Revisiting Meet-in-the-Middle Cryptanalysis of SIDH/SIKE with Application to the $IKEp182 Challenge

Aleksei Udovenko[1,2], Giuseppe Vitto[2]

[1]CryptoExperts
[2]SnT, University of Luxembourg

Selected Areas in Cryptography 2022
25th August 2022

UNIVERSITÉ DU LUXEMBOURG

securityandtrust.lu

Luxembourg National Research Fund

CRYPTO EXPERTS

High-level Overview

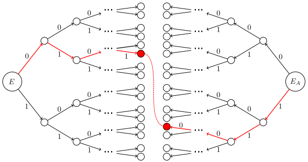
- SIDH/SIKE are isogeny-based PQ protocols
- Rely on hardness of finding *isogenies* between elliptic curves
- (Previously) Best attacks: generic claw finding (meet-in-the-middle)
- Physical memory constraints (size x speed)
  $\Rightarrow$ low-memory van Oorschot-Wiener (vOW)

$$
\begin{tikzcd}
E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle A \rangle \arrow[d, "\phi'_B"] \\
E/\langle B \rangle \arrow[r, "\phi'_A"'] & E/\langle A, B \rangle
\end{tikzcd}
$$

High-level Overview

$$\begin{tikzcd} E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle A \rangle \arrow[d, "\phi'_B"] \\ E/\langle B \rangle \arrow[r, "\phi'_A"'] & E/\langle A, B \rangle \end{tikzcd}$$

- SIDH/SIKE are isogeny-based PQ protocols
- Rely on hardness of finding *isogenies* between elliptic curves
- (Previously) Best attacks: generic claw finding (meet-in-the-middle)
- Physical memory constraints (size x speed)
  $\Rightarrow$ low-memory van Oorschot-Wiener (vOW)

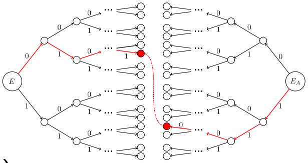


- **this work**: revisiting and optimizing the MitM approach

High-level Overview

$$\begin{tikzcd} E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle A\rangle \arrow[d, "\phi'_B"] \\ E/\langle B\rangle \arrow[r, "\phi'_A"'] & E/\langle A, B\rangle \end{tikzcd}$$

- SIDH/SIKE are isogeny-based PQ protocols
- Rely on hardness of finding *isogenies* between elliptic curves
- (Previously) Best attacks: generic claw finding (meet-in-the-middle)
- Physical memory constraints (size x speed)
  $\Rightarrow$ low-memory van Oorschot-Wiener (vOW)



- **this work**: revisiting and optimizing the MitM approach
- Proof-of-concept: breaking \$IKEp182 challenge (by Microsoft) ~~on a laptop on a weekend~~ on an HPC cluster in a week (9 core-years)

Comparison to the Castryck-Decru Attack (Castryck and Decru 2022)

Outdated? Is MitM useless?

Comparison to the Castryck-Decru Attack (Castryck and Decru 2022)

Outdated? Is MitM useless?

- Castryck-Decru attack relies on the torsion point images
- MitM is more generally applicable (existing + future schemes)
- Generic attack in the generic setting may still be relevant for security analysis

Plan

SIDH/SIKE

- Public parameters:
  1. prime p with $p+1 = 2^{e_A}3^{e_B}$
  2. starting curve E with 2^{e_A}- and 3^{e_B}-torsion bases

$$\begin{tikzcd}
E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle A \rangle \arrow[d, "\phi'_B"] \\
E/\langle B \rangle \arrow[r, "\phi'_A"'] & E/\langle A, B \rangle
\end{tikzcd}$$

Figure credits: TikZ for Cryptographers

SIDH/SIKE

- Public parameters:
  1. prime p with $p + 1 = 2^{e_A}3^{e_B}$
  2. starting curve E with 2^{e_A}- and 3^{e_B}-torsion bases
- Alice:
  1. computes a secret 2^{e_A} isogeny $\phi_A : E \to E/\langle A \rangle$
  2. publishes E_A with the 3^{e_B}-torsion image on it

$$
\begin{tikzcd}
E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle A \rangle \arrow[d, "\phi'_B"] \\
E/\langle B \rangle \arrow[r, "\phi'_A"'] & E/\langle A, B \rangle
\end{tikzcd}
$$

Figure credits: TikZ for Cryptographers

SIDH/SIKE

- Public parameters:
  1. prime p with $p + 1 = 2^{e_A} 3^{e_B}$
  2. starting curve E with 2^{e_A}- and 3^{e_B}-torsion bases
- Alice:
  1. computes a secret 2^{e_A} isogeny $\phi_A : E \to E/\langle A \rangle$
  2. publishes E_A with the 3^{e_B}-torsion image on it
- Bob:
  1. computes a secret 3^{e_B} isogeny $\phi_B : E \to E/\langle B \rangle$
  2. publishes E_B with the 2^{e_A}-torsion image on it

$$
\begin{tikzcd}
E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle A \rangle \arrow[d, "\phi'_B"] \\
E/\langle B \rangle \arrow[r, "\phi'_A"'] & E/\langle A, B \rangle
\end{tikzcd}
$$

Figure credits: TikZ for Cryptographers

SIDH/SIKE

- Public parameters:
  1. prime p with $p + 1 = 2^{e_A}3^{e_B}$
  2. starting curve E with 2^{e_A}- and 3^{e_B}-torsion bases
- Alice:
  1. computes a secret 2^{e_A} isogeny $\phi_A : E \to E/\langle A\rangle$
  2. publishes E_A with the 3^{e_B}-torsion image on it
- Bob:
  1. computes a secret 3^{e_B} isogeny $\phi_B : E \to E/\langle B\rangle$
  2. publishes E_B with the 2^{e_A}-torsion image on it

$$\begin{tikzcd} E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle A\rangle \arrow[d, "\phi'_B"] \\ E/\langle B\rangle \arrow[r, "\phi'_A"'] & E/\langle A, B\rangle \end{tikzcd}$$

- Alice and Bob then reapply their secret isogenies on the published curves and arrive at the same shared secret $E/\langle A, B\rangle$

Figure credits: TikZ for Cryptographers

SIDH/SIKE

- Public parameters:
  1. prime p with $p + 1 = 2^{e_A} 3^{e_B}$
  2. starting curve E with 2^{e_A}- and 3^{e_B}-torsion bases
- Alice:
  1. computes a secret 2^{e_A} isogeny $\phi_A : E \to E/\langle A\rangle$
  2. publishes E_A with the 3^{e_B}-torsion image on it
- Bob:
  1. computes a secret 3^{e_B} isogeny $\phi_B : E \to E/\langle B\rangle$
  2. publishes E_B with the 2^{e_A}-torsion image on it
- Alice and Bob then reapply their secret isogenies on the published curves and arrive at the same shared secret $E/\langle A, B\rangle$

$$\begin{tikzcd}
E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle A\rangle \arrow[d, "\phi'_B"] \\
E/\langle B\rangle \arrow[r, "\phi'_A"'] & E/\langle A, B\rangle
\end{tikzcd}$$

This work: recovering the 2^{e_A}-isogeny $\phi_A : E \to E/\langle A\rangle$, given only E and $E_A = E/\langle A\rangle$

SIDH/SIKE Arithmetic

- Montgomery curves: $E_A : y^2 = x^3 + Ax^2 + x$ defined over $\mathbb{F}_{p^2}$
- Efficient x-only arithmetic

SIDH/SIKE Arithmetic

- Montgomery curves: $E_A : y^2 = x^3 + Ax^2 + x$ defined over $\mathbb{F}_{p^2}$
- Efficient x-only arithmetic
- A 2^{e_A}-isogeny decomposes into e_A 2-isogenies $\phi_i : E_{A_{i-1}} \to E_{A_i}$:

$$\phi_{\mathsf{Alice}} = \phi_{e_A} \circ \ldots \circ \phi_1$$

SIDH/SIKE Arithmetic

- Montgomery curves: $E_A : y^2 = x^3 + Ax^2 + x$ defined over $\mathbb{F}_{p^2}$
- Efficient x-only arithmetic
- A 2^{e_A}-isogeny decomposes into e_A 2-isogenies $\phi_i : E_{A_{i-1}} \to E_{A_i}$:

$$\phi_{\mathsf{Alice}} = \phi_{e_A} \circ \ldots \circ \phi_1$$

- 2-isogeny $\phi_i : E_{A_{i-1}} \to E_{A_i}$:
  - requires the x-coordinate k of an order-2 point on E (the kernel gen.), $x \neq 0$
  - evaluate: $\phi_i(x) = \frac{x(kx-1)}{x-k}$
  - next curve: $A_i = 2 - 4k^2$

SIDH/SIKE Arithmetic

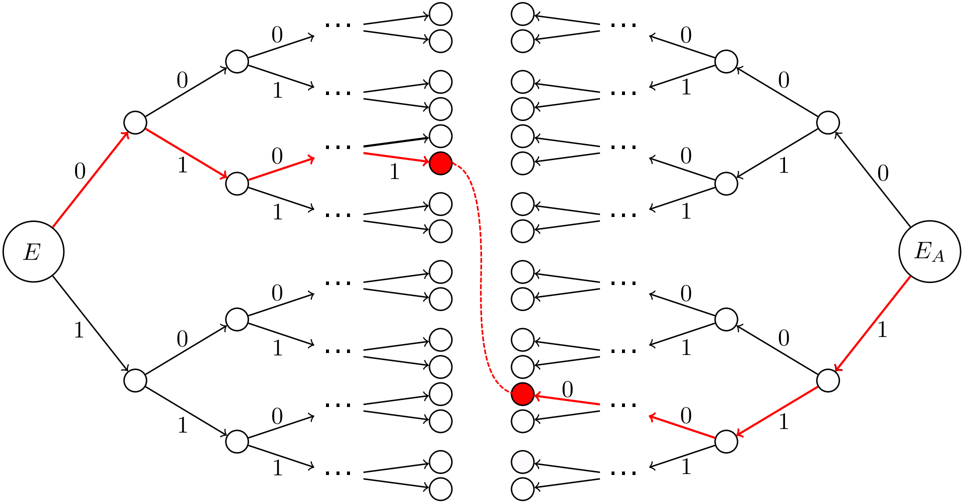
- Montgomery curves: $E_A : y^2 = x^3 + Ax^2 + x$ defined over $\mathbb{F}_{p^2}$
- Efficient x-only arithmetic
- A 2^{e_A}-isogeny decomposes into e_A 2-isogenies $\phi_i : E_{A_{i-1}} \to E_{A_i}$:

$$\phi_{\text{Alice}} = \phi_{e_A} \circ \ldots \circ \phi_1$$

- 2-isogeny $\phi_i : E_{A_{i-1}} \to E_{A_i}$:
  - requires the x-coordinate k of an order-2 point on E (the kernel gen.), $x \neq 0$
  - evaluate: $\phi_i(x) = \frac{x(kx-1)}{x-k}$
  - next curve: $A_i = 2 - 4k^2$
- The 2-kernels can be derived from the 2^{e_A}-kernel of ϕ_{Alice} by pushing through ϕ_i and raising to appropriate power $[2^{e_A-1-i}]$

Plan

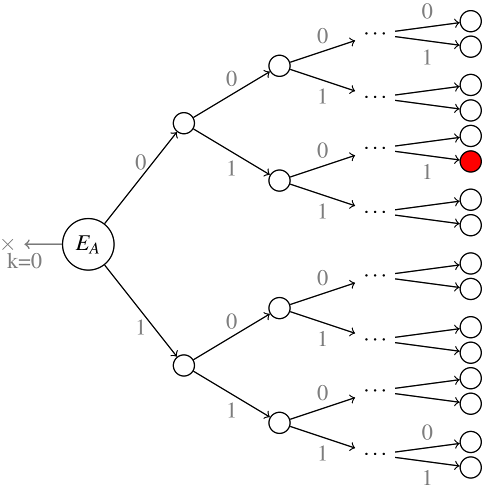
High-level MitM (Galbraith 1999; Adj et al. 2019)



SIKE: Left Tree

- Goal:

$$\text{LeftTree} = \left\{ j(E_A/\langle P + [s]Q \rangle) \mid s \in [0, 2^{e_A/2}] \right\}$$



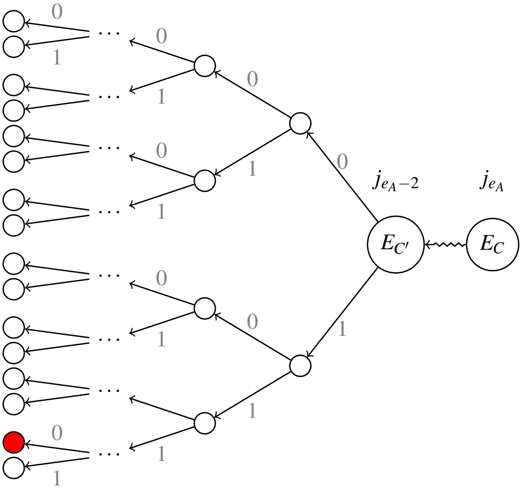
SIKE: Right Tree

- Optimized arithmetic formulas leak 2 last steps (Costello et al. 2020)
- We express the right tree in the same shape as the left tree
- Let

$$C' = 2\frac{C-6}{C+2}$$

- Goal:

$$\mathrm{RightTree} = \left\{ j(E_{C'}/\langle P' + [s]Q' \rangle) \mid s \in [0, 2^{e_A/2}] \right\}$$



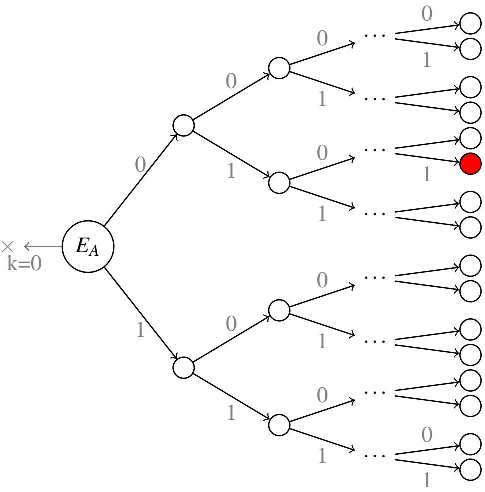
Plan

Recursive Generation (MitM-DFS (Adj et al. 2019))

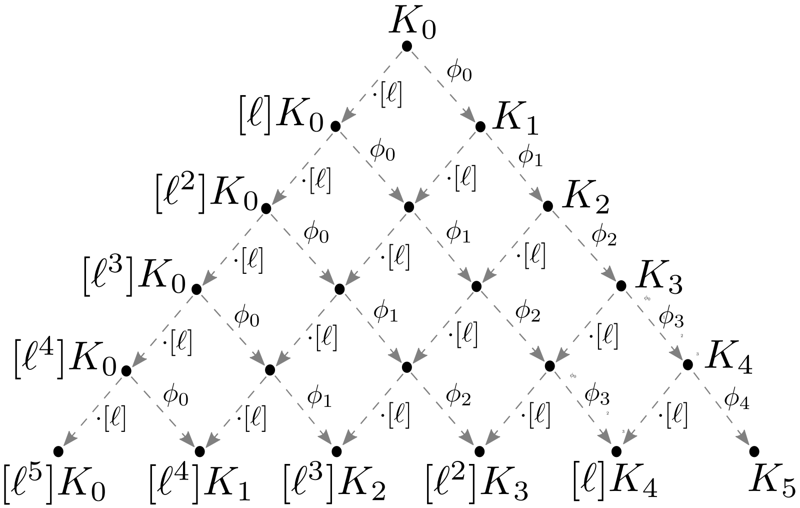
- Goal:

$$\text{Tree} = \left\{ j(E/\langle P + [s]Q\rangle) \mid s \in [0, 2^{e_A/2}] \right\}$$

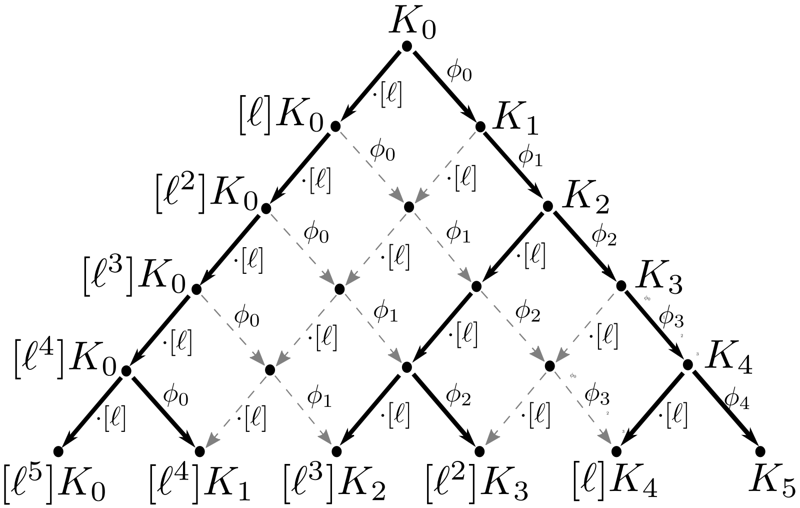
- DFS visit tree (recursively)
  - need 2-kernel points
  - maintain $2^{e_A - i}$-torsion, by pushing it through isogenies and updating accordingly
- We adapt the idea to SIKE's optimized arithmetic (e.g., ensure $x \neq 0$)
- We adapt optimal strategy for trade-off between doublings and isogeny evaluations



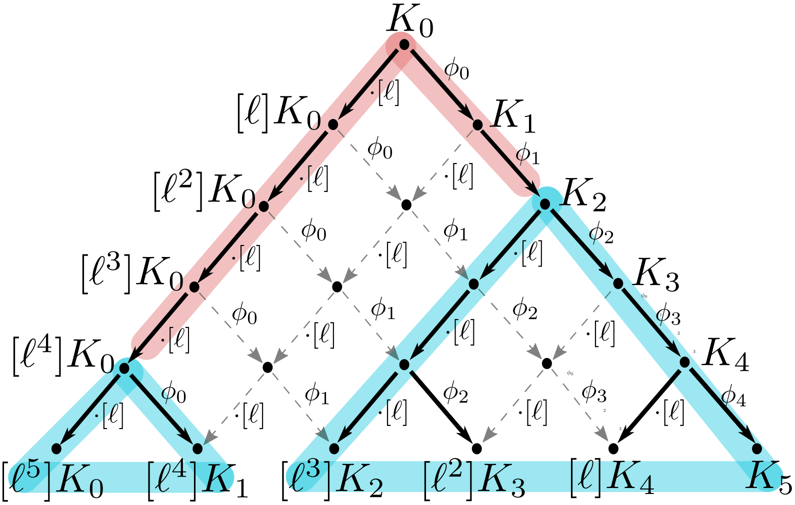
Optimal Strategy for Doubling-Isogeny Trade-off



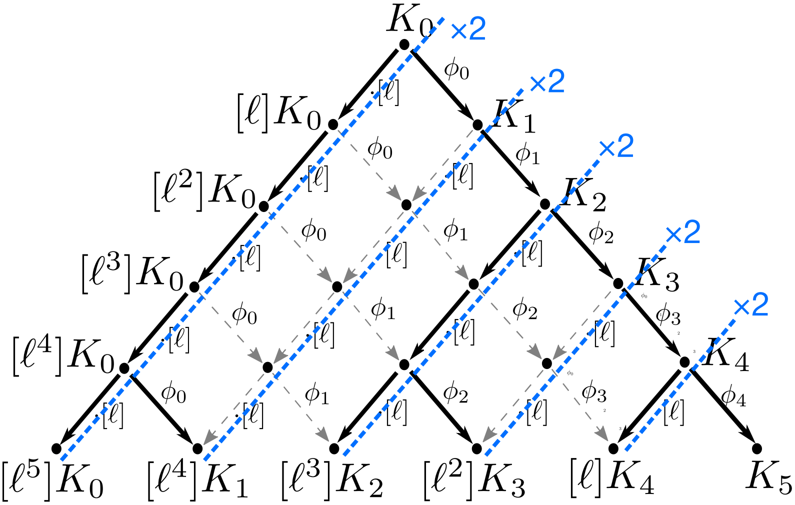
Optimal Strategy for Doubling-Isogeny Trade-off



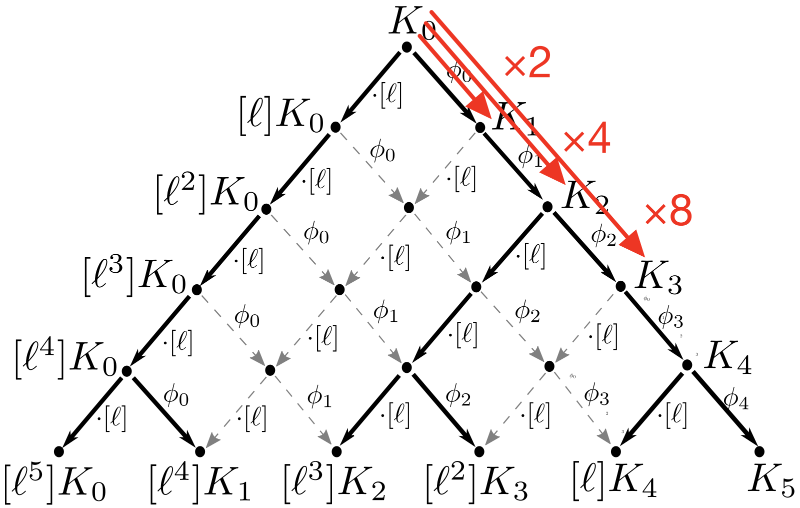
Optimal Strategy for Doubling-Isogeny Trade-off



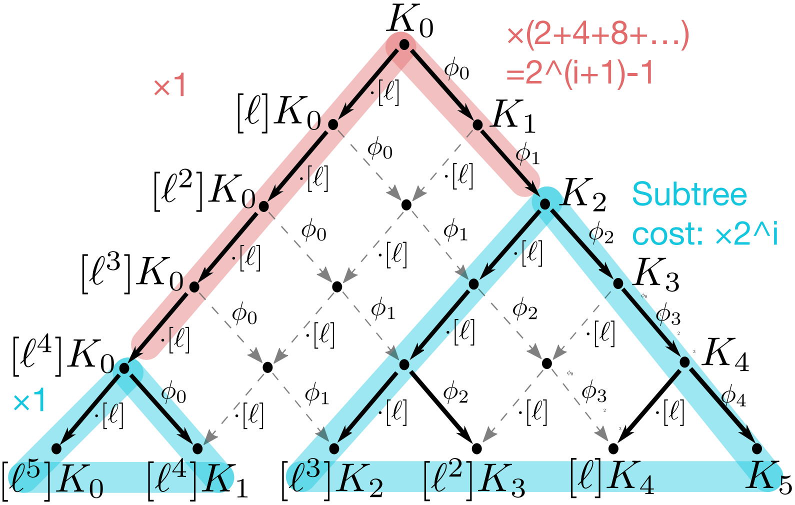
Optimal Strategy for Doubling-Isogeny Trade-off



Optimal Strategy for Doubling-Isogeny Trade-off



Optimal Strategy for Doubling-Isogeny Trade-off



Plan

Sort-and-Merge

- Standard MitM: hash-table (left tree), lookups (right tree)
  - $O(N)$ insertions/lookups
  - "Optimal" if $O(1)$ random memory access
  - Physically impossible on large scale

Sort-and-Merge

- Standard MitM: hash-table (left tree), lookups (right tree)
  - $O(N)$ insertions/lookups
  - “Optimal” if $O(1)$ random memory access
  - Physically impossible on large scale

- Sort-and-Merge
  - $O(N \log N)$ comparisons/swaps
  - Mostly sequential/local access possible
  - The constant behind $\log N$ is small in practice (e.g. radix sort)

Sort-and-Merge

- Standard MitM: hash-table (left tree), lookups (right tree)
  - $O(N)$ insertions/lookups
  - "Optimal" if $O(1)$ random memory access
  - Physically impossible on large scale

- Sort-and-Merge
  - $O(N \log N)$ comparisons/swaps
  - Mostly sequential/local access possible
  - The constant behind $\log N$ is small in practice (e.g. radix sort)
  - (Adj et al. 2019) considered 2D-mesh sorting as physically optimal. However, it is not clear at which scale the physical limits start to apply.
  - Although we use only 2^{45} storage, the limit of that paper set to 2^{80} seems not that far to actually reach physical limitations.

Extreme Storage Minimization

1. Drop path information
2. Truncate *j*-invariants
   Example: two 2^{44}-sized trees, truncated to 64 bits
   $\Rightarrow \quad \approx 2^{44 \cdot 2 - 64} = 2^{24}$ collisions (false positives)

Extreme Storage Minimization

1. Drop path information
2. Truncate *j*-invariants
   Example: two 2^{44}-sized trees, truncated to 64 bits
   $\Rightarrow \quad \approx 2^{44\cdot 2-64} = 2^{24}$ collisions (false positives)

Stage 1: Find intersection between truncated *j*-invariants (truncated collisions)
Stage 2: Regenerate trees and check full *j*-invariants matching truncated collisions

Extreme Storage Minimization

1. Drop path information
2. Truncate j-invariants
   Example: two 2^{44}-sized trees, truncated to 64 bits
   $\Rightarrow \quad \approx 2^{44\cdot 2-64} = 2^{24}$ collisions (false positives)

Stage 1: Find intersection between truncated j-invariants (truncated collisions)
Stage 2: Regenerate trees and check full j-invariants matching truncated collisions

3. Sorted & dense sets can be compressed by storing successive differences
   Example: 2^{44} elements of 64 bits have average difference 2^{20} (64 $\rightarrow$ 20 bits compression)

Extreme Storage Minimization

1. Drop path information
2. Truncate j-invariants
   Example: two 2^{44}-sized trees, truncated to 64 bits
   $\Rightarrow \quad \approx 2^{44 \cdot 2 - 64} = 2^{24}$ collisions (false positives)

Stage 1: Find intersection between truncated j-invariants (truncated collisions)
Stage 2: Regenerate trees and check full j-invariants matching truncated collisions

3. Sorted & dense sets can be compressed by storing successive differences
   Example: 2^{44} elements of 64 bits have average difference 2^{20} (64 $\rightarrow$ 20 bits compression)

In $IKEp182, at least $\times 5 - 6$ compression rate (vs 44-bit path + 64-bit j-invariant part)

Plan

1. Introduction
2. Meet-in-the-Middle Isogeny Search
3. Computing a SIKE-tree
4. Intersecting two SIKE-trees
5. Application to $IKEp182
6. Conclusion

Application to \$IKEp182 (1/2)

\$IKEp182 challenge:

- $p = 2^{91}3^{57} - 1$ (182 bits); $e_A = 91, e_B = 57$
- 91 steps split: 45 (left tree) + 44 (right tree) + 2 (A leakage)
- Conjugation trick (Costello et al. 2020): left tree size 2^{44}
- MitM: compute and intersect two sets of 2^{44} j-invariants

Application to $IKEp182 (2/2)

Process:

1. **Tree-DFS (Stage 1)**: 4.2 core-years and 256 TiB disk space (unoptimized, 70+ TiB should be enough)

Application to \$IKEp182 (2/2)

Process:

1. **Tree-DFS (Stage 1)**: 4.2 core-years and 256 TiB disk space (unoptimized, 70+ TiB should be enough)
2. **Sorting**: sort 2 GiB chunks / core locally

Application to $IKEp182 (2/2)

Process:

1. **Tree-DFS (Stage 1)**: 4.2 core-years and 256 TiB disk space (unoptimized, 70+ TiB should be enough)
2. **Sorting**: sort 2 GiB chunks / core locally
3. **Merge-1**: 2 GiB chunks $\rightarrow$ 512 GiB chunks (256 to 1); 0.15 core-years

Application to \$IKEp182 (2/2)

Process:

1. **Tree-DFS (Stage 1)**: 4.2 core-years and 256 TiB disk space (unoptimized, 70+ TiB should be enough)
2. **Sorting**: sort 2 GiB chunks / core locally
3. **Merge-1**: 2 GiB chunks $\rightarrow$ 512 GiB chunks (256 to 1); 0.15 core-years
4. **Merge-2**: 512 GiB chunks $\rightarrow$ 2 TiB compressed chunks (8 to 1); 0.08 core-years

Application to \$IKEp182 (2/2)

Process:

1. **Tree-DFS (Stage 1)**: 4.2 core-years and 256 TiB disk space (unoptimized, 70+ TiB should be enough)
2. **Sorting**: sort 2 GiB chunks / core locally
3. **Merge-1**: 2 GiB chunks $\rightarrow$ 512 GiB chunks (256 to 1); 0.15 core-years
4. **Merge-2**: 512 GiB chunks $\rightarrow$ 2 TiB compressed chunks (8 to 1); 0.08 core-years
5. **Sieve-3**: intersect groups of 4x4 chunks (each tree) (8 TiB x 8 TiB), parallelization; 0.77 core-years
   Found $16\ 777\ 119$ out of $16\ 777\ 216$ expected collisions

Application to $IKEp182 (2/2)

Process:

1. **Tree-DFS (Stage 1)**: 4.2 core-years and 256 TiB disk space (unoptimized, 70+ TiB should be enough)
2. **Sorting**: sort 2 GiB chunks / core locally
3. **Merge-1**: 2 GiB chunks $\rightarrow$ 512 GiB chunks (256 to 1); 0.15 core-years
4. **Merge-2**: 512 GiB chunks $\rightarrow$ 2 TiB compressed chunks (8 to 1); 0.08 core-years
5. **Sieve-3**: intersect groups of 4x4 chunks (each tree) (8 TiB x 8 TiB), parallelization; 0.77 core-years
   Found $16\ 777\ 119$ out of $16\ 777\ 216$ expected collisions
6. **Tree-DFS (Stage 2)**: 4.2 core-years to find matching j-invariants and paths

Application to $IKEp182 (2/2)

Process:

1. **Tree-DFS (Stage 1)**: 4.2 core-years and 256 TiB disk space (unoptimized, 70+ TiB should be enough)
2. **Sorting**: sort 2 GiB chunks / core locally
3. **Merge-1**: 2 GiB chunks $\rightarrow$ 512 GiB chunks (256 to 1); 0.15 core-years
4. **Merge-2**: 512 GiB chunks $\rightarrow$ 2 TiB compressed chunks (8 to 1); 0.08 core-years
5. **Sieve-3**: intersect groups of 4x4 chunks (each tree) (8 TiB x 8 TiB), parallelization; 0.77 core-years
   Found $16\ 777\ 119$ out of $16\ 777\ 216$ expected collisions
6. **Tree-DFS (Stage 2)**: 4.2 core-years to find matching j-invariants and paths

Total: 9.5 core-years and 256 TiB (70+ TiB should be enough)

Plan

Conclusion

- Optimizations for MitM-based isogeny search
- Useful generic tool: sort-and-merge intersection of large sets of 64-bit elements
- The paper contains discussion on scalability of the approach

ia.cr/2021/1421 github.com/cryptolu/sike_mitm

1

Conclusion

- Optimizations for MitM-based isogeny search
- Useful generic tool: sort-and-merge intersection of large sets of 64-bit elements
- The paper contains discussion on scalability of the approach
  1. reuse of the 89-bit path search does not improve vOW-based estimations
  2. using more RAM (e.g. 2^{80}) requires design and analysis of the architecture (2D-mesh may be too pessimistic)

ia.cr/2021/1421 github.com/cryptolu/sike_mitm

1

Conclusion

- Optimizations for MitM-based isogeny search
- Useful generic tool: sort-and-merge intersection of large sets of 64-bit elements
- The paper contains discussion on scalability of the approach
  1. reuse of the 89-bit path search does not improve vOW-based estimations
  2. using more RAM (e.g. 2^{80}) requires design and analysis of the architecture (2D-mesh may be too pessimistic)
  3. $IKEp217 challenge requires 1M more computations (same storage), ≈ 1 HPC-year

ia.cr/2021/1421 github.com/cryptolu/sike_mitm

1

Conclusion

- Optimizations for MitM-based isogeny search
- Useful generic tool: sort-and-merge intersection of large sets of 64-bit elements
- The paper contains discussion on scalability of the approach
  1. reuse of the 89-bit path search does not improve vOW-based estimations
  2. using more RAM (e.g. 2^{80}) requires design and analysis of the architecture (2D-mesh may be too pessimistic)
  3. ~~$IKEp217 challenge requires 1M more computations (same storage), ≈ 1 HPC-year~~
  4. (Castryck and Decru 2022) ≈ 1 laptop-weekend[1]

ia.cr/2021/1421 github.com/cryptolu/sike_mitm

[1] Actually, a few laptop-seconds, see optimized implementation by (Oudompheng and Pope 2022)

References I

Adj, Gora et al. (Aug. 2019). "On the Cost of Computing Isogenies Between Supersingular Elliptic Curves". In: *SAC 2018*. Ed. by Carlos Cid and Michael J. Jacobson Jr: vol. 11349. LNCS. Springer, Heidelberg, pp. 322–343. DOI: 10.1007/978-3-030-10970-7_15.

Castryck, Wouter and Thomas Decru (2022). *An efficient key recovery attack on SIDH (preliminary version)*. Cryptology ePrint Archive, Paper 2022/975. https://eprint.iacr.org/2022/975.

Costello, Craig et al. (May 2020). "Improved Classical Cryptanalysis of SIKE in Practice". In: *PKC 2020, Part II*. Ed. by Aggelos Kiayias et al. Vol. 12111. LNCS. Springer, Heidelberg, pp. 505–534. DOI: 10.1007/978-3-030-45388-6_18.

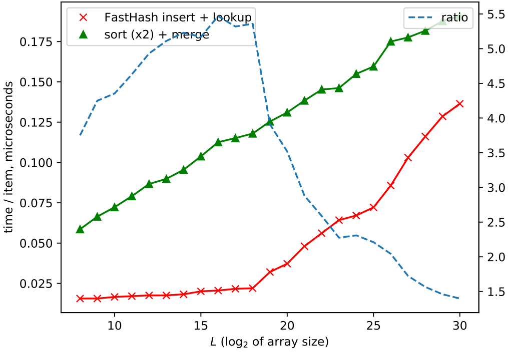
Galbraith, Steven D. (1999). "Constructing Isogenies between Elliptic Curves Over Finite Fields". In: *LMS Journal of Computation and Mathematics* 2, pp. 118–138.

References II

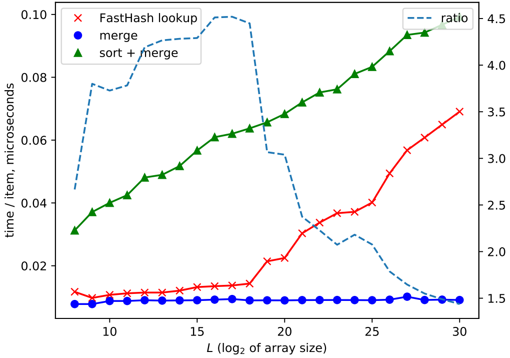
Oudompheng, Rémy and Giacomo Pope (2022). *SageMath implementation of the Castryck-Decru Attack on SIDH*. https://github.com/jack4818/Castryck-Decru-SageMath.

Comparison of HashTable and SortMerge on a PC

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



Intersecting two arrays without precomputation



Lookup in an array with precomputation