SIKE

Yi-Kai Liu

NIST

November 2021

Not for distribution

High-level description: SIDH (Supersingular Isogeny Diffie-Hellman)

- Isogeny graph
  - Vertices: Elliptic curves
  - Edges: Isogenies
    (maps from one curve to another)

- Claim: Given two curves E and E', finding an isogeny between them is hard
  - Like computing discrete logs, but in a graph rather than a group

De Feo, Jao, Plut (2011) https://eprint.iacr.org/2011/506

High-level description: SIDH (Supersingular Isogeny Diffie-Hellman)

- Isogenies

Let E_1 and E_2 be elliptic curves defined over a finite field $\mathbb{F}_q$. An isogeny $\phi : E_1 \to E_2$ defined over $\mathbb{F}_q$ is a non-constant rational map defined over $\mathbb{F}_q$ which is also a group homomorphism from $E_1(\mathbb{F}_q)$ to $E_2(\mathbb{F}_q)$

- Supersingular vs ordinary

An endomorphism of an elliptic curve E defined over $\mathbb{F}_q$ is an isogeny $E \to E$ defined over $\mathbb{F}_{q^m}$ for some m. The set of endomorphisms of E together with the zero map forms a ring under the operations of pointwise addition and composition; this ring is called the endomorphism ring of E and denoted $\mathrm{End}(E)$. The ring $\mathrm{End}(E)$ is isomorphic either to an order in a quaternion algebra or to an order in an imaginary quadratic field [37, V.3.1]; in the first case we say E is supersingular and in the second case we say E is ordinary.

  - Let A be a ring, and a finite-dimensional algebra over Q. Let O be a sub-ring of A.
  - O is an order iff O is a Z-lattice in A, and the span of O (over Q) is A

High-level description: SIDH (Supersingular Isogeny Diffie-Hellman)

- Isogeny classes

Two elliptic curves E_1 and E_2 defined over $\mathbb{F}_q$ are said to be isogenous over $\mathbb{F}_q$ if there exists an isogeny $\phi\colon E_1 \to E_2$ defined over $\mathbb{F}_q$. A theorem of Tate states that two curves E_1 and E_2 are isogenous over $\mathbb{F}_q$ if and only if $\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$ [45, §3]. Since every isogeny has a dual isogeny [37, III.6.1], the property of being isogenous over $\mathbb{F}_q$ is an equivalence relation on the finite set of $\bar{\mathbb{F}}_q$-isomorphism classes of elliptic curves defined over $\mathbb{F}_q$. Accordingly, we define an isogeny class to be an equivalence class of elliptic curves, taken up to $\bar{\mathbb{F}}_q$-isomorphism, under this equivalence relation.

  - Isogeny f and dual isogeny f' satisfy $f \circ f' = [n]$,
    where [n] is the isogeny that maps e to ne (multiplication by n)

- Note: isogenies and isomorphisms are different things.
  Roughly speaking, an isomorphism is an isogeny of degree 1.

High-level description: SIDH (Supersingular Isogeny Diffie-Hellman)

- Isogeny graphs

2.2. **Isogeny graphs.** An isogeny graph is a graph whose nodes consist of all elliptic curves in $\mathbb{F}_q$ belonging to a fixed isogeny class, up to $\bar{\mathbb{F}}_q$-isomorphism (so that two elliptic curves which are isomorphic over $\bar{\mathbb{F}}_q$ represent the same node in the graph). In practice, the nodes are represented using j-invariants, which are

Every supersingular elliptic curve in characteristic p is defined over either $\mathbb{F}_p$ or $\mathbb{F}_{p^2}$ [37], so it suffices to fix $\mathbb{F}_q = \mathbb{F}_{p^2}$ as the field of definition for this discussion. Thus, in contrast to ordinary curves, there are a finite number of isomorphism classes of supersingular curves in any given isogeny class; this number is in fact $g+1$, where g is the genus of the modular curve $X_0(p)$, which is roughly $p/12$. It turns out that all supersingular curves defined over $\mathbb{F}_{p^2}$ belong to the same isogeny class [27]. For a fixed prime value of $\ell \neq p$, we define the vertices of the supersingular isogeny graph $\mathcal{G}$ to consist of these g isomorphism classes of curves, with edges given by isomorphism classes of degree-ℓ isogenies, defined as follows: two isogenies $\phi_1, \phi_2 : E_i \to E_j$

- Claim: Given two curves E and E', finding an isogeny between them is hard
  - Like computing discrete logs, but in a graph rather than a group

High-level description: SIDH (Supersingular Isogeny Diffie-Hellman)

- Claim: Given two curves E and E', finding an isogeny between them is hard
  - Like computing discrete logs, but in a graph rather than a group

- However, the graph has algebraic structure, which we need to use
  - How to compute an isogeny?
  - Make Alice and Bob's operations "commute," to do key exchange
  - Reveal some extra information (torsion points)

  - Cryptanalysis: torsion-point attacks
  - Quantum cryptanalysis? (ordinary vs supersingular curves)

High-level description: SIDH (Supersingular Isogeny Diffie-Hellman)

- Specifying and computing isogenies, using their kernels

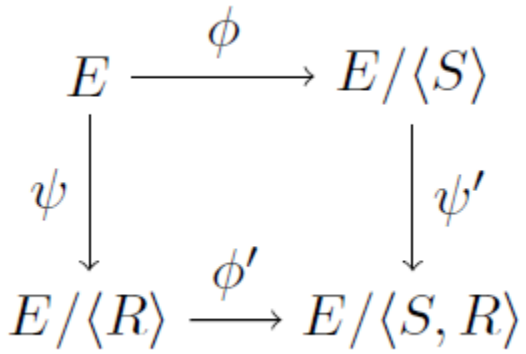
multiplicities) of degree ℓ originating from any given such supersingular curve. Given an elliptic curve E and a finite subgroup Φ of E, there is up to isomorphism a unique isogeny $E \to E'$ having kernel Φ [37, III.4.12]. Hence we can identify an isogeny by specifying its kernel, and conversely given a kernel subgroup the corresponding isogeny can be computed using Vélu's formulas [48]. Typically, this correspondence is of little use, since the kernel, or any representation thereof, is usually as unwieldy as the isogeny itself. However, in the special case of kernels generated by $\mathbb{F}_{p^2}$-rational points of smooth order, specifying a generator of the kernel allows for compact representation and efficient computation of the corresponding isogeny, as we

- "Points of smooth order" => l-torsion group, where l is smooth and composite

The ℓ-torsion group of E, denoted $E[\ell]$, is the set of all points $P \in E(\bar{\mathbb{F}}_q)$ such that ℓP is the identity. For ℓ such that $p \nmid \ell$, we have $E[\ell] \cong \mathbb{Z}/\ell\mathbb{Z} \oplus \mathbb{Z}/\ell\mathbb{Z}$.

High-level description: SIDH (Supersingular Isogeny Diffie-Hellman)

- Making Alice and Bob's operations "commute"
- Let R be in the torsion group $E[l_1]$
- Let S be in the torsion group $E[l_2]$



- Knowing ψ and E/<S>, want to compute E/<S,R>
- Need a hint: the action of ϕ on $E[l_1]$. This lets us compute ψ'

- Knowing ϕ and E/<R>, want to compute E/<S,R>
- Need a hint: the action of ψ on $E[l_2]$. This lets us compute ϕ'

- Note: For security, need $\gcd(l_1, l_2) = 1$

High-level description: SIDH (Supersingular Isogeny Diffie-Hellman)

$\mathcal{A}$		$\mathcal{B}$
		$\langle P_B, Q_B \rangle = E_0[\ell_B^{e_B}]$
Input: A, B, sID		**Input:** B
$m_A, n_A \in_R \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$		$m_B, n_B \in_R \mathbb{Z}/\ell_B^{e_B}\mathbb{Z}$
$\phi_A := E_0/\langle [m_A]P_A + [n_A]Q_A \rangle$		$\phi_B := E_0/\langle [m_B]P_B + [n_B]Q_B \rangle$
	$A, \text{sID}, \phi_A(P_B), \phi_A(Q_B), E_A \longrightarrow$	
	$B, \text{sID}, \phi_B(P_A), \phi_B(Q_A), E_B \longleftarrow$	
$E_{AB} := E_B/\langle [m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A) \rangle$		$E_{BA} := E_A/\langle [m_B]\phi_A(P_B) + [n_B]\phi_A(Q_B) \rangle$
Output: $j(E_{AB}), \text{sID}$		**Output:** $j(E_{BA}), \text{sID}$

Jao et al (2020) https://sike.org/files/SIDH-spec.pdf

Proposed standard: SIKE (Supersingular Isogeny Key Encapsulation)

- Implementation details:
- Montgomery curves
- Torsion groups $E[2^e]$ and $E[3^{e'}]$

- Clever algorithms for elliptic curves (https://eprint.iacr.org/2016/413)
- Public key compression: 41% shorter, somewhat slower, not compatible with uncompressed scheme (https://eprint.iacr.org/2019/499)

- Public key encryption, w/ CPA security
- Key encapsulation, w/ CCA security (decaps() does re-encryption)

1. A generic reference implementation written exclusively in portable C with simple algorithms to compute isogeny and field operations, using GMP for multi-precision arithmetic,

2. An optimized implementation written exclusively in portable C that includes efficient algorithms to compute isogeny and field operations,

3. An additional, optimized implementation for x64 platforms that exploits x64 assembly,

4. An additional, optimized implementation for x64 platforms that exploits x64 assembly and public key compression (§1.5),

5. An additional, optimized implementation for ARM64 platforms that exploits ARMv8 assembly,

6. An additional, optimized implementation for ARM Cortex M4 platforms that exploits ARM thumb assembly,

7. An additional, speed-optimized VHDL model for FPGA and ASIC platforms that parallelizes various aspects of the isogeny computation and field operations, and

8. An additional, simple textbook implementation written exclusively in portable C, using elliptic curves in short Weierstrass form.

All implementations except implementations number 1 and 8 are protected against timing and cache attacks at the software level. Specifically, they avoid the use of secret address accesses and secret branches.

SIKE implementations: short keys, slow computation

Scheme	secret key sk	public key pk	ciphertext ct	shared secret ss
SIKEp434	374	330	346	16
SIKEp503	434	378	402	24
SIKEp610	524	462	486	24
SIKEp751	644	564	596	32
SIKEp434_compressed	350	197	236	16
SIKEp503_compressed	407	225	280	24
SIKEp610_compressed	491	274	336	24
SIKEp751_compressed	602	335	410	32

Table 2.2: Size (in bytes) of inputs and outputs in SIKE.

Scheme	KeyGen	Encaps	Decaps	total (Encaps + Decaps)
Optimized Implementation				
SIKEp434	56,378	90,773	96,592	187,365
SIKEp503	85,744	140,781	149,972	290,753
SIKEp610	160,401	294,628	296,577	591,205
SIKEp751	288,827	468,175	502,983	971,158
Additional implementation using x64 assembly				
SIKEp434	5,927	9,681	10,343	20,024
SIKEp503	8,243	13,544	14,415	27,959
SIKEp610	14,890	27,254	27,445	54,699
SIKEp751	25,197	40,703	43,851	84,553
Compressed SIKE implementation using x64 assembly				
SIKEp434_compressed	10,158	15,120	11,077	26,197
SIKEp503_compressed	14,452	21,190	15,733	36,923
SIKEp610_compressed	26,360	37,470	29,216	66,686
SIKEp751_compressed	40,935	63,254	46,606	109,860

Table 2.1: Performance (in thousands of cycles) of SIKE on a 3.4GHz Intel Core i7-6700 (Skylake) processor. Cycle counts are rounded to the nearest 10^3 cycles.

Note: Reference implement ation was ~15x slower

Hand-optimized assembly code is ~10x faster

Scheme	KeyGen (stack)	Encaps (stack)	Decaps (stack)	static library	
				speed (-O3)	size (-Os)
Optimized Implementation					
SIKEp434	8,040	8,360	8,744	105,474	54,170
SIKEp503	8,072	8,456	8,904	120,202	58,714
SIKEp610	12,008	12,408	12,936	163,312	56,400
SIKEp751	13,912	14,040	14,696	164,810	60,162
Additional implementation using x64 assembly					
SIKEp434	8,136	8,456	8,840	108,208	56,672
SIKEp503	8,152	8,536	8,984	116,022	61,166
SIKEp610	13,536	12,512	12,112	135,470	68,494
SIKEp751	14,064	14,192	14,960	159,032	76,840
Compressed SIKE implementation using x64 assembly					
SIKEp434	16,920	15,640	17,000	593,138	458,418
SIKEp503	18,872	17,560	19,128	648,404	509,636
SIKEp610	23,824	22,048	24,144	866,796	698,148
SIKEp751	28,024	27,936	28,320	1,296,540	1,070,688

Table 2.3: Peak memory usage (stack memory, in bytes) and static library size (in bytes) of the various implementations of SIKE on a 3.4GHz Intel Core i7-6700 (Skylake) processor. Static libraries were obtained by compiling with clang and optimizing for speed (-O3) and for size (-Os).

Scheme	KeyGen	Encaps	Decaps	total (Encaps + Decaps)
Optimized implementation (portable)				
SIKEp434	718	1,175	1,254	2,429
SIKEp503	1,076	1,773	1,886	3,659
SIKEp610	2,011	3,701	3,722	7,423
SIKEp751	3,657	5,915	6,353	12,267
Additional implementation using ARMv7 Cortex-M4 assembly				
SIKEp434	54	89	95	184
SIKEp503	76	125	133	257
SIKEp610	134	246	248	493
SIKEp751	229	371	399	770

Table 2.5: Performance (in millions of cycles) of SIKE on a 168MHz 32-bit ARM Cortex-M4 processor on the STM32F407G-DISC1 board. Results are measured in ms and scaled to cycles using the nominal processor frequency. Cycle counts are rounded to the nearest 10^6 cycles.

Running on ARM Cortex-M4 w/o assembly code is way too slow!

SIKE implementations: more recent work

- Faster algorithms for compressed SIKE
- More recent implementations using RISC-V processors, FPGAs
- List of publications at https://sike.org/

Security strength (1)

- (Supersingular) isogeny walk problem
  - Meet-in-the-middle algorithm
  - Quantum and classical algorithms for finding claws and collisions
  - Recent work: accounting for the cost of memory (Jaques and Schanck (2019), https://eprint.iacr.org/2019/103)
- However, breaking SIKE could be strictly easier than this problem, because SIKE has public torsion-point information
  - Recent work: torsion-point attacks, affecting variants of SIKE (but not SIKE itself) (de Quehen et al (2021), https://eprint.iacr.org/2020/633)

Security strength (2)

- Quantum attacks
  - For ordinary elliptic curves, can find isogenies in quantum subexponential time (Childs et al (2010), https://arxiv.org/abs/1012.4019)
  - For supersingular elliptic curves, this attack fails, b/c endomorphism ring is non-commutative
  - But, for supersingular curves, a variant of this attack is possible, using torsion-point information. This affects variants of SIKE (but not SIKE itself) (Kutas et al (2021), https://eprint.iacr.org/2021/282)

- Algorithms/complexity/number theory questions
  - Deuring correspondence: supersingular curves ←→ maximal orders in quaternion algebra (non-constructive)
  - KLPT algorithm solves the quaternion analogue of the isogeny path problem (Kohel et al (2014), https://eprint.iacr.org/2014/505)
  - Reductions between path-finding, and computing the endomorphism ring (Eisentraeger et al (2018), https://eprint.iacr.org/2018/371)

Security strength (3)

- Active attacks, side-channel attacks, fault attacks, etc.
  - Early work: Galbraith et al (2016), https://eprint.iacr.org/2016/859
  - Many more recent results, see list: https://sike.org/
  - Need to use the KEM version of SIKE, which is CCA-secure
  - Need to protect against side-channels, using tricks that are (mostly?) standard in elliptic curve crypto
- Security proof: see Hofheinz et al (2017), https://eprint.iacr.org/2017/604

Theorem 1 ([19]). *For any IND-CCA adversary B against* KEM*, issuing at most q_G (resp. q_H) queries to the random oracle G (resp. H), there exists an IND-CPA adversary A against* PKE *with*

$$\mathrm{Adv}^{\mathrm{IND-CCA}}_{\mathrm{KEM}}(B) \leq \frac{2q_G + q_H + 1}{2^n} + 3 \cdot \mathrm{Adv}^{\mathrm{IND-CPA}}_{\mathrm{PKE}}(A).$$

	Target level	Classical gate requirement [48]	Classical security estimates		
			Total time [1] memory 2^{80} units	Gates [23, Fig. 4(d)] memory 2^{96} bits	x64 instructions [9] memory 2^{80} units
SIKEp434	1	143	128	142	143
SIKEp503	2	146	152	169*	169*
SIKEp610	3	207	189	209	210
SIKEp751	5	272	-	263*	262

Table 5.1: Classical security estimates of the three SIKE parameterizations according to Adj et al. [1], Jaques and Schanck [23], and Costello et al. [9]. Gate requirements and classical security estimates are all expressed as their base-2 logarithms. The values marked with (*) are not found in the actual papers. In the case of [9], we obtained the numbers for SIKEp503 using their scripts, where (for the half-sized isogenies used in vOW) the optimal strategy for the 2-torsion resulted in 362 doublings and 189 4-isogenies, and the optimal strategy for the 3-torsion yielded 229 triplings and 275 3-isogenies. In the former scenario, a vOW isogeny required over 2^{22} x64 instructions, and in the latter, over 2^{23} x64 instructions. In the case of [23], the RAM operations for SIKEp503 and SIKEp751 were taken from the width-restricted table in §5.2.

Summary

- Keys are very small
- To make it run fast, and protect against side-channel attacks, need non-trivial algorithms and assembly code
  - Build on past experience with elliptic curve crypto
  - (Remark: the arguments for SIKE are somewhat analogous to those for Falcon)
  - Interesting possibility: SIKE+ECDH hybrids
- Security is not completely understood?
  - Torsion-point attacks on variants of SIKE: these seem both new and fundamental? https://eprint.iacr.org/2020/633 , https://eprint.iacr.org/2021/282
  - Currently, these attacks don't affect SIKE. If they do in the future, there are plausible countermeasures: Costello (2021), https://eprint.iacr.org/2021/543