Revisiting Meet-in-the-Middle Cryptanalysis of SIDH/SIKE with Application to 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

Abstract. This work focuses on concrete cryptanalysis of the isogeny-based cryptosystems SIDH/SIKE under realistic memory/storage constraints. More precisely, we are solving the problem of finding an isogeny of a given smooth degree between two given supersingular elliptic curves. Recent works by Adj et al. (SAC 2018), Costello et al. (PKC 2020), Longa et al. (CRYPTO 2021) suggest that parallel "memoryless" golden collision search by van Oorschot-Wiener (JoC 1999) is the best realistic approach for the problem. We show instead that the classic meet-in-the-middle attack is still competitive due to its very low computational overhead, at least on small parameters.
As a concrete application, we apply the meet-in-the-middle attack with optimizations to the $IKEp182 challenge posed by Microsoft Research. The attack was executed on a cluster 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 · Meet-in-the-Middle · Set Intersection

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) [13]. SIKE [9] (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

[⋆] 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 [24] – see https://hpc.uni.lu.

by De Feo and Jao [10] (and Plût [7]), following and improving the ideas of the constructions proposed by Couveignes, Rostovtsev and Stolbunov [6,15,19]. *Isogeny*-based cryptography only recently gained attention and 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 so-called *computational supersingular isogeny* problem, CSSI). The classic meet-in-the-middle attack (MitM, also known as bidirectional search), applied in the isogeny setting by Galbraith [8], requires $\mathcal{O}(p^{1/4})$ time and memory/storage in the SIKE setting. Adj, Cervantes-Vazquez, Chi-Domínguez, Menezes and Rodríguez-Henríquez [1] 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 [23] 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 [5]. Based on this analysis, Longa, Wang and Szefer [11] 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.

In order to motivate security analysis of SIKE, Microsoft recently published two challenges [12] with reduced-size instances of SIKE: \$IKEp182 and \$IKEp217. The classic security of the instances via the meet-in-the-middle attack amounts to only about 2^{45} and 2^{55} isogeny evaluations and storage units, respectively. However, such amount of memory (2^{45} storage units $\geq$ 256 TiB) is not trivial to manage efficiently.

In this work, we revisit the application of the classic MitM attack to the isogeny search problem. While it requires efficient usage of large amounts of memory/storage, it has much lower computational overhead than the vOW method, 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. We focus on optimizing the application of MitM to the SIKE cryptosystem and show how to efficiently use disk-based high-performance storage which is more practical than RAM memory. As a concrete application, we solve the \$IKEp182 challenge on a high-performance cluster. We estimate that the attack can in principle be executed in at most 9.5 core-years using about 70 TiB of high-performance storage. Our implementation is mainly written in SageMath [21] and C++, using parts of the SIDH library [22]. It will be publicly available at github.com/cryptolu/SIKE_MitM.

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 applied and improved several existing optimizations from both MitM and vOW settings in the literature, namely:

2-bit leak from the knowledge of the final curve. In [5], 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 by a factor of 4 the set of j-invariants reachable from the final curve that need to be considered.

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.

1-bit conjugation-based reduction. In SIKE, the initial Montgomery curve is $y^2 = x^3 + 6x^2 + x$, and by being defined over $\mathbb{F}_p$, almost all the curves ℓ-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. half of the trace 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 [5].

Depth-first tree exploration. 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). It was shown in [1] how to perform depth-first tree exploration (denoted MITM-DFS) by maintaining a basis allowing to generate full current subtree. This idea was also partially used in the vOW attack in [5] for precomputing first levels of the tree.

In addition, we developed the following new optimizations:

Efficient arithmetic for MITM-DFS. Whereas the work [1] relied on generic Vélu's formulas for computing isogenies on Weierstrass elliptic curves, we show how to adapt the MITM-DFS tree exploration method to the highly optimized x-only isogeny arithmetic on Montgomery elliptic curves used in SIKE.

Optimal strategy for the tree variant of the isogeny/multiplication trade-off. In the work proposing SIDH [7], the authors showed how to compute an optimal strategic trade-off between the number of ℓ-isogeny evaluations and point multiplications during an ℓ^e-isogeny computation. We show how an analogue of this strategy can be applied to explore the search tree more efficiently. This is an optimization of the MITM-DFS technique.

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* algorithms, which, instead, mostly perform local or sequential access patterns.

A similar idea was suggested by Bernstein [2] for searching collisions of hash functions using a 2-dimensional grid of devices (mesh sorting using an optimal algorithm by Schnorr and Shamir [16]); the authors of [1] also estimated performance of mesh sorting applied to the isogeny path problem, but considered only $p \geq 2^{448}$ and concluded that it is not competitive.

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 the case of \$IKEp182, we used 64-bit entries, which already at 32 GiB 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 Isogenies between Supersingular Elliptic Curves

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 a map is defined over $\mathbb{F}_q$, E and E' are said to be isogenous over $\mathbb{F}_q$. By Tate's Isogeny Theorem [20], two curves are isogenous over $\mathbb{F}_q$ if and only if they have the same number of points in $\mathbb{F}_q$.

In this work, we only consider *separable* isogenies, whose kernel has size equal to the degree of the respective rational map. We call an isogeny of degree ℓ an ℓ-isogeny. Separable isogenies $\phi : E(\overline{\mathbb{F}}_q) \to E'(\overline{\mathbb{F}}_q)$ (up to isomorphism) are in bijections with subgroups G of $E(\overline{\mathbb{F}}_q)$ so that $\ker(\phi) = G$ and ϕ is a $|G|$-isogeny: in such case, the curve E' as group is isomorphic to the group quotient E/G. When $p \nmid \ell$, the ℓ-torsion $E[\ell]$ of an elliptic curve E defined over a field of characteristic p has structure isomorphic to $\mathbb{Z}_\ell \times \mathbb{Z}_\ell$ and $\ell + 1$ cyclic subgroups of order ℓ if ℓ is prime.

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 ℓ is composite, it is possible to decompose a ℓ-isogeny into a composition of isogenies of prime orders. This property allows, in practice, to compute efficiently high (smooth) degree isogenies. More precisely, if $\ell = p_0^{e_0} \cdot \ldots \cdot p_n^{e_n}$ and ϕ is a ℓ-isogeny, then there exists p_i-isogenies $\phi_j^{p_i}$ for $i \in [0, n], j \in [1, e_i]$, so that $\phi = \phi_1^{p_0} \circ \ldots \circ \phi_{e_0}^{p_0} \circ \ldots \circ \phi_1^{p_n} \circ \ldots \circ \phi_{e_n}^{p_n}$.

In the following, we will only consider separable isogenies over *Montgomery elliptic curves*, which are parametrized[3] by $A \in \mathbb{F}_q, A \neq 4$ and are defined by the equation $E_A\ :\ y^2 = x^3 + Ax^2 + x$

Two elliptic curves are $\overline{\mathbb{F}}_q$-isomorphic if they have the same j-invariant. The j-invariant of a Montgomery elliptic curve E_A is equal to $j(E_A) = \frac{256(A^2-3)^3}{A^2-4}$.

The trace t of an elliptic curve E defined over $\mathbb{F}_q$ is the integer satisfying $\#E(\mathbb{F}_q) = q + 1 - t$. An elliptic curve is called *supersingular* if it is defined over a field of characteristic p and has trace t congruent to $0 \mod p$. The j-invariant of a supersingular elliptic curve belongs to $\mathbb{F}_{p^2}$ (see [18, V.3 - Theorem 3.1.a]). In fact, any supersingular curve is isomorphic to an elliptic curve defined over $\mathbb{F}_{p^2}$. From Tate's Isogeny Theorem it follows that the set of supersingular elliptic curves is closed under isogenies. We conclude that the property of being supersingular 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. For any prime p, there exist approximately $\lfloor \frac{p+1}{12} \rfloor$ supersingular j-invariants [17, Theorem 4.6].

For any fixed j-invariant and a positive integer $\ell, p \nmid \ell$, any curve E with $j(E) = j$ has the same set of $\ell + 1$ ℓ-isogenous curves (up to isomorphism), with the isogenies being defined by the distinct order-ℓ subgroups of the torsion $\mathbb{Z}_\ell \times \mathbb{Z}_\ell$. An order-$\ell^e$ kernel on a supersingular elliptic curve defines a decomposition of the respective ℓ^e-isogeny into e ℓ-isogenies, which we shall call a *walk*.

Definition 1 (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 $\mathbb{F}_{p^2}$, 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

[3] More general Montgomery curves are given by $E_{A,B} = By^2 = x^3 + Ax^2 + x$, however the values of B are not relevant for our work.

e (with no backtracking). Kernels of the form $\langle P + [s]Q \rangle$, with $s \in \mathbb{Z}_{\ell^e}$, are 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 are explored.

Problem 1 (CSSI). Given two elliptic curves E and E' and integers ℓ, e such that there exists a separable isogeny $\phi : E \to E'$ of degree ℓ^e, compute ϕ (up to isomorphism) or, equivalently, find (a generator of) the subgroup G of $E[\ell^e]$ such that $E' \cong E/G$.

In practice, it suffices to find a composition of e ℓ-isogenies between the two curves, i.e., a length-e walk from E to E'. Then, a solution to Problem 1 can be efficiently recovered from the composition in a digit-by-digit (in base ℓ) manner.

2.2 SIDH and SIKE

Supersingular Isogeny Key Encapsulation (SIKE) [9] is a post-quantum key encapsulation mechanism (KEM) based on the difficulty of computing/finding an isogeny between two ℓ^e-isogenous elliptic curves (Problem 1). It is based on the Supersingular Isogeny Diffie-Hellman (SIDH) [10] key exchange. The public-key encryption and key encapsulation mechanisms in SIKE are derived from the basic key exchange protocol (SIDH), which we focus on.

In SIDH/SIKE, the prime 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 $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 + 1)$. 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 $E[\ell_B^{e_B}] = \mathbb{Z}_{\ell_B^{e_B}} \times \mathbb{Z}_{\ell_B^{e_B}} = \langle P_B, Q_B \rangle$.

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 $s_A \overset{\$}{\leftarrow} \mathbb{Z}_{2^{e_A}}$ and computes the 2^{e_A}-isogeny $\phi_A : E \to E_A$ with $\ker \phi_A = \langle P_A + [s_A]Q_A \rangle$. She then sends to Bob E_A and the points $\phi_A(P_B)$, $\phi_A(Q_B)$.
- Bob picks secret $s_B \overset{\$}{\leftarrow} \mathbb{Z}_{3^{e_B}}$ and computes the 3^{e_B}-isogeny $\phi_B : E \to E_B$ with $\ker \phi_B = \langle P_B + [s_b]Q_B \rangle$. 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 $\phi_{\tilde{A}} : E_B \to E_{BA}$ with $\ker \phi_{\tilde{A}} = \langle \phi_B(P_A) + [s_A]\phi_B(Q_A) \rangle$ and sets the common secret to $j(E_{BA})$.
- Bob computes the 3^{e_B}-isogeny $\phi_{\tilde{B}} : E_A \to E_{AB}$ with $\phi_{\tilde{B}} = \langle \phi_A(P_B) + [s_B]\phi_A(Q_B) \rangle$ 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 [9, 10].

The original version of SIDH [7] proposed to choose the kernel of the shape $\langle [s]P + [s']Q \rangle$ instead of $\langle P + [s]Q \rangle$. The latter version, adopted in SIKE and in the implementation of SIDH [22], reduces the number of possible kernels from $\ell^{e-1}(\ell+1)$ to ℓ^e. In SIKE, it also avoids some technicalities introduced by adopting efficient 2-isogeny computation formulas: the order-2 point $(0,0)$ is not allowed to belong to the isogeny kernel $\langle P_A + [s]Q_A \rangle$ with $s \in \mathbb{Z}_{2^{e_A}}$. Thanks to a result of Renes [14, Corollary 2], this is guaranteed by choosing the generators P_A, Q_A of the torsion $E[2^{e_A}]$ so that $[2^{e_A-1}]Q_A = (0,0)$.

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 2-isogenies, relevant for SIKE and for our attacks. Proofs that the following formulas define isogenies can be found, for example, in [3,14]. We shall distinguish isogenies based on these specific formulas as "SIKE 2-isogenies".

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

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

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 $A' = 2 - 4x_R^2$.

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

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 the 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.

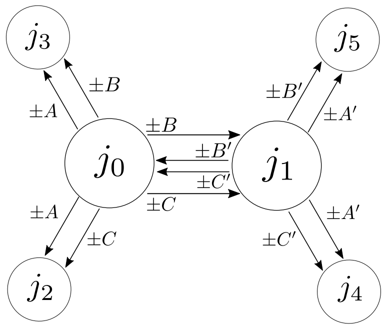


Fig. 1: The different j-invariants reached by 2-isogenies defined by Proposition 1. Here $\pm A, \pm B, \pm C$ are the 6 values of X satisfying $j(E_X) = j_0$ (resp. $\pm A', \pm B', \pm C'$ and j_1), and define 6 Montgomery curves isomorphic over $\mathbb{F}_{p^2}$. The two edges associated to a certain coefficient represent isogenies with kernels order-2 subgroups not equal to $\langle (0,0) \rangle$.

As already noted in Remark 2, by using SIKE 2-isogenies, we cannot have $\langle (0,0) \rangle$ as kernel. Since these formulas fix the point $(0,0)$, it follows that a SIKE 2-isogeny $\phi : E_B \to E_{A'}$ never has a dual SIKE 2-isogeny $\widehat{\phi} : E_{A'} \to E_B$.

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 $E_{\pm A}$, [4], 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 Meet-in-the-Middle Attack on SIKE

In this section we will provide an overview of the meet-in-the-middle attack to solve the CSSI problem and optimizations specific to isogeny arithmetic used in SIKE.

A high level description is as follows. 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 looking for a non-trivial intersection: since isogenies are defined up to isomorphisms, we can identify the matching curve(s) *in-the-middle* by computing their j-invariants.

[4] 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_{-*}.

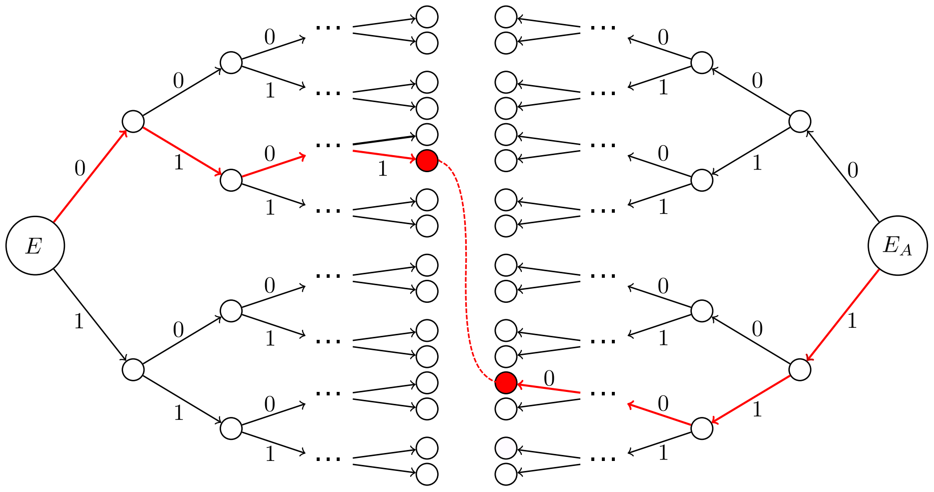


Fig. 2: Example of 2-isogeny trees starting from the two 2^e-isogenous curves E and E_A. Red nodes in the middle denote curves with same j-invariant, whose respective path in the tree (in red) connect E_A to E_B. Edge labels are assigned arbitrarily in order to identify the paths.

The correctness of this approach follows from the fact that the last $\lceil e/2 \rceil$ steps of the actual walk from E_A to E_B can be reversed due to existence of dual isogenies. Thus, the j-invariant in-the-middle is the one visited by the original walk after $\lfloor e/2 \rfloor$ steps from E_A.

In SIDH/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.

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 $\left\langle [\ell^{e-1}]Q \right\rangle$, leaving only ℓ^e isogenies in total. Furthermore, in Section 3, 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 finding appropriate torsion basis $\langle P', Q' \rangle = E_{A'}[\ell^e]$ with $[\ell^{e-1}]Q' = (0,0)$.

This refines the meet-in-the-middle 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. The more-detailed pseudocode is given in Algorithm 1.

Definition 2 (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 di-*

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).

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

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

rected graph consisting of all length-d walks from E arising from $[\ell^{e-d}] \cdot (P + [s]Q)$ with $s \in \mathbb{Z}_{\ell^e}$.

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, we can still obtain a tree by just distinguishing nodes arising from different paths, even if the respective image curves are the same. This approach will be of help when implementing our MitM attack.

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 the 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 1 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 $t \in [0, \ell^i - 1]$ will match the first i steps of length-e walks arising from $P + [s]Q$ with $s \in [0, \ell^e - 1]$. This can also be seen as an optimization, since computing ℓ^i-isogenies is much cheaper than computing the first i steps of ℓ^e-isogenies.

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 2 (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).

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's and Bob's walks in their respective torsions. To allow this, each party publishes the intermediate image curves[5] E_A and E_B. As was further noticed in [5], 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 the exact curve visited two steps before the end remains undetermined (i.e., the j-invariant is known but the A-coefficient is not). On the other hand, we can choose one of the 6 curves with the given j-invariant based on the following condition: the kernel $\langle (0,0) \rangle$ must define an isogeny towards the penultimate j-invariant on the original path. Then, such a curve would span a SIKE-tree including a walk through the same j-invariants as the original path (excluding the last two steps). This allows to represent the meet-in-the-middle problem for SIKE (with the 2-bit leak incorporated) in terms of intersecting two SIKE-trees, as described before.

Interestingly, by taking into consideration the walk structure induced by SIKE 2-isogenies, the relation between such a 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 distinct points of order 2 not equal to $(0,0)$. By applying the 2-isogeny formulas from Proposition 1 to the groups generated by K_0 and K_1, we obtain, respectively, two isogenies $\phi_1 : E_B \to E_A$ and $\phi_2 : E_B \to E_{A'}$ 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

[5] Even though the curves are not explicitly given, the torsion points needed by the other party are given, implicitly defining the concrete curve.

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 [9], into a sequence of two 2-isogenies followed by a sign flip of the pushed curve coefficient. It then suffices to flip the sign of the coefficient of the final curve to match the assumptions of Lemma 1, thus obtaining

$$A' = 2 + \frac{16}{A-2} \qquad \text{(one final 4-isogeny)} \tag{1}$$

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 [5], 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 considering the intermediate A-coefficients and j-invariants modulo the conjugation (e.g., using the norm or the trace suffices), this property effectively allows to halve the size of the initial SIKE-tree. Once an intersection between the SIKE-trees is found, it is left to check both conjugate candidates, which can be done easily due to the respective paths being element-wise conjugate. We refer to [5] for theoretical and graphical description of this phenomenon.

4 Efficient Tree Generation

In this section, we focus on optimized generation of the leaves of the tree spanned by given torsion generators. We focus on the case of 2-isogenies but the discussion can be generalized to other values of ℓ 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 (up to a precomputation of a fixed number of the first steps) as a *single step* in the low-memory van Oorschot-Wiener collision search applied to SIKE [1,5,11]. However, many intermediate curves will be visited multiple times for different values s in the kernel $\langle P + [s]Q\rangle$. To avoid the extra work, the authors of [1] developed a recursive method (called MITM-DFS) for computing the full tree in a depth-first traversal order, by efficiently maintaining a 2^{e-i}-torsion basis on each new visited curve at depth i.

In this section, we improve the MITM-DFS method in two ways. First, we show how to maintain the torsion basis in the case of SIKE isogenies, which allows to use the highly effective arithmetic of SIKE (including the available optimized implementation [22]). This requires careful choice of new generators so as to avoid the possibility of hitting the kernel $(0, 0)$. Second, we show how to adapt the strategic trade-off between isogeny evaluations and scalar multiplications described in SIDH [7] to the tree generation.

4.1 Maintaining Torsion Basis for Efficient Isogeny Computations

We now describe a method which allows 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. This extends a similar method for generic isogenies from [1]. The proof follows from the fact that the 2-isogeny formulas in SIKE (see Proposition 1) fix the point $(0, 0)$ (see Remark 2). The idea is to base the DFS tree exploration on the parity of the possible value s defining the kernel $\langle P_i + [s]Q_i\rangle$ at the depth i. Indeed, this parity defines the two possible 2-isogeny choices, and the following proposition shows how to compute the right torsion basis for the codomains of each of the two isogenies. This allows to recursively run the exploration in each of the curves. A pseudocode illustrating the use of this proposition is given in Algorithm 1.

Proposition 2. *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 $s \in [0, 2^e - 1]$, the pair $P', Q' \in E_{A'}(\mathbb{F}_{p^2})$ is a basis of $E_{A'}[2^{e-1}]$ with $[2^{e-2}]Q' = (0, 0)$, where*

$$P' = \phi(P), \qquad Q' = \phi([2]Q), \quad \text{if } \ker\phi = \left\langle [2^{e-1}]P \right\rangle \ \text{(i.e., } s \text{ is even)};$$
$$P' = \phi(P + Q), \quad Q' = \phi([2]P), \quad \text{if } \ker\phi = \left\langle [2^{e-1}](P + Q) \right\rangle \ \text{(i.e., } s \text{ is odd)}.$$

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 $\ker\phi = \left\langle [2^{e-1}]P \right\rangle$ 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 $\ker\phi = \left\langle [2^{e-1}](P + Q) \right\rangle$, 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 $\ker\phi = \left\langle [2^{e-1}]P \right\rangle$, 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 $\ker\phi = \left\langle [2^{e-1}](P + Q) \right\rangle$.

□

4.2 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 $e_0 < e$, and later push all such points through isogenies, to aid scalar multiplications on the subsequent curves arising in the walk. Indeed, this allows to compute the kernel of the next-step ℓ-isogeny with just $e-1-e_0$ point multiplications by ℓ using the maximum e_0 for which $[\ell^{e_0-1}](P+[s]Q)$ is stored, while storing and pushing smaller multiples is also useful for the 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 [7], 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 a dynamic programming-based algorithm. We now provide a brief overview of how optimal evaluation strategies are found in [7]. 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

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

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 $i+j+1 \le e-1$;
– "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 $i+j+1 \le e-1$.

A strategy for evaluating all the kernels $\ker \phi_i = \langle [\ell^{e-1-i}]K_i \rangle$ 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 [7], it was shown that there exist minimal-cost strategies with recursive structures. The problem is decomposed into two subproblems, where the costs of the subgraph induced after following i multiplication edges (height $e-1-i$), and the subgraph induced after following $e-i$ isogeny evaluation edges (height

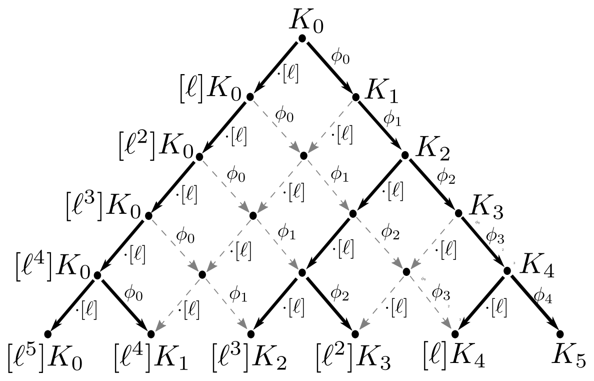


Fig. 3: An example of evaluation strategy graph. Multiplication by $[\ell]$ edges ($\swarrow$) and isogeny evaluation edges ($\searrow$) transform $K_0 \in E[\ell^6]$ to the leaf values $\{[\ell^{6-i-1}]K_i\}_{i\in[0,5]}$. In bold, an optimal evaluation strategy assuming $C_{\text{eval}} = 1.5 \cdot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

$i-1$) are taken into account. For $e \geq 2$, the minimal cost C_e for evaluating trees of height $e-1$ is given by

$$C_e = \min_{1\leq i\leq e-1}(i \cdot C_{\text{mult}} + (e-i)\cdot C_{\text{eval}} + C_{e-i} + C_i)$$

This is possible due to the fact that, in paths towards leaves, the order of any two consecutive 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 i 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 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 and edges being explored in the isogeny tree by ℓ (including the isogeny edge itself). To account for 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. On the other hand, all weights at isogeny depth i are simply multiplied by a factor ℓ^i. Therefore, it is sufficient to find best strategies for trees of heights $1, \ldots, e-1$ rooted at K_0. This leads to a simple 1-dimensional dynamic programming algorithm with complexity $\mathcal{O}(e^2)$, based on the new recursive expression:

$$C_e = \min_{1 \le i \le e-1} \left(\underbrace{i \cdot C_{\text{mult}}}_{i \text{ mult. edges}} + \underbrace{\left(\sum_{j=1}^{e-i} \ell^j \right) \cdot C_{\text{eval}}}_{e-i \text{ isog. edges}} + \underbrace{C_{e-i}}_{\text{left subtree}} + \underbrace{\ell^{e-i} \cdot C_i}_{\text{right subtree}} \right)$$

5 Set Intersection using Sort and Merge

5.1 Hash-tables or Sort and Merge?

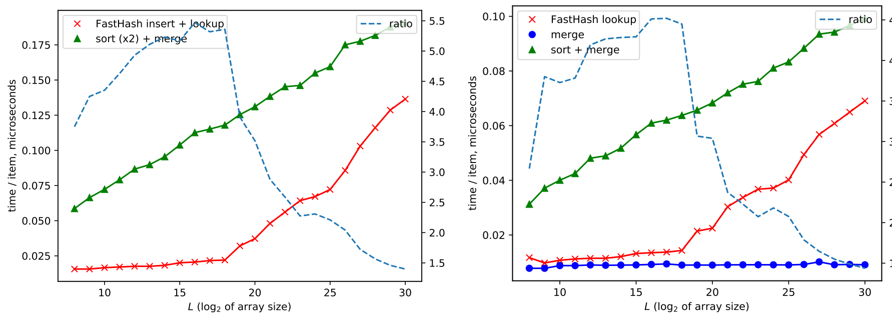
A standard way to implement the final stage of a meet-in-the-middle attack, i.e., intersecting the two sets of values in-the-middle, is by using hash-tables: we fill one of such tables with entries from the first dataset, and we then lookup every element in the second one. In theory, the amortized cost of a 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 affecting the 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 (in theory) the sorting step has quasilinear complexity $\mathcal{O}(n \log n)$ in the (biggest) dataset size n, and to complete it we need memory/storage 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



(a) Intersecting two arrays without precomputation (b) Lookup in an array with precomputation

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

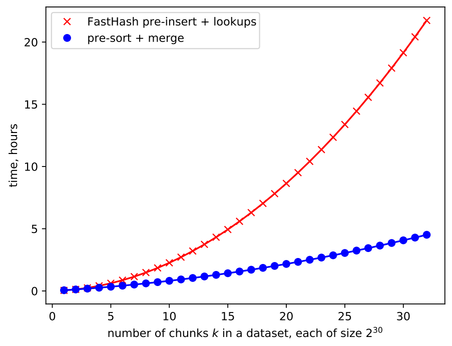


Fig. 5: Performance comparison between FastHash and SortMerge running on parallel over k 64-bit integers chunks, each of size 2^{30} (extrapolated using timings from Figure 4b).

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.

Sorting Big-data. Sorting large amounts of data that cannot fit the main memory is known as *external sorting*. A well-known approach for external sorting is a hybrid sort-and-merge. First, the data is split into relatively small chunks that fit memory of the used machine and that are sorted in parallel using any standard algorithm (e.g., radix sort). Sorted chunks are written to the disk-based storage. The second stage is merging the sorted chunks into bigger sorted chunks. If the number of initial chunks is too large, this process can be performed in several layers, each merging every t sorted chunks into one bigger sorted chunk. This requires (parallel) sequential reading of the t chunks and a size-t heap (which exhibits random memory accesses but in a small memory range). For the purpose of set intersection, the last layer may merge chunks from both sets and compare elements on the fly, removing the need to write the full sorted dataset (which requires costly I/O operations).

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 chunk pairs intersection takes preprocessed data as input. In our in-RAM experiments (see Subsection 5.1) we observed that already for $k = 2$, SortMerge (which requires a total number of 2 sort calls and k merges/intersections per chunk) outperforms the FastHash hash-set approach (which requires 1 chunk insertion and k chunks lookups per chunk).

5.2 Storage-Collisions Trade-off and Compression

The large problem scale requires to reduce storage requirements as much as possible. We describe three techniques for this purpose.

Omitting path information. Basic MitM would store the paths associated to j-invariants in each tree in order to quickly reconstruct the isogeny path. This information can be omitted at the cost of an extra iteration of tree exploration, required to recover full j-invariants associated to colliding representations and the respective paths in the trees. This extra exploration can be considered memoryless, if the expected number of collisions is sufficiently small (for example, if it fits the local memory of a computing node).

Truncating j-invariants representations. Storage reduction can be made by reducing the number of bits we use to represent j-invariants (modulo the conjugation), while allowing only a reasonable amount of false positive collisions. Since each tree has only approximately $2^{e_A/2} \approx p^{1/4}$ leaves, using n bits to represent j-invariants leads to approximately $c = (p^{1/4})^2/2^n$ collisions.

Sorted set compression. If the n-bit j-invariant representations are uniformly distributed, we can compress *sorted* chunks of 2^m such elements by noticing that any two consecutive elements are expected to differ, on average, by 2^{n-m} (as integers). We can then store only such reduced differences, reserving 1 *flag* bit for distinguishing a small difference from a full n-bit representation (in case two elements differ by more than 2^{n-m}). This, in fact, reduces memory requirements from $n2^m$ bits to $\approx (n-m+1)2^m$ bits, with different implementation-specific word size trade-offs in the middle.

We note that by requiring the chunks to be sorted, this compression technique goes towards the SortMerge intersection finding approach we detailed in Section 5. Since the chunks can be decompressed on the fly without any overhead, the merge steps can be performed as usual.

6 Cryptanalysis of the $IKEp182 Challenge

In this section we will detail how all the above ideas can be used to concretely break the $IKEp182 challenge [12], 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 specification, we have $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2+1)$, $\#E_6(\mathbb{F}_{p^2}) = (p+1)^2$ and $E_6[2^{91}] = \langle P_A, Q_A \rangle$, $E_6[3^{57}] = \langle P_B, Q_B \rangle$. The coordinates of the points P_A, Q_A, P_B, Q_B as well as all other values related to the attack in this section are reported in Appendix A due to page limitations.

Our meet-in-the-middle attack targets the 2^{91}-torsion in order to recover Alice's full 91-steps walk, followed by the private key recovery. After a quick Setup, the full attack consists of 5 main stages: Trees Traversal, k-way Merge, Compression, Sieving and Final Trees Traversal.

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 3 x-coordinates $x_{\phi_A(P_B)}, x_{\phi_A(Q_B)}, x_{\phi_A(Q_B)-\phi_A(P_B)}$. Such compressed representation is justified by use of efficient implementations which exploits x-only arithmetic: we refer to [9] 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 [4, 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

$$A = \frac{(1 - x_P x_Q - x_P x_{Q-P} - x_Q x_{Q-P})^2}{4 x_P x_Q x_{Q-P}} - x_P - x_Q - x_{Q-P}$$

To take advantage of the final 2-bit leak described in Section 3, 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 (1) to obtain the coefficient $A' = 2 + 16/(A-2)$.

The Setup phase was implemented in SageMath [21].

Trees Traversal. We proceed by attacking the 89-steps path in the 2-isogeny graph between $j(E_6)$ and $j(E_{A'})$. Note that there may be no path in the SIKE-tree (Definition 2) between the exact *curves*, as we might chose a different representative curve, but there must exist a path in the 2-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 arising from E_6 (up to the depth 45) and the SIKE-tree arising from $E_{A'}$ (up to the depth 44), employing the optimal strategies detailed in Subsection 4.2.

We note that, as discussed in Section 3, 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 half of its trace. In our implementation, multiple jobs explore in parallel distinct branches of each tree: when a job collects 2 GiB 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 cluster we used, each of these job took approximately 17 minutes to complete on a single core of an Intel® Xeon® E5-2680v4 clocked at 2.4GHz with 4 GiB of RAM reserved. This sums up to a total of approximately 4.2 core-years and 256 TiB 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 128 TiB.

***k*-way Merge.** We employed our custom *k-way merge* implementation optimized for 64-bit unsigned integers, to merge the 2 GiB sorted chunks generated from each tree: on a single core with 4 GiB of RAM, we needed approximately 2.5 core hours to merge 256 2 GiB chunks into a single 512 GiB sorted chunk. We note that, to keep memory requirements close to the ones needed to store all j-invariants representations, chunks can be merged at the same time with the depth-first traversal, as soon as enough new 2 GiB 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 our 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 512 GiB 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 5.2), 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 compressed chunks of 1 TiB each from each tree, for a total of 128 TiB 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 Subsection 5.1. This stage consists in merging tuples of (compressed) 1 TiB 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 128 TiB 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 2 PiB of data read, which is acceptable to allow sufficient parallelization.

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

Final Trees Traversal. 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 half of their trace matching any of the collisions found, and their paths in the respective trees.

After the full collision is found, 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, including the check for the conjugate path arising from E_6 (see Section 3). In our case, the colliding j-invariants in-the-middle obtained by expanding the trees from E_6 and $E_{A'}$ were, indeed, conjugate pairs. The respective values, j_0 and j_1, are reported in Appendix A.

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 allows 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 [12].

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 $\approx 28\%$). We remark that these percentages only correspond to the final tree regeneration step, the previous stages were fully completed.

This brings the total cost of our attack to approximately $4.2 + (54 + 27 + 280)/365 + 4.2 \cdot (0.63 + 0.44)/2 \leq 9.5$ core years and 256 TiB of disk memory.

We note, however, that we only decided to employ compression after the Trees Traversal phase was completed, 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 9.5 core years with just slightly more than 128 TiB of disk memory available. The storage requirement can be reduced further by sacrificing parallelization and performing the main steps for a single group of the second tree at a time. In our case, we used 4-chunk groups (4 TiB) on each side and so only $64 + 4 = 68$ TiB of storage is sufficient for the (less-parallel) attack.

7 Discussion on Scalability

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. More generally, on instances up to which size can the meet-in-the-middle attack compete with the van Oorschot-Wiener method?

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

The computational cost should scale linearly with the sizes of the trees, resulting in $2^{(110-91)/2} \cdot 9.5 \approx 6900$ core-years. We remark though, that, on the cluster we used, the main limitation is the I/O performance upper bounded by about 20 GiB/s. Even if an unlimited storage 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 192 PiB of data on such a cluster, one would need at least 116 days. Since the full attack performs several I/O rounds, the attack would likely take more than a year.

Towards the other side of the memory-time trade-off, we could reuse the current attack setup on \$IKEp182 with $\tilde{e}_A = 91$, by guessing $e_A - \tilde{e}_A = 19$ final steps on the path, leading to the estimation of $2^{19} \cdot 9.5 \approx 5$ million core-years for computations. This is a very "clean" upper-bound estimation in that it is based on a real experiment and it parallelizes perfectly (with the number of involved clusters similar to the one we used).

We can conclude that, depending on the physical feasibility of high-speed access to 192 PiB of storage, our attack on \$IKEp217 may take between 6.9k and 5M core-years. We remark that we did not take into account possibilities of further optimization of the implementation or, more low-level implementations (GPU/FPGA/ASIC).

(SIKEp377) SIKEp377 is the smallest instance proposed in [11] based on detailed hardware-cost analysis of matching the NIST Security Level 1 (roughly equal to security of the block cipher AES). The respective prime p is equal to $2^{191}3^{117}-1$.

As it is already unrealistic to consider $2^{(e_A-3)/2} = 2^{94}$ units of storage, we resort to the approach of [1,5] of bounding the memory units by 2^{80}, in order to obtain our attacker-optimistic estimation. Here, a unit may be, for example, a 128-bit integer (or a 64-bit integer after the difference-based compression). Then, after guessing 28 2-isogeny steps, the adversary would run the MitM attack on SIKE-trees of size $2^{(e_A-28-3)/2} = 2^{80}$ (we assume that the second tree is checked on-the-fly in chunks of negligible size; for example, using $2^{80}+2^{77} = 2^{80.17}$ units results in slowdown of 2^3). The basic meet-in-the-middle analysis predicts the cost of $2^{28} \cdot 2 \cdot 2^{80} = 2^{119}$ tree-element (i.e., j-invariants) generations. On the other hand, for the storage of 2^{80} units, a realistic implementation of the sort-and-merge approach (repeated 2^{29} times) would clearly blow up the complexity beyond 2^{128} operations or even AES encryptions.

Similarly to $IKEp217, we could also reuse the attack on $IKEp182 to get an estimate for SIKEp377. Here, the multiplicative complexity factor is 2^{100}. In order to provide a comparison with the financial cost estimation given in [11] (based on the hardware implementation of the vOW method), we (optimistically) assume that our attack can be reproduced in 1 day on a device costing $1000. Therefore, with a $1 billion budget, we could use 1M such devices in parallel, leading to an estimate of 2^{71} years, compared to about 2^{40} years given in [11]. Even with an unlimited budget, we could use about 2^{35} such devices to fit the 2^{80} memory limit, leading to an estimate of $2^{100-35}/365 = 2^{56}$ years.

Conclusions. As we could see, the advantage of the Meet-in-the-middle attack over the van-Oorschot-Wiener method against SIKE decreases with the growth of the involved prime p. However, precise comparison of two methods is complicated by unclear physical limits of the set intersection problem. The estimation based on mesh sorting in [1] is too pessimistic at least for the toy instances $IKEp182/$IKEp217, where the required amount of memory is manageable and physical limitations do not yet have an effect. Our implementation of the attack on $IKEp182 is relatively straightforward and only uses an existing computational architecture. According to our analysis, it has high potential to be applied to $IKEp217 with sufficient amount of resources. However, already for the smallest non-toy instance SIKEp377, our implementation does not allow to straightforwardly beat the vOW-based estimation of [11] and it does not seem to threaten the claimed 128-bit security. On the other hand, we could not discard the possibility that a well-thought hardware-based architecture for the MitM attack could still compete with the vOW method on SIKEp377.

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[6] for sparkling our interest in the topic.

References

1. Adj, G., Cervantes-Vázquez, D., Chi-Domínguez, J.J., Menezes, A., Rodríguez-Henríquez, F.: On the cost of computing isogenies between supersingular elliptic curves. In: Cid, C., Jacobson Jr:, M.J. (eds.) SAC 2018. LNCS, vol. 11349, pp. 322–343. Springer, Heidelberg (2019) 2, 3, 4, 12, 13, 23
2. Bernstein, D.: Cost analysis of hash collisions : will quantum computers make sharcs obsolete? In: SHARCS'09 Workshop Record (Proceedings 4th Workshop on Special-purpose Hardware for Attacking Cryptograhic Systems, Lausanne, Switserland, September 9-10, 2009). pp. 105–116 (2009) 4
3. Costello, C., Hisil, H.: A simple and compact algorithm for SIDH with arbitrary degree isogenies. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part II. LNCS, vol. 10625, pp. 303–329. Springer, Heidelberg (2017) 7
4. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 572–601. Springer, Heidelberg (2016) 19
5. Costello, C., Longa, P., Naehrig, M., Renes, J., Virdia, F.: Improved classical cryptanalysis of SIKE in practice. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 505–534. Springer, Heidelberg (2020) 2, 3, 11, 12, 23
6. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006) 2
7. Feo, L.D., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Journal of Mathematical Cryptology **8**(3), 209–247 (2014) 2, 3, 6, 13, 14
8. Galbraith, S.D.: Constructing isogenies between elliptic curves over finite fields. LMS Journal of Computation and Mathematics **2**, 118–138 (1999) 2
9. Jao, D., Azarderakhsh, R., Campagna, M., Costello, C., Feo, L., Hess, B., Jalali, A., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Pereira, G., Renes, J., Soukharev, V., Urbanik, D.: Supersingular Isogeny Key Encapsulation. https://sike.org/files/SIDH-spec.pdf (October 2020), NIST Post-Quantum Cryptography Round 3 - Alternate Candidate 1, 6, 12, 19
10. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.Y. (ed.) Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011. pp. 19–34. Springer, Heidelberg (2011) 2, 6
11. Longa, P., Wang, W., Szefer, J.: The cost to break SIKE: A comparative hardware-based analysis with AES and SHA-3. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 402–431. Springer, Heidelberg, Virtual Event (2021) 2, 12, 23

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

12. Microsoft Research: SIKE Cryptographic Challenge (2021), https://www.microsoft.com/en-us/msrc/sike-cryptographic-challenge 2, 19, 22
13. 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
14. Renes, J.: Computing isogenies between Montgomery curves using the action of (0, 0). In: Lange, T., Steinwandt, R. (eds.) Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018. pp. 229–247. Springer, Heidelberg (2018) 7
15. Rostovtsev, A., Stolbunov, A.: Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145 (2006) 2
16. Schnorr, C.P., Shamir, A.: An optimal sorting algorithm for mesh connected computers. In: Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing. p. 255–263. STOC '86, Association for Computing Machinery, New York, NY, USA (1986) 4
17. Schoof, R.: Nonsingular plane cubic curves over finite fields. Journal of Combinatorial Theory, Series A **46**(2), 183–211 (1987), https://www.sciencedirect.com/science/article/pii/0097316587900033 5
18. Silverman, J.: The Arithmetic of Elliptic Curves, vol. 106 (January 2009) 5
19. Stolbunov, A.: 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) 2
20. Tate, J.: Endomorphisms of abelian varieties over finite fields. Inventiones mathematicae **2**(2), 134–144 (1966) 4
21. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 9.4) (2021), https://www.sagemath.org 2, 20
22. The SIDH Library Developers: SIDH v3.4 (C Edition) (2021), https://github.com/microsoft/PQCrypto-SIDH 2, 7, 13
23. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. Journal of Cryptology **12**(1), 1–28 (1999) 2
24. Varrette, S., Bouvry, P., Cartiaux, H., Georgatos, F.: Management of an academic HPC cluster: The UL experience. In: Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014). pp. 959–967. IEEE, Bologna, Italy (July 2014) 1

A Intermediate Values from the Attack on $IKEp182

In Figure 6, we report the concrete values of the relevant quantities computed while performing the attack outlined in Section 6.

$$
\begin{aligned}
P_A &= (\texttt{0x05a324935a4d7b75024fdc3601fe8b5888cea9f88212b2 +}\\
&\quad\ \texttt{0x02357bdd576772bf2a93e3d680ed7306e16eafc6aff904}\cdot\texttt{i},\\
&\quad\ \texttt{0x242a9e09aa8e6995e4fdce9f68e8c2c902154c332de68a +}\\
&\quad\ \texttt{0x011b23646f8884b7a9faa5159ef13842880ed0f9f43dcd}\cdot\texttt{i})\\
Q_A &= (\texttt{0x27b8def415bae0506a9607fff7704832151cdcbc93cb22 +}\\
&\quad\ \texttt{0x085c86f386b94b8c413f5e49736f26de95103a9b65f31a}\cdot\texttt{i},\\
&\quad\ \texttt{0x16af6790fb0f5cfd0e124033bb7619e2f75a25cae5f42b +}\\
&\quad\ \texttt{0x172567b99058dd9d5b99ce5ea4bacd685f57c8326011a3}\cdot\texttt{i})\\
P_B &= (\texttt{0x02ca3bc7e98f88b3ca3239c276eb7a224c51f61bc8c5ed},\\
&\quad\ \texttt{0x262a38701d1b61dd8875909ff268a50d912f620db980a1})\\
Q_B &= (\texttt{0x02dcff7123e2380f552f5bff91da77ae62e9556b866d8f},\\
&\quad\ \texttt{0x06aeb7c764aa40913b3fc784d569833d4226cc4a53432f}\cdot\texttt{i})\\
x_{\phi_A(P_B)} &= \texttt{0x17d02d323c815eee1ec75f1c675609b0bea78064cb8cc1 +}\\
&\quad\ \texttt{0x12fa80de8027f68c3f780b5bcd519e8205606ac249025d}\cdot\texttt{i}\\
x_{\phi_A(Q_B)} &= \texttt{0x272c54d49af950b0829072753e3525091aaf87085bd7b2 +}\\
&\quad\ \texttt{0x23efe3c087965a49fcc5161e6453dbe632d7dec90bab12}\cdot\texttt{i}\\
x_{\phi_A(Q_B)-\phi_A(P_B)} &= \texttt{0x22c38abb1427245de1e049408dab87ed9ba54efeb4a4e4 +}\\
&\quad\ \texttt{0x0c5d768e87a762b6a460b941bcc5537ba0f73ce8b9f955}\cdot\texttt{i}\\
A &= \texttt{0xc0cbda5ef968048cd2c1b125774f1417125b9b02b6f91 +}\\
&\quad\ \texttt{0x1e8121a2a60fd266d321bb9db8d9e3111e3095c08e0bc6}\cdot\texttt{i}\\
A' &= \texttt{0x164db610b03a9b3c38e59bf29485a60462d1cd9f22d95e +}\\
&\quad\ \texttt{0x1a8d75d6d0285807042e900df3c2cf74b4eb160d50a92e}\cdot\texttt{i}\\
j(E_{A'}) &= \texttt{0xe48a8271ea06ec4193db09970a23bea55c777ef2fb5be +}\\
&\quad\ \texttt{0x56910191b4835901ef45e4b857817391ad1213080afa9}\cdot\texttt{i}\\
j_0 &= \texttt{0x0008132653e4d53cb9cc0defb36a0141d900adbb128a24f0 +}\\
&\quad\ \texttt{0x0001049f06c78aaed22786dfcff5b202ce3a50429f369b86}\cdot\texttt{i}\\
j_1 &= \texttt{0x0008132653e4d53cb9cc0defb36a0141d900adbb128a24f0 +}\\
&\quad\ \texttt{0x0027910d1a0d795d077f40d1480a4dfd31c5afbd60c96479}\cdot\texttt{i}
\end{aligned}
$$

Fig. 6: Concrete coefficients recovered in the attack of $IKEp182