implies $s = t = 0$, a contradiction. A similar contradiction is reached also for the case $\ker\phi = \langle [2^{e-1}](P+Q) \rangle$.

□

Our formulation can be used to straightforward map isogenies used to traverse tree nodes to binary strings: using a bit, we can represent the relation between the kernel used to walk a certain step and the (current) torsion generators (e.g. we associate "0" if $\ker\phi = \langle [2^{e-d-1}]\tilde{P} \rangle$ and "1" if $\ker\phi = \langle [2^{e-d-1}](\tilde{P} + \tilde{Q}) \rangle$), as illustrated in Figure 2.

This allows us to easily reconstruct later from such binary strings[4] the full sequence of j-invariants traversed, which in turn can be easily mapped back to the value s whose walk arising from $P + [s]Q$ traverses exactly the same j-invariants.

3.3.2 Optimal Strategies for the Doubling/Isogeny Evaluation Trade-off

During evaluation of the isogeny walk arising from $P + [s]Q$, the order-ℓ kernel for the next step can be obtained through scalar multiplication as $[\ell^{e-1}](P + [s]Q)$. To compute such kernels more efficiently, we can store some intermediate values $[\ell^{e_0-1}](P + [s]Q)$ with

[4] With some abuse of notation, we will often refer to such binary strings as *paths*.

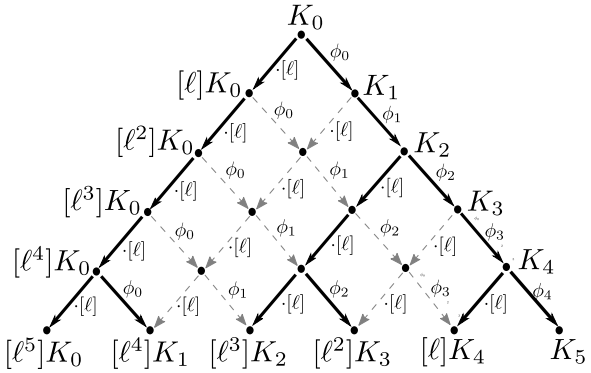


Figure 3: An example of evaluation strategy graph. Multiplication by $[\ell]$ edges ($\swarrow$) and isogeny evaluation edges ($\searrow$) transform $K_0 \in E[\ell^6]$ to the leaf values $\{[\ell^{6-i-1}]K_i\}_{i\in[0,5]}$, needed to compute the walk arising from K_0. In bold, an optimal evaluation strategy assuming $C_{eval} = 1.5 \cdot C_{mult}$ (verified experimentally).

$e_0 < e$, and later push all such points through isogenies and scalar multiplications. Indeed, this allows to compute the kernel of the next-step ℓ-isogeny with just $e - 1 - e_0$ point multiplications by ℓ for the maximum e_0 for which $[\ell^{e_0-1}](P + [s]Q)$ is stored, while storing and pushing smaller multiples will be useful for later steps. It is then clear the relevance of finding good trade-offs between the number of multiplications by ℓ and the number of isogeny evaluations needed to traverse a walk: indeed, depending on the implementation adopted, these two operations have different costs.

In the extended version of [JD11], i.e. [DJP11], De Feo, Jao and Plût describe how to derive an *optimal evaluation strategy* for the best trade-off between scalar multiplications and isogeny evaluations, using the dynamic programming paradigm.

We now provide a brief overview of how optimal evaluation strategies are found in [DJP11]. Let $K_0 \in E_A[\ell^e]$, $\phi_i : E_{A_i} \to E_{A_{i+1}}, i \in [0, e-1]$ be the sequence of isogenies on the length-e walk defined by K_0 and $K_i = \phi_{i-1}(K_{i-1})$ for $i \in [1, e-1]$. The goal is to compute $\ker \phi_i = \langle [\ell^{e-1-i}]K_i \rangle$ for all $i \in [0, e-1]$ in a minimum overall cost in terms of scalar multiplications and isogeny evaluations.

Aiming at this, we construct a directed graph with nodes

$$\left\{ \; [\ell^i]K_j \;\; | \;\; j \in [0, e-1], i \in [0, e-1-j] \; \right\},$$

connected by two types of edges, namely:

- "multiplication by $[\ell]$" edges of cost C_{mult}, connecting $[\ell^i]K_j$ to $[\ell^{i+1}]K_j$, for $i+j+1 \leq e-1$;
- "isogeny evaluation" edges of cost C_{eval}, connecting $[\ell^i]K_j$ to $[\ell^i]K_{j+1}$ (through an ℓ-isogeny ϕ_j), for $i + j + 1 \leq e - 1$.

A strategy for evaluating all the $\ker \phi_i = \langle [\ell^{e-1-i}]K_i \rangle$ can then be described by a tree subgraph in this graph, rooted in K_0 and consisting of directed paths towards the goal leaf nodes $[\ell^{e-1-i}]K_i$ for $i \in [0, e-1]$. The cost of a strategy is then the sum of the costs of the edges in it, counting only once edges traversed by multiple paths. It is then clear that best strategies are those ones in which paths to leaves overlap as much as possible. An example of such graph along with an optimal strategy is illustrated in Figure 3

In [DJP11], it is shown that there exist minimal-cost strategies with recursive structures. The problem is decomposed into two subproblems: the subgraph induced after following a multiplication edges, and the subgraph induced after following $e - a$ isogeny evaluation

edges. This is possible due to the fact that, in paths towards leaves, the order of any two consequent edges can be swapped (if it does not break strategy consistency), since multiplication commutes with isogenies and such swaps do not change the overall strategy cost. An optimal strategy can thus be obtained by evaluating all possible choices of a and solving recursively the induced subproblems. Since the subproblems are fully characterized by their size (and are independent from the root kernel chosen), their solutions can be cached and reused (dynamic programming).

Application to tree generation We are interested in using best strategies during tree generation to make path computations faster.

The difference between the tree generation and a simple isogeny evaluation is that now, each isogeny evaluation edge creates ℓ new exploration nodes deeper in the tree. However, all the ℓ induced sub-trees differ only by curves and generators, and so all can follow the same sub-strategy. Effectively, an isogeny evaluation edge *multiplies* the number of nodes being explored in the isogeny tree by ℓ. To account this, we can then set the weight of an isogeny evaluation edge ϕ_j to ℓ^{j+1}, while we assign to multiplication edges $[\ell^i]K_j \to [\ell^{i+1}]K_j$ a weight of ℓ^j, since in this case the overall number of nodes being explored doesn't change.

Once weights are assigned, the dynamic programming approach can be applied in order to find best strategies on these new graphs. However, in contrast to best strategies for single paths, sub-problems are not fully characterized by their size: edge weights depend, indeed, on where we currently are in the strategy graph. Therefore, we have to solve all sub-problems separately.

Alternatively, we can observe that the cost of a sub-problem of height e rooted at $[\ell^i]K_j$ can be obtained by multiplying by ℓ^j the cost of solving a pure instance (i.e., rooted at $[\ell^0]K_0$) of height $e - i - j$. This reduces the dynamic programming dimension back to 1.

3.3.3 Full Algorithm

We sketch the full attack pseudo-code for the case $\ell = 2$ in Algorithm 1.

4 Further Optimizations

4.1 Final Curve 2-bit Leak

In SIKE, the shared secret key is (computed from) the j-invariant of the image curve E_{AB} of the isogeny resulting from composing Alice and Bob walks in their respective torsions. To allow this, each party publishes intermediate image curves E_A and E_B along with images of others' party torsion basis through their secret isogeny (see Subsection 2.2 for more details). For example, Alice, who computes her 2^{e_A}-isogeny $\phi_A : E \to E_A$, provides Bob a basis $(\phi_A(P_B), \phi_A(Q_B))$ for the 3^{e_B}-torsion of E_A, and the coordinates of such basis leak the final curve itself, i.e. the value $A \in \mathbb{F}_{p^2}$. As was further noticed in [CLN+20], the final value A leaks the j-invariant of the curve visited two 2-isogeny steps before reaching the final curve during her walk: more concretely, it can be shown that the order-4 points $\tilde{Q} = (1, \pm\sqrt{A+2})$ lie in the kernel of the dual of the isogeny $\phi : E_6 \to E_A$, and we can thus easily obtain the j-invariant $j' = j(E_{A'})$ of the curve $E_{A'} = E_A / \langle \tilde{Q} \rangle$ visited two steps before the end.

We note however that, since j-invariants of Montgomery curves are characterized by A^2, the A-value of the curve effectively visited two step before the end remains undetermined: indeed, by solving the equation $j' \cdot (A'^2 - 4) - 256(A'^2 - 3)^3 = 0$ we obtain (at most) 6 solutions for A', and all of them correspond to Montgomery curves isomorphic to the curve visited 2 steps before E_A.

Algorithm 1 Tree generation ($\ell = 2$)

Input: $A_0 \in \mathbb{F}_{p^2}$, (P_0, Q_0) a basis of $E_{A_0}[2^e]$ with $[2^{e-1}]Q_0 = (0,0)$
Output: j-invariants of curves 2^e-isogenous to E_{A_0} through isogenies with kernel $\langle P_0 + [s]Q_0 \rangle$ for some $s \in [0, 2^e - 1]$
Remark: x-coordinate only arithmetic may be directly implemented (details omitted).

function RECURSE(d, path, A_d, L)
  if $d = e$ **then**
    output (path, $j(E_{A_d})$)
    return
  $(P, Q, i) \leftarrow \arg\max_{(P,Q,i) \in L} i$
  $(P', Q') \leftarrow ([2^{e-1-i}]P,\ [2^{e-1-i}]Q)$; add tuples $([2^{i'-i}]P,\ [2^{i'-i}]Q, i')$
    to L according to the optimal strategy (depends on d, i')
  for $b \in \{0, 1\}$ **do**
    $K \leftarrow (P' + [b]Q') \in E_{A_d}$
    $(\phi, A_{d+1}) \leftarrow \phi : E_{A_d} \to E_{A_{d+1}}$ is a 2-isogeny with $\ker \phi = \langle K \rangle$
    $L' \leftarrow \emptyset$
    for $(P, Q, i) \in L, i \leq e - 1$ **do**
      if $b = 0$ **then**
        $(P, Q) \leftarrow (\phi(P), \phi([2]Q))$ ▷ Proposition 3
      else
        $(P, Q) \leftarrow (\phi(P + Q), \phi([2]P))$ ▷ Proposition 3
      $L' \leftarrow L' \cup \{(P, Q, i + 1)\}$
    RECURSE($d + 1$, path$||b$, A_{d+1}, L')

RECURSE(0, (), A_0, $\{(P_0, Q_0, 0)\}$)

We can use 4-isogeny formulas from [JAC+20], in order to detect which coefficients A' can be pushed directly to the final curve E_A through the 2-isogeny formulas from Proposition 1. For an order 4 point $(x_k, y_k) \in E_{A'}$ defining the isogeny $\phi : E_{A'} \to E_A$, we have that $A = 4x_k^4 - 2$. It then suffices to check for which of the 6 candidate values for A', the point on $E_{A'}$ with x-coordinate $(-i)^n \cdot \sqrt[4]{\frac{A+2}{4}}$, with $n \in [0, 3]$, has order 4.

Since $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$, the map $(x, y) \mapsto (-x, iy)$ defines an isomorphism between $E_{A'}$ and $E_{-A'}$, thus the order-4 point in $E_{A'}(\mathbb{F}_{p^2})$ with x-coordinate $(-i)^n \sqrt[4]{(A+2)/4}$ and $n \in [0, 3]$, corresponds to an order-4 point in $E_{-A'}(\mathbb{F}_{p^2})$ with x-coordinate $(-i)^m \sqrt[4]{(A+2)/4}$, where $m = n + 2 \pmod 4$.

From Section 2.3 and Figure 1, we know that there are 4 coefficients $\pm B', \pm C'$ out of 6, each corresponding to a curve with j-invariant equal to the j-invariant of the curve visited two step before E_A, that satisfy the final 4-isogeny with respect to E_A: it is then enough to pick any of the 2 remaining coefficients $\pm A'$, and set the updated final curve to $E_{A'}$: in this way, all isogenies from this curve defined as in Proposition 1, will (have at least a) walk towards the starting curve E_6. This allows us to saves a factor of 4 in the tree generated from the final curve if traversals are matched in-the-middle looking at j-invariants.

Alternative expression for A' Interestingly, by taking into consideration the walk structure induced by SIKE 2-isogenies (Section 2.3), the relation between such A' and the final curve coefficient A is very easy to express.

Lemma 1. *Let E_B be a Montgomery supersingular elliptic curve over $\mathbb{F}_{p^2}$ with $p \neq 2$, and let $K_0, K_1 \in E_B(\mathbb{F}_{p^2})$ be two order 2 points distinct from $(0,0)$. By applying the 2-isogeny*

formulas from Proposition 1 to the groups generated by K_0 and K_1, we obtain, respectively, two isogenies $\phi_1 : E_B \to E_A, \phi_2 : E_B \to E_{A'}$ such that

$$(A-2)(A'-2) = 16.$$

Proof. Let $\tilde{x}, \tilde{z} = 1/\tilde{x}$ be the roots of $x^2 + Bx + 1 = 0$. Then $\tilde{x}, \tilde{z}$ are the x-coordinates of K_0 and K_1. By applying the 2-isogeny formulas from Proposition 1 on these two points, we then obtain $A = 2 - 4\tilde{x}^2$ and $A' = 2 - 4\tilde{z}^2$. It then immediately follows that $(A-2)(A'-2) = (-4)^2 \cdot \tilde{x}^2\tilde{z}^2 = 16$. □

Let us now consider the last 3 traversed nodes in Alice's walk, i.e. the j-invariant of $E_{A'}$, followed by a middle node j', and the final $j(E_A)$. Then there exists a $B \in \mathbb{F}_{p^2}$ so that $j(E_B) = j'$, which is pushed, depending on the kernel chosen, through 2-isogenies to $-A$ and A' (cf. Figure 1). Note that we use $-A$ instead of A, since otherwise we would select the $\pm B$ from the original path which only has a backwards edge towards $j(E_{A'})$ (i.e., an isogeny with the kernel $\langle(0,0)\rangle$). From Lemma 1, we then conclude that

$$A' = 2 - \frac{16}{A+2} \qquad \text{(two final 2-isogenies)}$$

This equation assumes that the last two steps consist of a sequence of two 2-isogenies. If a 4-isogeny is used instead (as in the case of SIKE challenges), this decomposes, according to specification [JAC+20], into a sequence of two 2-isogenies followed by a sign flip of the pushed curve coefficient. We then need to flip the sign of the coefficient of the final curve to match the assumptions of Lemma 1, thus obtaining

$$A' = 2 + \frac{16}{A-2} \qquad \text{(one final 4-isogeny)} \tag{3}$$

4.2 Storing Conjugation Representatives

In SIKE, the starting curve is chosen to be $E_6(\mathbb{F}_{p^2})$, and since $A = 6 \in \mathbb{F}_p$, the Frobenius map $\pi : (x, y) \mapsto (x^p, y^p)$ defines an automorphism for $E_6(\mathbb{F}_{p^2})$. As already noticed in [CLN+20], this implies that for any kernel $\langle R\rangle \subset E_6$, $j(E_6/\langle R\rangle)^p = j(E_6/\pi(\langle R\rangle))$, that is pairs of conjugate kernels give rise to paths to curves having conjugate j-invariants. By recalling that π fixes $\mathbb{F}_p$, from the above we further easily obtain that if E_A is isogenous to E_6, then $E_{\bar{A}}$ is isogenous to E_6 as well, since $\overline{j(E_A)} = j(E_{\bar{A}})$.

We take advantage of this fact to approximately halve the time complexity of the meet-in-the-middle-attack. It is indeed sufficient to explore non-conjugate subtrees starting from E_6, and store the norm of j-invariants in the middle to be able to detect intersections of j-invariants. It can be shown that at any (non-trivial) depth of the tree expanded from E_6, there exists exactly two curves with their A-coefficients in $\mathbb{F}_p$, while all the other nodes are conjugate pairs (each arising from two conjugate subtrees). This means that at a certain depth $e > 0$, we have exactly $2^{e-1} + 1$ different norms for j-invariants, which can be computed by exploring just $\frac{2^{e-1}+1}{2^e} \approx \frac{1}{2}$ of the tree expanded from E_6.

If the j-invariants found in the intersection during the meet-in-the-middle attack have same norm but are conjugates, we need to retrieve the correct "conjugate path" on the tree from E_6, in order to solve Problem 1. From the above properties, the sequence of j-invariants on the path from E_6 to a curve with j-invariant j', is an element-wise conjugate of the sequence towards a curve of j-invariant $\overline{j'}$: it is then enough for a kernel $P + [s]Q$ going to j', to suitably flip bits in s, so that the resulting path walks step-by-step over nodes with conjugate j-invariants.

As a final note, SIKE works in the quadratic extension $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$, so elements can be seen as complex numbers over $\mathbb{F}_p$. Conjugates can be then obtained by simply changing sign to elements' imaginary parts, and in place of expensive to compute norms, we can store as a unique representative of a conjugation class just real parts of j-invariants in the middle.

4.3 Set Intersection

4.3.1 Hash-tables or Sort and Merge?

A standard way to implement the final stage of our meet-in-the-middle attack, i.e., finding elements common to the two tree j-invariant norm datasets, is by using hash-tables: we fill one of such tables with entries from the first dataset, and we then lookup every element in the second one. In theory, the amortized cost of an hash-table lookup would be $\mathcal{O}(1)$, but in practice, *random* memory accesses get slower and slower as the table size grows and memory *latency* starts dominating execution time.

An alternative approach is to *sort* the two datasets and perform a linear-time *merge* operation by keeping common elements only, an operation requiring *sequential* memory accesses. The drawback of this approach is that the sorting step has quasilinear complexity $\mathcal{O}(n \log n)$ in the (biggest) dataset size n, and to complete it we need memory access patterns which are not necessarily sequential.

In order to compare these two approaches, we implemented a simple hash-set for 64-bit integers with linear scanning and double-sized buffer (i.e., to store n elements, the structure allocates memory for $2n$ elements). In the following, we will refer to such custom hash-set with the name FastHash. In our experiments, it outperforms the default C++ `unordered_set` (compiled on `g++ 9.3.0`) more than a few times. We then implemented the sort and merge approach (denoted `SortMerge`).

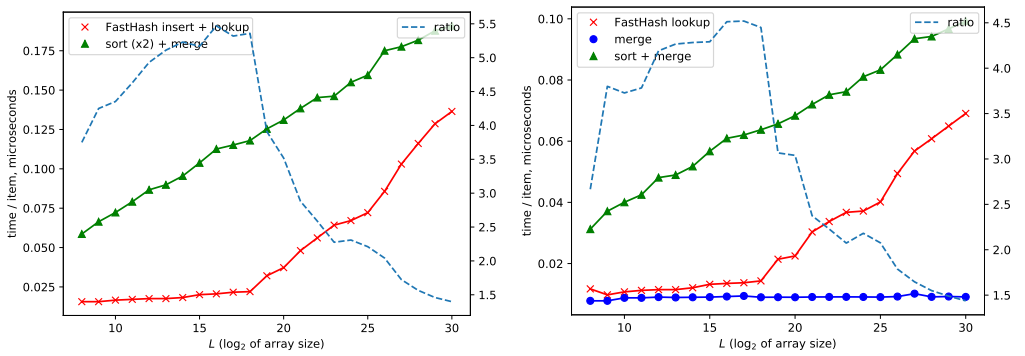
In Figure 4, we provide different benchmarks for both FastHash and SortMerge at different array sizes 2^L.

More precisely, in Figure 4a we compare the time to intersect two unsorted 64-bit integers arrays, assuming no preprocessing of the input datasets. Here, the FastHash-based approach first inserts all 2^L elements of the first dataset, and then performs lookups of the 2^L elements of the second dataset. In SortMerge, instead, the two datasets are first sorted (using C++ `sort()`) and then merged using a two-pointers linear scan. Although FastHash outperforms the sorting approach on up to $2^{30} \approx 10^9$ entries, the advantage ratio decreases quickly from an initial value of 4 (for $L = 8$) to a ratio close to 1 for $L = 30$. In particular, a sharp advantage drop is visible after $L = 18$, which is likely related to the dataset not fitting the CPU cache (Intel® Core™ i5 10210U 1.6-4.2 GHz).

In Figure 4b, we compare the two methods under some allowed precomputations. For FastHash this means that only lookups are counted (insertions are excluded from time computation), while for SortMerge we consider two cases: i) the first dataset is pre-sorted, that is the timings include sorting the second dataset only and merging (in green); ii) both datasets are pre-sorted, that is the timings include only merging (in blue). We can see that the pure merging cost remains constant for any array size, and is negligible compared to both FastHash lookups and sorting.

Parallelization When dataset sizes are large, efficient parallelization techniques are a requirement. The most straightforward approach for parallelizing intersection finding, consists in splitting the input datasets A and B in k (equally sized) chunks $(A_0, \ldots, A_k)$ and $(B_0, \ldots, B_k)$, and then intersect all k^2 distinct pairs $A_i \cap B_j$ independently in parallel. Clearly, this parallelization comes at the cost of k times more work than standard lookups/merges, but can be acceptable if k is small.

An advantage of this approach, is that each chunk can be preprocessed independently, so that each of the k^2 chunks intersection takes preprocessed data as input. From Figure 4 it is clear that already for $k = 2$, SortMerge (which requires an amortized number of 2 sorting and k merges/interesections per chunk) outperforms the FastHash hash-set approach (which requires 1 chunk insertion and k chunks lookups per chunk). In Figure 5, we illustrate how these two approaches perform, for different values of k, on k chunks each of size 1GiB.



(a) Intersecting two arrays without preparation (b) Lookup in a prepared array

Figure 4: Performance comparison between FastHash and SortMerge over 64-bit integer arrays of total size 2^L.

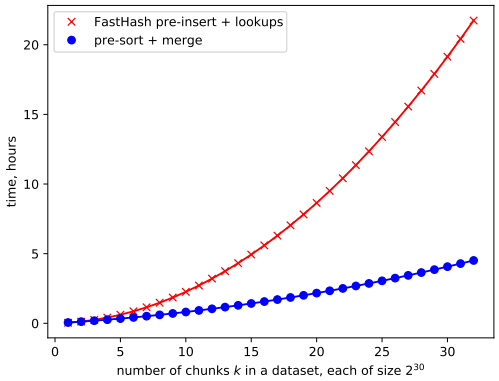


Figure 5: Performance comparison between FastHash and SortMerge running on parallel over k 64-bit integers chunks, each of size 2^{30} (extrapolated using timings from Figure 4b).

4.4 Storage-Collisions Trade-off and Compression

Real parts (or, in general, norms) of j-invariants belong to $\mathbb{F}_p$: in our meet-in-the-middle attack, each tree has only approximately $2^{e_A/2} \approx p^{1/4}$ leaves, therefore, the chance of having multiple collisions is negligible.

However, we would like to reduce storage requirements as much as possible. This can be done by reducing the number of bits we use to represent j-invariants, while allowing only a reasonable amount of false positive collisions. More concretely, if we use n bits to represent j-invariants, we then expect to observe approximately $(p^{1/4})^2/2^n$ collisions.

In addition, paths associated to j-invariants in the middle (useful to quickly test full isogenies associated to collisions) may be omitted too. This comes at the cost of an extra (memoryless) tree exploration, required to recover full j-invariants associated to a colliding representation[5] and the respective paths in the trees.

In some cases, it is possible to further reduce memory requirements. If the n-bit j-invariant representations are uniformly distributed, we can compress *sorted* chunks of 2^m such elements by noticing that two consecutive elements are expected to differ, on average, by 2^{n-m}. We can then store only such reduced differences, reserving 1 *flag* bit in order

[5]Since just few false positive collisions are expected, recovering full $2\log p$-sized j-invariants is useful to immediately detect their correct conjugate branch in the tree expanded from E_6.

to distinguish between a small difference from an n-bit full representation, in case two elements differ by more than 2^{n-m} (in practice, a larger difference is allowed to capture more small elements). This, in fact, reduces memory requirements from $n2^m$ bits to at most $\approx (n-m+1)2^m$ bits, with different implementation-specific wordsize trade-offs in the middle.

We note that by requiring chunks to be sorted, this compression technique goes towards the SortMerge intersection finding approach we detailed in Subsection 4.3: indeed, merges can be performed equivalently by using two extra n bits registers, in which we store the n bits values to compare, obtained by iteratively adding differences to each dataset's first uncompressed element.

5 Cryptanalysis of the $IKEp182 Challenge

In this section we will detail how all the above ideas allowed us to break the $IKEp182 challenge [Mic21b], a small parameters specification-compliant SIKE instance generated by Microsoft in a live event during the 3rd NIST PQC Standardization conference.

In $IKEp182, the field characteristic is equal to $p = 2^{91}3^{57} - 1$. According to the specification, we have $\mathbb{F}_{p^2} = \mathbb{F}_p(i) = \mathbb{F}_p[x]/(x^2+1)$, $\#E_6(\mathbb{F}_{p^2}) = (p+1)^2$ and $E_6[2^{91}] = \langle P_A, Q_A \rangle$, $E_6[3^{57}] = \langle P_B, Q_B \rangle$ with

$$
\begin{aligned}
P_A = (&\texttt{0x05a324935a4d7b75024fdc3601fe8b5888cea9f88212b2 +}\\
&\texttt{0x02357bdd576772bf2a93e3d680ed7306e16eafc6aff904}\cdot\texttt{i},\\
&\texttt{0x242a9e09aa8e6995e4fdce9f68e8c2c902154c332de68a +}\\
&\texttt{0x011b23646f8884b7a9faa5159ef13842880ed0f9f43dcd}\cdot\texttt{i})\\
Q_A = (&\texttt{0x27b8def415bae0506a9607fff7704832151cdcbc93cb22 +}\\
&\texttt{0x085c86f386b94b8c413f5e49736f26de95103a9b65f31a}\cdot\texttt{i},\\
&\texttt{0x16af6790fb0f5cfd0e124033bb7619e2f75a25cae5f42b +}\\
&\texttt{0x172567b99058dd9d5b99ce5ea4bacd685f57c8326011a3}\cdot\texttt{i})\\
P_B = (&\texttt{0x02ca3bc7e98f88b3ca3239c276eb7a224c51f61bc8c5ed},\\
&\texttt{0x262a38701d1b61dd8875909ff268a50d912f620db980a1})\\
Q_B = (&\texttt{0x02dcff7123e2380f552f5bff91da77ae62e9556b866d8f},\\
&\texttt{0x06aeb7c764aa40913b3fc784d569833d4226cc4a53432f}\cdot\texttt{i})
\end{aligned}
$$

Our meet-in-the-middle attack will target the 2^{91} torsion and will recover Alice's full 91-steps walk in the Supersingular 2-Isogeny Graph. After a quick Setup, the full attack will consist of 5 main stages: Trees Expansions, k-way Merge, Compression, Sieving and Final Trees Expansions.

Setup In the first step of the SIKE protocol (Subsection 2.2), Alice sends to Bob a compressed representation of the points $\phi_A(P_B)$, $\phi_A(Q_B)$, consisting of the following 3 x-coordinates

$$
\begin{aligned}
x_{\phi_A(P_B)} &= \texttt{0x17d02d323c815eee1ec75f1c675609b0bea78064cb8cc1 +} \\
&\quad \texttt{0x12fa80de8027f68c3f780b5bcd519e8205606ac249025d·i} \\
x_{\phi_A(Q_B)} &= \texttt{0x272c54d49af950b0829072753e3525091aaf87085bd7b2 +} \\
&\quad \texttt{0x23efe3c087965a49fcc5161e6453dbe632d7dec90bab12·i} \\
x_{\phi_A(Q_B)-\phi_A(P_B)} &= \texttt{0x22c38abb1427245de1e049408dab87ed9ba54efeb4a4e4 +} \\
&\quad \texttt{0x0c5d768e87a762b6a460b941bcc5537ba0f73ce8b9f955·i}
\end{aligned}
$$

Such representation is justified by use of efficient implementations which exploits x-only arithmetic: we refer to [JAC+20] for more details.

If we denote the tuple $(x_{\phi_A(P_B)}, x_{\phi_A(Q_B)}, x_{\phi_A(Q_B)-\phi_A(P_B)})$ as (x_P, x_Q, x_{Q-P}) we obtain [CLN16, Section 6] the A coefficient of the Montgomery curve E_A on which the points $\phi_A(P_B)$, $\phi_A(Q_B)$ lie, as

$$
\begin{aligned}
A &= \frac{(1 - x_P x_Q - x_P x_{Q-P} - x_Q x_{Q-P})^2}{4 x_P x_Q x_{Q-P}} - x_P - x_Q - x_{Q-P} \\
&= \texttt{0xc0cbda5ef968048cd2c1b125774f1417125b9b02b6f91 +} \\
&\quad \texttt{0x1e8121a2a60fd266d321bb9db8d9e3111e3095c08e0bc6·i}
\end{aligned}
$$

To take advantage of the final 2-bit leak described in Subsection 4.1, we computed the coefficient A' such that $j(E_{A'})$ lies on the (secret) traversed path 2 steps before the final curve, and the SIKE-tree arising from A' does not go towards the final curve E_A. This can be achieved by using (3), i.e., $A' = 2 + 16/(A - 2)$, to obtain:

$$
\begin{aligned}
\tilde{A}' &= \texttt{0x164db610b03a9b3c38e59bf29485a60462d1cd9f22d95e +} \\
&\quad \texttt{0x1a8d75d6d0285807042e900df3c2cf74b4eb160d50a92e·i} \\
j(E_{A'}) &= \texttt{0xe48a8271ea06ec4193db09970a23bea55c777ef2fb5be +} \\
&\quad \texttt{0x56910191b4835901ef45e4b857817391ad1213080afa9·i}
\end{aligned}
$$

The Setup phase was implemented in SageMath [The21].

Trees Expansions We set $A' = \tilde{A}'$, and we proceed by attacking the 89-steps path in the isogeny graph between $j(E_6)$ and $j(E_{A'})$. Note that there may be no path in the SIKE-tree (Definition 7) between the exact *curves*, as we might chose a different representative curve, but there must exist a path in the isogeny graph between $j(E_6)$ and $j(E_{A'})$, and the SIKE-trees arising from E_6 and $E_{A'}$ must contain paths following this path by j-invariants (from the opposite endpoints). In order to meet in the middle, we generate in a depth-first manner the SIKE-tree expanded from E_6 (up to the depth 45) and the SIKE-tree expanded from $E_{A'}$ (up to the depth 44), employing the optimal strategies detailed in Subsubsection 3.3.2.

We note that, as discussed in Subsection 4.2, it suffices to explore only half of conjugate sub-branches of the tree expanded from E_6: this results in an almost equal number of leaves in-the-middle generated from both trees, with a total of $2^{44} + 1$ leaves for the tree expanded from E_6, and 2^{44} leaves for the one expanded from $E_{A'}$.

Once the depth-first generation reaches a leaf, we compute the corresponding j-invariant and we store the least significant 64 bits of its real part. In our implementation, multiple jobs explore in parallel distinct branches of each tree: when a job collects 2GiB of 64-bit j-invariant representations (which correspond to 2^{28} j-invariants visited), this chunk is sorted in-memory, written to disk, and then the job terminates. On the University of

Luxembourg High Performance Computing (HPC) facilities [VBCG14], each of these job took approximately 17 minutes to complete on a single core of an Intel® Xeon® E5-2680v4 @ 2.4GHz with 4GiB of RAM reserved. This sums up to a total of approximately 4.2 core-years and 256TiB of disk space needed to explore both trees and store the truncated j-invariants.

Remark 5. By utilizing the Merge and Compression earlier - on the fly after a sufficient amount of chunks is generated, the storage requirement could be reduced to close to 128TiB.

k-way Merge We employed our custom *k-way merge* implementation optimized for 64-bit unsigned integers, to merge the 2GiB sorted chunks generated from each tree: on a single core of an Intel® Xeon® E5-2680v4 @ 2.4GHz and 4GiB of RAM, we needed approximately 2.5 core hours to merge 256 2GiB chunks into a single 512GiB sorted chunk. We note that, to keep memory requirements close to the ones needed to store all j-invariants representations, chunks can be merged in parallel to the depth-first expansions, as soon as enough new 2GiB chunks from a certain tree are generated. Practicality of running multiple such merges in parallel depends, however, on storage architecture, cluster load and maximum disks I/O throughput: on the University of Luxembourg HPC cluster, we were able to run 4 nodes in parallel, running 28 merge jobs each, without degrading too much I/O performances. This merging stage took, overall, approximately 54 core days.

Compression Since 512GiB chunks contain already 2^{36} 64-bit elements each, at this point we ran single-core jobs to merge 4 chunks directly in compressed form (Subsection 4.4), using 32 bits (including 1 flag bit) to encode elements differences. This resulted in a compression factor very close to $\frac{1}{2}$. In the same configuration as above (and under the same limitations), we needed roughly 5 core hours to complete one of such merge-to-compressed job (we ran only 2-3 nodes concurrently, each executing 28 such jobs), for a total of 27 core-days to complete all jobs.

We then finally obtained 64 1TiB compressed chunks from each tree, for a total of 128TiB disk space used (all previous sub-chunks were deleted).

Sieving At this point we proceed with finding elements shared by chunks from different trees. Since chunks are sorted already, we can use the parallel version of SortMerge with parameter $k = 64$ detailed at the end of Subsubsection 4.3.1. This stage consists in merging tuples of (compressed) 1TiB chunks and storing only the common elements. If ran in a single thread on the full data, this stage only requires sequential read of the 128TiB of data. However, the heap operations in k-way merge dominate the performance and can not be parallelized. In our implementation, a sieving job consisted in merging at the same time 4 chunks from the first tree with 4 chunks from the second tree, by decompressing elements and storing only collisions: on a single core, it took approximately 1.1 core days to complete, for a total of 280 core-days for 256 such jobs. This trade-off results in 2PiB of data read, which is acceptable to allow sufficient parallelization.

We expected and we actually found $16\,777\,119 \approx 2^{44 \cdot 2}/2^{64} = 2^{24} = 16\,777\,216$ 64-bit collisions among the two trees: once such collisions are safely stored, we can delete all the 128 1TiB chunks from previous stages.

Final Trees Expansions With the collisions just found, we run the tree explorations again, similarly as in the first stage of the attack, but this time we store only full j-invariants in the middle that have the least 64 bits of their real part matching any of the collisions found, and their paths in the respective trees.

During tree explorations, we regularly check if were found j-invariants from different trees that share same real part: if yes, we stop tree traversals, and we reconstruct Alice's

full walk from E_6 to E_A (and thus her secret) using the paths associated to the matching j-invariants, with special care in case the two result to be conjugate (Subsection 4.2).

In our case, the colliding conjugate j-invariants in-the-middle obtained by expanding the trees from E_6 and $E_{A'}$ were, respectively,

$$\begin{aligned} j_0 &= \texttt{0x0008132653e4d53cb9cc0defb36a0141d900adbb128a24f0} + \\ &\quad \texttt{0x0001049f06c78aaed22786dfcff5b202ce3a50429f369b86} \cdot \texttt{i} \\ j_1 &= \texttt{0x0008132653e4d53cb9cc0defb36a0141d900adbb128a24f0} + \\ &\quad \texttt{0x0027910d1a0d795d077f40d1480a4dfd31c5afbd60c96479} \cdot \texttt{i} \end{aligned}$$

Using the (implementation-dependent) path information we stored, we then reconstruct, in linear time, the Alice's private key as

$$s_A = \texttt{0x59d64d476da9487be414734}$$

which allowed us to easily compute Alice's and Bob's shared secret from Bob's public key exchanged, as

$$\begin{aligned} j(E_{AB}) = \texttt{0x7a470546a24124f06f49bcbb855a6e3c1402ba1004bfc} + \\ \texttt{0x1a88f02557168dd75b64f8407a368aa4ff2bc03121fbaf} \cdot \texttt{i} \end{aligned}$$

whose value is a correct pre-image for the publicly released SHA512 hash of the challenge shared secret [Mic21b].

We found the solution to the challenge after exploring approximately 44% of the tree expanded from E_6 (only conjugate-unique sub-branches) and 63% of the tree expanded from $E_{A'}$ (success probability of $\approx 28\%$).

This brings the total cost of our attack to approximately 8.5 core years and 256TiB of disk memory.

We note however that we decided to employ compression only during the execution of the above attack, in order to reduce the amount of not fully parallelizable disk reads needed for the parallel SortMerge. Thus, in fact, the whole attack can be executed in 8.5 core years with just slightly more than 128TiB of disk memory available. The storage requirement can be reduced further by sacrificing parallelization and performing the main steps for a single part of the second tree at a time. In our case, we used 4-chunk groups (4TiB) on each side and so only $64 + 4 = 68$TiB of storage is sufficient for the (less-parallel) attack.

6 Conclusions

In this work, we showed how the $IKEp182 challenge can be broken in practice. A natural question is whether the $IKEp217 challenge is reachable for attacking using our method.

In $IKEp217, the prime p is equal to $2^{110}3^{67} - 1$, so that $e_A = 110$, leading to 192PiB storage requirement if our attack on $IKEp182 is applied directly and 64-bit j-invariant representations are stored (which may produce a large number of collisions, namely $2^{107-64} = 2^{43}$).

On the ULHPC cluster, the main limitation is the I/O performance, which is about 20GiB/s[6]. Even if arbitrary storage size is available, this maximum throughput limits the time needed to solve the instance, since full data must be read/written at least once. To read the 192PiB of data on the ULHPC cluster, one would need at least 116 days. Since full attack performs several I/O rounds, the attack would likely take more than a year.

On the other hand, an attacker with a custom highly-parallelized supercomputer may perform the attack much faster. Precise trade-off analysis and parameter selection are left as a future work.

[6] https://hpc-docs.uni.lu/filesystems/gpfs/

Acknowledgements

We thank the ULHPC cluster for providing the computational resources, and Teddy Valette from ULHPC for helping with the project setup. We also thank Microsoft Research for creating the challenge which inspired this work and the Isogeny-based Cryptography School[7] for sparkling our interest in the topic.

References

[AAM19] Gora Adj, Omran Ahmadi, and Alfred Menezes. On isogeny graphs of supersingular elliptic curves over finite fields. *Finite Fields and Their Applications*, 55:268–283, 2019. 4

[ACC+19] 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 Carlos Cid and Michael J. Jacobson Jr:, editors, *SAC 2018*, volume 11349 of *LNCS*, pages 322–343. Springer, Heidelberg, August 2019. 1, 10

[CH17] Craig Costello and Hüseyin Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 303–329. Springer, Heidelberg, December 2017. 6

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 572–601. Springer, Heidelberg, August 2016. 19

[CLN+20] Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes, and Fernando Virdia. Improved classical cryptanalysis of SIKE in practice. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 505–534. Springer, Heidelberg, May 2020. 1, 2, 10, 13, 15

[Cos20] Craig Costello. B-SIDH: Supersingular isogeny Diffie-Hellman using twisted torsion. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 440–463. Springer, Heidelberg, December 2020. 4

[DJP11] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Cryptology ePrint Archive, Report 2011/506, 2011. https://eprint.iacr.org/2011/506. 2, 10, 12

[Gal99] Steven D. Galbraith. Constructing isogenies between elliptic curves over finite fields. *LMS Journal of Computation and Mathematics*, 2:118–138, 1999. 1

[JAC+20] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular Isogeny Key Encapsulation. https://sike.org/files/SIDH-spec.pdf, October 2020. NIST Post-Quantum Cryptography Round 3 - Alternate Candidate. 1, 6, 14, 15, 19

[7] https://isogenyschool2020.co.uk/

[JD11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 19–34. Springer, Heidelberg, November / December 2011. 1, 6, 10, 12

[LWS21] Patrick Longa, Wen Wang, and Jakub Szefer. The cost to break SIKE: A comparative hardware-based analysis with AES and SHA-3. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 402–431, Virtual Event, August 2021. Springer, Heidelberg. 1, 10

[Mic21a] Microsoft Research. *SIDH v3.4 (C Edition)*, 2021. https://github.com/microsoft/PQCrypto-SIDH. 2

[Mic21b] Microsoft Research. *SIKE Cryptographic Challenge*, 2021. https://www.microsoft.com/en-us/msrc/sike-cryptographic-challenge. 2, 18, 21

[Nat22] National Institute of Standards and Technology (NIST). *Post-Quantum Cryptography Standardization*, 2016-2022. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization. 1

[Piz90] Arnold Pizer. Ramanujan graphs and hecke operators. *Bulletin of the American Mathematical Society*, 23:127–137, 1990. 5

[Piz98] Arnold Pizer. Ramanujan graphs. *Computational perspectives on number theory (Chicago,IL, 1995)*, 7:159–178, 1998. 5

[Ren18] Joost Renes. Computing isogenies between Montgomery curves using the action of (0, 0). In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 229–247. Springer, Heidelberg, 2018. 6, 10

[RS06] Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. https://eprint.iacr.org/2006/145. 1

[Sch87] René Schoof. Nonsingular plane cubic curves over finite fields. *Journal of Combinatorial Theory, Series A*, 46(2):183–211, 1987. 5

[Sil09] Joseph Silverman. *The Arithmetic of Elliptic Curves*, volume 106. January 2009. 4

[Sto10] Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Advances in Mathematics of Communications*, 4(2):215–235, 2010. 1

[Tat66] John Tate. Endomorphisms of abelian varieties over finite fields. *Inventiones mathematicae*, 2(2):134–144, 1966. 4

[The21] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.4)*, 2021. `https://www.sagemath.org`. 2, 19

[VBCG14] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos. Management of an academic HPC cluster: The UL experience. In *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*, pages 959–967, Bologna, Italy, July 2014. IEEE. 1, 2, 20

[vW99] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, January 1999. 1