Faster Isogenies for Quantum-Safe SIKE

Rami Elkhatib, Brian Koziel, and Reza Azarderakhsh

{rami.elkhatib, brian.koziel, razarder}pqsecurity.com
PQSecre Technologies, LLC
Boca Raton, FL

Abstract. In the third round of the NIST PQC standardization process, the only isogeny-based candidate, SIKE, suffers from slow performance when compared to other contenders. The large-degree isogeny computation performs a series of isogenous mappings between curves, to account for about 80% of SIKE's latency. Here, we propose, implement, and evaluate a new method for computing large-degree isogenies of an odd power. Our new strategy for this computation avoids expensive recomputation of temporary isogeny results. We modified open-source libraries targeting x86, ARM64, and ARM32 platforms. Across each of these implementations, our new method achieves 10% and 5% speedups in SIKE's key encapsulation and decapsulation operations, respectively. Additionally, these implementations use 3% less stack space at only a 48 byte increase in code size. Given the benefit and simplicity of our approach, we recommend this method for current and emerging SIKE implementations.

Keywords: isogeny-based cryptography, post-quantum cryptography, SIKE, isogeny computations.

1 Introduction

Quantum computing technology is heralded as the next big leap in processing powers. This new type of computer will allow humanity to solve a wealth of difficult problems in applications such as medicine, finance, and data analytics. However, the dark side of these new quantum computers is that they are capable of breaking the foundational cryptography that secures the internet and our digital infrastructure. Namely, today's deployed public-key cryptosystems are protected by the difficulty to compute discrete logarithms and factorization, both of which are infeasible for classical computers once the numbers become large. However, in 1994, Peter Shor proposed a polynomial-time algorithm that can break these hard problems in conjunction with a large-scale quantum computer [44]. It is unclear when a large enough quantum computer will emerge, so there exist many estimates ranging from a few years to several decades.

With quantum computer fears in mind, the US National Institute for Standards and Technology (NIST) has initiated a post-quantum cryptography (PQC) standardization process for public-key cryptosystems [46]. Started in 2017, the standardization effort is currently in its third round, cutting the original 69 submissions to just 15. Quantum-safe schemes fall into a number of categories

ranging from lattices, codes, hashes, multi-variate equations, and isogenies. Unfortunately, there are no clear "drop-in replacements" for today's currently deployed technology. Each of the proposed quantum-resilient schemes feature performance, size, communication, and security tradeoffs.

This work focuses on optimizations to the NIST PQC candidate SIKE, which is based on the hardness of finding isogenies between supersingular elliptic curves. SIKE is currently a third round alternative scheme for key encapsulation mechanisms. SIKE is lauded for its small public keys, straightforward parameter selection, immunity to decryption errors, and understanding of its generic attacks. However, SIKE is an order of magnitude slower than lattices and other schemes and many are requiring further examination of its foundational security. Nevertheless, SIKE's small key and ciphertext sizes enable some applications that can take the hit to performance. NIST sees SIKE "as a strong candidate for future standardization with continued improvements." [2]

In this work, we present a faster method for computing the large-degree isogeny in SIKE for isogenies of an odd power. As we have observed, current open source implementations of SIKE such as that in the SIKE submission [5] or Microsoft's SIDH Library [15] utilize a slow strategy for computing large-degree isogenies that require a mix of 2 and 4-isogenies. Our new strategy modifies this strategy, achieving a nice speedup in SIKE's key generation and decapsulation operations. Currently, this applies to only the NIST Level 3 SIKE parameter set. However, new security analyses have proposed additional SIKE parameter sets for which we can and do apply this large-degree isogeny method.

Our contributions:

- We propose a new method for computing large-degree isogenies of an odd power.
- We provide explicit algorithms and formulas to speed up these large-degree isogenies.
- We implement, deploy, and evaluate our methodology on x86, ARM64, and ARM32 platforms.
- We achieve 10% and 5% speedups in SIKEp610's key encapsulation and decapsulation, respectively, with approximately a 3% improvement in stack usage in our experimental results.

2 Preliminaries: Isogenies on Elliptic Curves

In this section, we provide a brief overview of the large-degree isogeny and its use in SIKE.

2.1 Isogeny-Based Cryptography

History. Primarily, isogeny-based cryptography has focused on the use of isogenies on elliptic curves in cryptosystems. An isogeny is a non-constant rational

map between elliptic curves that is also a group homomorphism. The use of isogenies for a cryptosystem was first proposed in independent works by Couveignes [17] and Rostovtsev and Stolbunov [41] that were first published in 2006. These works proposed an isogeny-based key-exchange based on the hardness to compute isogenies between ordinary elliptic curves. Initially, these were believed to be resistant to quantum attacks. However, in 2010, Childs, Jao, and Soukharev [13] proposed a new quantum subexponential algorithm that computes isogenies between ordinary curves, thus breaking these schemes. In 2009, Charles, Lauter, and Goren [12] proposed an isogeny-based hash function that was based on the hardness to compute isogenies between supersingular elliptic curves which creates an expander graph. Then, in 2011, Jao and De Feo [27] modified the original isogeny-based key-exchange to now also utilize supersingular elliptic curves, creating the Supersingular Isogeny Diffie-Hellman (SIDH) key-exchange. Interestingly, the quantum algorithm from Childs, Jao, and Soukharev [13] relies on a commutative endomorphism ring. Ordinary elliptic curves have a commutative endomorphism ring, making them vulnerable, while supersingular elliptic curves have a non-commutative endomorphism ring. SIKE is an IND-CCA2 upgrade of SIDH that was submitted to the NIST PQC standardization process in 2017 [6]. Since the inception of SIDH in 2011 to the 4 years of SIKE in the NIST PQC process, these supersingular isogeny-based schemes remain unbroken.

Since the emergence of SIDH, many facets have isogeny-based cryptography have been explored through research. Many researchers have investigated security foundations from the isogeny hard problems [24,1,16,28] to side-channel attacks [26,47,32,31]. On the application side, we have seen the use of SIDH/SIKE public key compression [7,14,39,40], new isogeny-based digital signatures schemes [25,49], isogeny-based hybrid key exchange [15,9], isogeny-based password authenticated key exchange [45,8], and new isogeny-based key agreement with CSIDH [11]. Among isogeny-based cryptosystem implementations, we have seen a wealth of implementations including software [18,15,36,23,3,42,43], hardware [35,33,22,34,30,20,19], and even software-hardware co-design [38,21].

Elliptic Curves. Isogeny-based cryptography can be thought of as an extension of elliptic curve cryptography. Both operate on elliptic curves defined over finite fields. However, rather than stick to a single elliptic curve, isogenies on elliptic curves focus on the relationship between elliptic curves.

We define an elliptic curve E over a finite field $\mathbb{F}_q$ as the collection of all points (x, y) as well as the point at infinity that satisfy the short Weierstrass curve equation:

$$E/\mathbb{F}_q \ : \ y^2 = x^3 + ax + b$$

where $a, b, x, y \in \mathbb{F}_q$. By defining a method for adding points using geometry, this collection forms an abelian group over addition. The scalar point multiplication is a repeated use of point addition such that $Q = kP$, where $k \in \mathbb{Z}$ and $P, Q \in E$. Given P and k, it is simple to compute $Q = kP$ via a sequence point additions and point doublings. However, as the order of the elliptic curve E becomes very large (such as 2^{256}), finding k when given Q and P becomes in-

feasible for classical computers. Unfortunately, a large-scale quantum computer can efficiently solve this problem by using Shor's algorithm [44].

Isogenies. An elliptic curve isogeny over $\mathbb{F}_q$, $\phi : E \to E'$ is defined as a non-constant rational map from $E(\mathbb{F}_q)$ to $E'(\mathbb{F}_q)$ that preserves the point at infinity. Isogenies between elliptic curves can be computed over a kernel, $\phi : E \to E/\langle ker \rangle$, by using Vélu's formulas [48]. The degree of an isogeny is its degree as a rational map. Large-degree isogenies of the form ℓ^e can be computed by chaining e isogenies of degree ℓ. Lastly, an elliptic curve's j-invariant acts as an identifier of its isomorphism class. An isogeny moves from one elliptic curve isomorphism class to another isomorphism class.

Isogeny-based cryptography relies on the difficulty to compute isogenies between elliptic curves. For $\phi : E \to E'$, it is simple to compute an isogeny from E to E' given a finite kernel. However, when only given E and E', it is difficult to find the isogenous mapping between elliptic curves. SIDH and SIKE compute ϕ as a sequence of small-degree isogenies. This problem can be visualized as a walk on an isogeny graph of degree ℓ where each node represents an isomorphism class and the edges are isogenies of degree ℓ. For supersingular elliptic curves $E(\mathbb{F}_q)$, $q = p^2$, there are approximately $p/12$ isomorphism classes. For an isogeny graph over base degree ℓ, this is a complete graph where each node has $\ell + 1$ unique isogenies up to isomorphism of degree ℓ. SIDH and SIKE perform an isogeny walk by computing a large-degree isogeny of the form ℓ^e. In this walk, there are e steps of ℓ-degree isogenies.

Isogeny-Friendly Primes. SIDH and SIKE utilize a prime of the form $p = \ell_A^{e_A} \ell_B^{e_B} f \pm 1$, where ℓ_A and ℓ_B are small primes, e_A and e_B are positive integers, and f is a small cofactor to make the number prime. This prime is used to find a supersingular elliptic curve $E_0(\mathbb{F}_{p^2})$ and find torsion bases $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ that generate $E_0[\ell_A^{e_A}]$ and $E_0[\ell_B^{e_B}]$, respectively. Curve E_0 has order $\#E_0 = (p \mp 1)^2$ which has a smooth order so that rational isogenies of exponentially large degree can be computed efficiently as a chain of low-degree isogenies. Alice performs her large-degree isogeny ϕ_A over the ℓ_A isogeny graph and Bob performs his large-degree isogeny ϕ_B over the ℓ_B isogeny graph.

The core Diffie-Hellman principle of SIDH and SIKE is that Alice and Bob each separately perform their large-degree isogenies over ℓ_A and ℓ_B graphs, respectively, exchange the resulting curve and a few extra points, and then perform their large-degree isogeny over the other party's public key. Both parties will have new elliptic curves for which ℓ_A and ℓ_B isogeny walks were performed, so Alice and Bob can use the elliptic curve's j-invariant as a shared secret. Since Alice and Bob perform separate walks, the SIDH and SIKE schemes are as strong as the weaker isogeny graph to attack. For security, it is desirable that $\ell_A^{e_A} \approx \ell_B^{e_B}$.

2.2 Supersingular Isogeny Key Encapsulation

The Supersingular Isogeny Key Encapsulation (SIKE) mechanism [5] allows for an IND-CCA2 key establishment between two parties. In this scheme, Alice and Bob want to agree on a shared secret. To accomplish this, Alice and Bob perform

Table 1. SIKE parameter sets for each NIST security level [5]. All sizes are in bytes.

SIKE Scheme	Security Level	As Strong as	Prime Form	Secret Key	Public Key	Cipher Text	Shared Secret
SIKEp434	NIST level 1	AES128	$p_{434} = 2^{216}3^{137} - 1$	374	330	346	16
SIKEp503	NIST level 2	SHA256	$p_{503} = 2^{250}3^{159} - 1$	434	378	402	24
SIKEp610	NIST level 3	AES192	$p_{610} = 2^{305}3^{192} - 1$	524	462	486	24
SIKEp751	NIST level 5	AES256	$p_{751} = 2^{372}3^{239} - 1$	644	564	596	32

Table 2. SIKE parameter sets for each round. "Level" indicates the NIST security level of the prime for the NIST round. [5]

Parameter Set	Prime Form	NIST Standardization Round			
		Round 1	Round 2	Round 3	Proposed in [37]
SIKEp434	$2^{216}3^{137} - 1$		Level 1	Level 1	
SIKEp503	$2^{250}3^{159} - 1$	Level 1	Level 2	Level 2	
SIKEp610	$2^{305}3^{192} - 1$		Level 3	Level 3	
SIKEp751	$2^{372}3^{239} - 1$	Level 3	Level 5	Level 5	
SIKEp964	$2^{486}3^{301} - 1$	Level 5			
SIKEp377 [37]	$2^{191}3^{117} - 1$				Level 1
SIKEp546 [37]	$2^{273}3^{172} - 1$				Level 3
SIKEp697 [37]	$2^{356}3^{215} - 1$				Level 5

their own isogeny walks over separate isogeny graphs. SIKE is protected by the computational supersingular isogeny (CSSI) problem [1].

SIKE API. SIKE is a key encapsulation mechnism split into three operations: key generation, key encapsulation, and key decapsulation. In the SIKE protocol, Bob wants to agree on a shared secret with Alice. He begins the protocol by performing key generation, where a secret key and public key are generated. He sends this public key to Alice over a public channel. Alice, upon receiving Bob's public key, performs key encapsulation to generate a ciphertext and shared secret. Alice then sends her ciphertext to Bob, again over a public channel. Bob then finalizes this protocol by performing key decapsulation over his secret key and Alice's ciphertext, generating a shared secret. From there, Alice and Bob can utilize this shared secret to exchange encrypted communications.

SIKE Parameters. The primary computations in SIKE include a double-point multiplication, large-degree isogeny, and SHAKE256 hashing. For efficiency and simplicity, SIKE fixes a prime of the form $p = 2^{e_A}3^{e_B} - 1$ where $2^{e_B} \approx 3^{e_B}$. The sizes of these isogeny graphs are the primary metric for SIKE's security strength. Based on known cryptanalysis and known attacks, the SIKE team has proposed SIKE parameter sets of the form SIKEpXXX, where "XXX" is the size of the prime in bits. Each SIKE parameter set targets a NIST Security Level from 1 to 5. NIST Security Level 1 is considered as hard to break as a brute-force attack on AES128, Level 2 is as hard as finding a hash collision of SHA256,

Level 3 is as hard as a brute-force attack on AES192, and Level 5 is as hard as a brute-force attack on AES256. Interestingly, SIKE's prime sizes have decreased in size over the course of the NIST PQC standardization process, showing that the CSSI problem was harder than originally thought.

SIKE's strongest advantage is its small public keys and ciphertexts. We summarize SIKE's round 3 parameter sets in Table 1. The communication overhead is for the uncompressed SIKE scheme. The compressed SIKE scheme further reduces the public key and ciphertext by almost half. Compared to other schemes, SIKE has the smallest public keys and almost the smallest ciphertext. Only Classic McEliece [10] has a smaller ciphertext. Kyber [4] is a lattice-based scheme with fairly small public keys, but SIKE's public keys and ciphertexts are more than half of Kyber's size for an equivalent security level. For instance, Kyber Security Level 1 has an 800 byte public key and 768 byte ciphertext.

SIKE Primes. In Table 2, we summarize the SIKE parameter sets for the NIST PQC standardization process. Based on cryptanalysis of [1,16] and the like, it was deemed that SIKE's Round 1 parameters were too conservative, so they were reduced for Round 2 and beyond. We also added the proposed primes from [37] as it is another work that considers the CSSI problem harder than expected, resulting in even smaller primes.

For these results, we note that SIKE parameters are not permanent and could continue to change based on security analysis. When searching for SIKE primes, there are also limited options. When choosing a random form for p, it is desirable that $2^{e_A} \approx 3^{e_B}$ so that low-level modular arithmetic is more efficient and there are fewer isogenies to perform. Based on the prime number theorem, there is approximately a $\frac{1}{\ln x}$ chance that a random integer x is prime. If we only choose odd numbers by using the form $2^{e_A}3^{e_B} - 1$, then there is a $\frac{2}{\ln x}$ chance that the number is prime. In practice, this means that a 400-bit candidate has a 1 in 139 chance of being prime and a 700-bit candidate has a 1 in 243 chance of being prime.

3 Proposed Method for Large-Degree Isogenies of Odd Degree

In this section, we propose a new method to compute the large-degree isogeny operation in SIKE. This method can be applied to any scheme that computes a large-degree isogeny of an odd power of the form ℓ^{2k+1}, where k is an integer. In terms of the parameter schemes presented in Table 2, this applies to each prime, as every prime has at least one of the 2-isogeny or 3-isogeny graph sizes of an odd power. However, this is only if it is more efficient to perform the large-degree isogenies as a chain of ℓ^2 isogenies rather than ℓ isogenies, which is the case for the 2-isogeny graphs, where practitioners compute it as a sequence of 4-isogenies. Thus, this method specifically applies to the SIKE schemes SIKEp610, SIKEp377, and SIKEp546 where the 2-isogeny graph has an odd power.

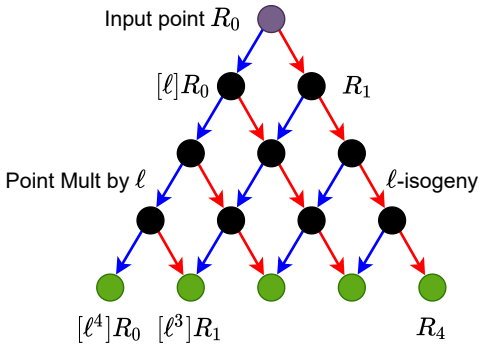

Fig. 1. Visualization of the large-degree isogeny computation.

3.1 Large-Degree Isogenies.

First, we review how the large-degree isogeny is traditionally performed. This computation accounts for 80% of the latency in SIDH and SIKE operations. Given a large-degree isogeny of the form ℓ^e, we compute this by chaining together e isogenies of degree ℓ. For a base curve E_0 and kernel point $R_0 = R$ of order ℓ^e we compute e isogenies of degree ℓ as follows:

$$E_{i+1} = E_i/\langle \ell^{e-i-1} R_i \rangle,\ \phi_i : E_i \to E_{i+1},\ R_{i+1} = \phi_i(R_i) \tag{1}$$

This computation can be visualized as traversing an acyclic graph in the shape of a triangle starting from the kernel point (R_0) to each of the leaves ($\ell^{e-i-1}R_i$) as is shown in Figure 1. Here, we are computing an isogeny of degree ℓ^5 by starting at the purple kernel node of order ℓ^5 and traversing to each of the green leaf nodes that has order ℓ with which we can compute an ℓ isogeny. When traversing this graph, there is an uneven cost to move left by computing a point multiplication by ℓ and to move right by evaluating an ℓ-degree isogeny over a point.

The simplest method to traverse this graph is the multiplication-based strategy with complexity $O(e^2)$ [27], which is the same as Equation 1. However, the only requirement for the large-degree isogeny is that an isogeny is computed at each of the leaf nodes. Thus, one can save specific points that act as a pivot to dramatically reduce the number of point multiplication and isogeny evaluations. As was first proposed in [18], optimal strategies can be devised as one of the least cost, with complexity $O(e \log e)$. The key insight when finding an optimal strategy is that an optimal strategy is composed of two optimal sub-strategies. The standard method to perform the large-degree isogeny is to utilize a precomputed strategy, whereby we cycle through the following steps until all ℓ-isogenies are computed.

1. Perform point multiplications by ℓ to traverse left to reach a leaf node, while storing pivot points according to a precomputed strategy

2. Compute an ℓ-degree isogeny to move from E_i to E_{i+1}
3. Evaluate ℓ-degree isogenies on all stored pivot points, translating them from E_i to E_{i+1}

Table 3. Fastest known point arithmetic and isogeny formulas for 2 and 4 isogenies [5].

Operation	$\#\mathbb{F}_{p^2}$ Add	$\#\mathbb{F}_{p^2}$ Sub	$\#\mathbb{F}_{p^2}$ Mult	$\#\mathbb{F}_{p^2}$ Sqr
2-Isogenies				
Point Doubling	2	2	4	2
Compute 2 Isogeny	0	1	0	2
Evaluate 2 Isogeny	3	3	**4**	**0**
4-Isogenies				
Point Quadrupling	4	4	8	4
Compute 4 Isogeny	4	1	0	4
Evaluate 4 Isogeny	3	3	**6**	**2**

Why 4 Isogenies? The simple reason that 4 isogenies are used instead of 2 isogenies is that 4 isogenies are more efficient. The cost of the fastest known isogeny formulas are shown in Table 3. In this table, each isogeny operation is made up of point multiplication by ℓ, computing an ℓ isogeny that performs a mapping between curves, and evaluating an ℓ isogeny that maps a point to a new isogenous curve. Also in this table, the most important computations are $\mathbb{F}_{p^2}$ multiplications and $\mathbb{F}_{p^2}$ squarings that require expensive field multiplications. In the SIKE landscape, $\mathbb{F}_{p^2}$ multiplications require 3 modular multiplications and $\mathbb{F}_{p^2}$ squarings require 2 modular multiplications. When comparing the cost of formulas, evaluating two 2-isogenies requires 8 $\mathbb{F}_{p^2}$ multiplications while evaluating a single 4-isogeny requires 6 $\mathbb{F}_{p^2}$ multiplications and 2 $\mathbb{F}_{p^2}$ squarings. Thus, 4-isogenies require 2 fewer field multiplications here. In the case of large-degree isogenies, hundreds of isogeny evaluations are performed, meaning that saving a few modular multiplications each step is valuable. Otherwise, 3 isogenies are known to be faster than 9 isogenies and larger isogeny algorithms are less explored as the costs of larger base degree isogenies do not scale well. The results of this paper are specifically applied to 2 and 4 isogenies, but given advances in isogeny formulas, could apply to other base degrees.

3.2 Large-Degree Isogenies of an Odd Power

Since large-degree isogenies of base degree 4 are faster than that of base degree 2, computing large-degree isogenies of the form 2^{2k} is preferred. However, given the limited availability of SIKE primes, such SIKE-friendly primes may not exist. When computing large-degree isogenies of the form 2^{2k+1}, at least one 2-isogeny must be computed.

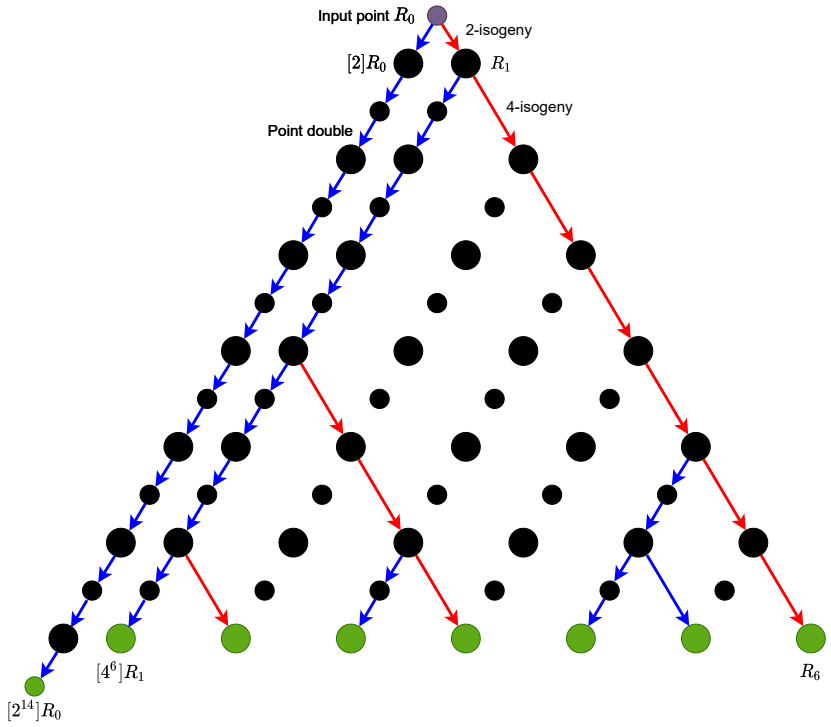

Fig. 2. Visualization of the state-of-the-art [5] methodology to compute large-degree isogenies of an odd power.

There are a few options for performing large-degree isogenies of an odd power. The simplest option may be to just skip the 2-isogeny. However, this weakens the security of the 2-isogeny graph and should only be done if the 2-isogeny graph is much larger than the 3-isogeny graph, i.e. $2^{e_A} \gg 3^{e_B}$.

Current Methodology. Currently, the SIKE submission [5] and other libraries perform the 2^{2k+1} isogeny by performing an initial 2-isogeny, and then proceeding with an optimized large-degree isogeny of base degree $2^{2k} = 4^k$. Thus, this approach starts with the point multiplication-based strategy and then proceeds with the optimized strategy.

A visualization of this approach is shown in Figure 2 with a toy example. Although the Figure shows the computation of a 2^{15} large-degree isogeny, it is simple to adapt this figure for ℓ and ℓ^2 isogenies for a different base degree. We use similar colors and representation as Figure 1. Here, a small-sized node represents a point of an odd power 2^{2k+1} and a medium-sized node represents a point of an even power $2^{2k} = 4^k$. Thus, we still traverse to the green leaf nodes, but we have the option to compute 2 or 4 isogenies at the leaves. Since

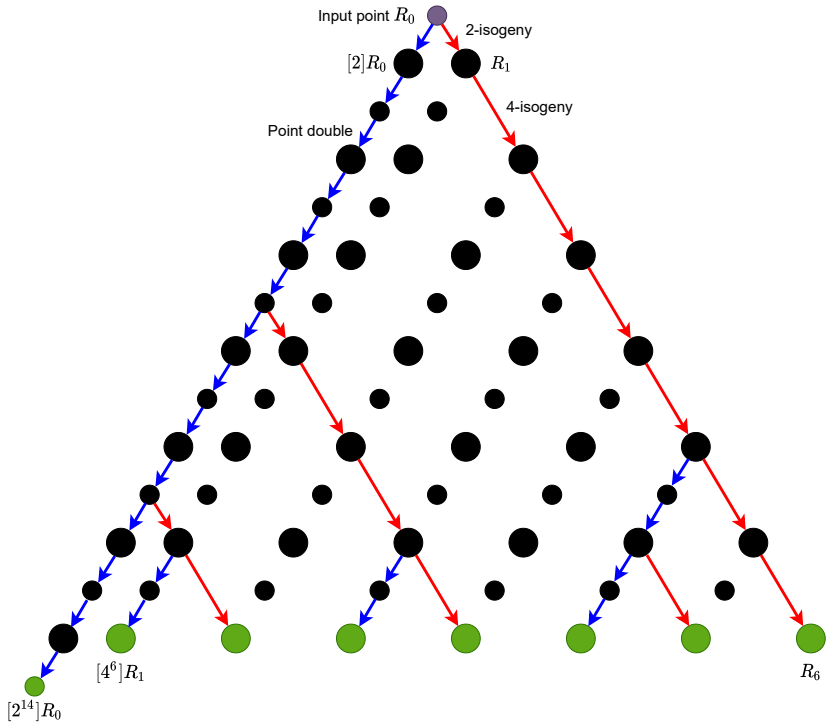

Fig. 3. Visualization of the proposed methodology to compute large-degree isogenies of an odd power.

4-isogenies are preferred, the example strategy utilizes seven 4-isogenies and one 2-isogeny to compute the 2^{15} large-degree isogeny.

As Figure 2 shows, there is a large amount of recomputation needed to compute the large-degree isogeny. Namely, this strategy performs two traversals from the top node to the left-most node. Of the 33 point doublings in this Figure, 27 point doublings are computed for the first 2-isogeny and subsequent 4-isogeny.

Proposed Methodology. To dramatically reduce this wasted traversal, our proposed fix is to store pivot points on the first traversal to the left, as is shown in Figure 3. In order to make this work, we need to store pivot points of an odd power order 2^{2k+1} that correspond to the point multiples used in the optimized 4-isogeny strategy. After performing the first 2-isogeny, we push each of these pivot points through the 2-isogeny, incrementing their order to $2^{2k} = 4^k$. Thus, we can then continue with an optimized 4-isogeny as normal.

Improvement. Between Figures 2 and 3, we showed two different methods for computing the large-degree isogeny 2^{15}. The state-of-the-art method requires 33 point doublings, 1 2-isogeny, and 11 4-isogenies, whereas the new method requires 22 point doublings, 3 2-isogenies, and 11 4-isogenies. Thus, for this toy

example, our proposed method exchanges 10 point doublings for 2 additional 2-isogeny evaluations. Based on the cost of current isogeny formulas shown in Table 3, this saves 16 $\mathbb{F}_{p^2}$ multiplication operations and 20 $\mathbb{F}_{p^2}$ squaring operations. When applied to something like SIKEp610, one sample strategy comparison showed that our new method exchanges 303 point doubling operations for 5 additional 2-isogeny point evaluations.

Discussion. With this proposed methodology, we chose the same policy of performing the 2-isogeny first. This is primarily because once done, no more of the (slower) 2-isogeny operations need to be performed in the middle of the large-degree isogeny. One interesting consideration was the use of mixing 2 and 4-isogeny operations. For example, one could perform about half of the isogeny computations and then the 2-isogeny. This makes the algorithm more complex and did not provide any real benefit from our analysis as the more 2-isogeny arithmetic you use, the more the large-degree isogeny slows down. Lastly, one could use a mix of 2 and 4-isogenies to compute new representations of pivot points. We did not find any benefit, but leave this open for further investigation.

4 Proposed Explicit Formulas for Large-Degree Isogenies

In this section, we propose a new algorithm and method to perform the large-degree isogeny within SIKE or SIDH. Although we directly apply this algorithm to the large-degree isogeny of odd power over base degree 2 for SIKE, we generalize this algorithm for any large-degree isogeny of a single base degree.

4.1 Proposed Efficient Algorithm for Large-Degree Isogenies with a Remainder

The large-degree isogeny computes an isogeny of degree ℓ^e by chaining together e isogenies of degree ℓ. Typically, ℓ is a small prime such as 2 or 3. When isogeny operations over degree ℓ^x, where x is a positive integer, are more efficient than performing x ℓ-isogenies, then ℓ^x isogenies should be used. However, when e is not divisible by x, there will be a remainder r. Thus, we represent $\ell^e = \ell^{x \cdot \lfloor e/x \rfloor + r}$, where $r = e \bmod x$.

For this algorithm we assume that the computations for ℓ^x isogeny computations, ℓ^x isogeny evaluations, and scalar point multiplications by ℓ^x will speed up the large-degree isogeny operation over other powers of ℓ. An isogeny computation computes an isogenous mapping between two elliptic curves given a kernel and an isogeny evaluation uses an isogenous mapping to map a point on one elliptic curve to its new representation on the isogenous curve. As is used by efficient SIDH and SIKE implementations, we can precompute an optimized strategy to traverse the large-degree isogeny computation as is shown in Figures 2 and 3. This significantly improves the complexity of the large-degree isogeny. For our large-degree isogeny with a remainder algorithm, we only need to precompute the optimized strategy for a large-degree isogeny of the form $\ell^{x \cdot \lfloor e/x \rfloor}$. Although there are several methods to store a strategy, the strategy simply represents the

number of point multiplications to traverse left. In most implementations, this is stored as a series of e positive integers. When using isogenies of base degree ℓ^x, we only need to store e/x positive integers.

New Large-Degree Isogeny Algorithm. Our proposed algorithm for large-degree isogenies with a remainder is shown in Algorithm 1. In general, the method boils down to computing the remainder isogeny ℓ^r first by computing a point multiplications by ℓ^x up to $\ell^{x \cdot \lfloor e/x \rfloor}$ where we store the pivot points along the way. These point multiplications produce a point of order ℓ^r for which an ℓ^r isogeny can be computed. After the ℓ^r isogeny, the remaining $\ell^{x \cdot \lfloor e/x \rfloor}$ isogenies can then be computed using an optimal strategy based on the latency of ℓ^x isogeny evaluations and point multiplications by ℓ^x.

Algorithm 1 uses a very similar format as the large-degree isogeny formulas found in the SIKE submission. [5]. We note that we use D to represent a deque, or double-ended queue, where each data element contains an index representing a point and the order of the point. h represents the remaining number of ℓ^x scalar point multiplications that could be applied, so $h = 1$ indicates the point is of order ℓ^x. Indeed, Lines 1 to 3 and Lines 18 to 35 are roughly equivalent to the state-of-the-art [5]. Lines 4 to 17 compute the remainder isogeny ℓ^r in a similar fashion as Lines 18 to 35, however only one isogeny needs to be computed. Specifically, Lines 4 through 9 perform the point multiplications by ℓ^x according to the large-degree isogeny strategy, Lines 10 to 11 compute the ℓ^r isogeny, and Lines 13 to 17 apply the ℓ^r isogeny to all stored pivot points.

Although we do not describe how to compute the ℓ^r isogeny, it can be computed in whatever the most efficient method is, whether that is a combination of smaller isogenies or a single ℓ^r isogeny. For simplicity, Algorithm 1 assumes that the ℓ^r isogeny is computed in one go. If multiple smaller isogenies are computed, then the Lines 11 through 17 of the algorithm simply need to be revised such that smaller point multiplications are performed to get a kernel point, whereby each smaller isogeny computation over the kernel is followed by a smaller isogeny evaluation over each stored pivot point.

Strategy Considerations. Interestingly, we found that only an optimal strategy needs to be made for the $\ell^{x \cdot \lfloor e/x \rfloor}$ large-degree isogeny. When applying the $\ell^{x \cdot \lfloor e/x \rfloor}$ strategy to a kernel point of order $\ell^{x \cdot \lfloor e/x \rfloor + r}$, the first series of scalar point multiplications will find pivot points of the order ℓ^{kx+r} where k is some positive integer whereas applying the strategy to a kernel point of order $\ell^{x \cdot \lfloor e/x \rfloor}$ will produce points of the order ℓ^{kx}. Thus, after applying the ℓ^r isogeny to the pivot points generated with the kernel point of order $\ell^{x \cdot \lfloor e/x \rfloor + r}$, we will naturally have pivot points of the order ℓ^{kx} as we need. The only caveat is that we need to perform one additional ℓ^x scalar point multiplication before the ℓ^r isogeny so that the point actually has order ℓ^r. Our Algorithm accounts for this by performing point multiplications until $h = 0$ rather than $h = 1$, as is done in Line 4. Furthermore, after the ℓ^r isogeny evaluations, we do not update the h indices in the deque as an ℓ^x isogeny has not been performed.

Complexity. For a large-degree isogeny of degree ℓ^e, this proposed Algorithm features the same $O(e \log e)$ complexity as the original use of optimized

Algorithm 1: Computing and evaluating an ℓ^e-isogeny with remainder

function ℓ`_e_iso`
- **Static Parameters:** Integer e from public parameters, Integer x for fastest ℓ^x isogenies, Integer $r = e \bmod x$, a *strategy* $(s_1, \ldots, s_{\lfloor e/x \rfloor - 1}) \in (\mathbb{N}^+)^{\lfloor e/x \rfloor - 1}$
- **Input:** Curve E_0 and point S on E_0 with exact order $\ell^e = \ell^{x \cdot \lfloor e/x \rfloor + r}$
- **Output:** Curve $E = E_0/\langle S \rangle$ by computing e isogenies of degree ℓ

Initialize empty deque D
`push(`D, $(\lfloor e/x \rfloor, S)$`)`
$E \leftarrow E_0$, $i \leftarrow 1$, $h \leftarrow \lfloor e/x \rfloor$
while $h \neq 0$ **do**
  $(h, R) \leftarrow$`pop(`D`)`
  `push(`$D, (h, R)$`)`
  for $j \leftarrow 0$ **to** s_i **do**
    $R \leftarrow$ `mult_by_`ℓ^x`(`R, E`)`
  `push(`$D, (h - s_i, R)$`)`, $i \leftarrow i + 1$
$(h, R) \leftarrow$ `pull(`D`)`
$(E', \phi) \leftarrow$ `compute_`ℓ^r`_iso(`E,R`)`
Initialize empty deque D'
while D *not empty* **do**
  $(h, R) \leftarrow$ `pull(`D`)`
  $R \leftarrow$ `evaluate_`ℓ^r`_iso(`E', ϕ, R`)`
  `push(`$D', (h, R)$`)`
$D \leftarrow D', E \leftarrow E'$
while D *not empty* **do**
  $(h, R) \leftarrow$`pop(`D`)`
  if $h = 1$
    $(E', \phi) \leftarrow$ `compute_`ℓ^x`_iso(`E,R`)`
    Initialize empty deque D'
    while D *not empty* **do**
      $(h, R) \leftarrow$ `pull(`D`)`
      $R \leftarrow$ `evaluate_`ℓ^x`_iso(`E', ϕ, R`)`
      `push(`$D', (h - 1, R)$`)`
    $D \leftarrow D', E \leftarrow E'$
  elif $0 < s_i < h$
    `push(`$D, (h, R)$`)`
    for $j \leftarrow 0$ **to** s_i **do**
      $R \leftarrow$ `mult_by_`ℓ^x`(`R, E`)`
    `push(`$D, (h - s_i, R)$`)`, $i \leftarrow i + 1$
  else
    Error: Invalid strategy
return $E = E_0/\langle S \rangle$

strategies proposed in [18]. However, when the fastest base degree isogeny ℓ^x has an x which is not a divisor of e, this method efficiently performs the ℓ^r isogeny in complexity $O(e)$ and has already stored pivot points for the following $\ell^{x \cdot \lfloor e/x \rfloor}$ large-degree isogeny in complexity $O(e \log e)$. This Algorithm also does not change the size of the precomputed strategy.

4.2 Proposed Faster 2-Isogeny Formulas for Large-Degree Isogenies.

In the SIKE submission [5] and similar SIKE libraries, we noted that the 2-isogeny formulas are not optimized for the scenario when multiple 2-isogeny evaluations are performed. Thus, we propose new and faster 2-isogeny formulas for this case that can be used in conjunction with our large-degree isogeny algorithm to accelerate the large-degree isogeny in SIKE parameters such as SIKEp610. Specifically, SIKEp610 performs a large-degree isogeny of degree 2^{305} which can be rewritten as $2^{2 \cdot 152+1} = 4^{152}2$. Our optimized formulas slightly alter the 2-isogeny APIs, moving computations over a point of order 2 from the 2-isogeny evaluation function to the 2-isogeny computation function, resulting in fewer field addition and subtractions when multiple 2-isogeny evaluations are performed.

For the following optimized formulas, we operate on Montgomery curves of the form $E/\mathbb{F}_q \ : \ by^2 = x^3 + ax^2 + x$. Similar to short Weierstrass curves, Montgomery curves are the set of all points (x, y) as well as the point at infinity that satisfy this equation. This curve is defined over the finite field $\mathbb{F}_q$, so, $b, y, x, a \in \mathbb{F}_q$.

There are additional representations of this curve for SIDH and SIKE to reduce the complexity of isogeny and point arithmetic. Notably, the introduction of a projective "C" term is used so that isogenies can be performed without inversions until the very end of a large-degree isogeny. The following formulas fix the b coefficient to 1 and simplify the Montgomery form to $E/\mathbb{F}_q \ : \ y^2 = x^3 + ax^2 + x$. Next, we denote the projective representation $(A \ : \ C)$ to denote the equivalence $(A \ : \ C) \sim (a \ : \ 1)$. Rather than keep track of A and C, the following equations represent the Montgomery curve with the values $(A_{24}^+ \ : \ C_{24}) \sim (A + 2C \ : \ 4C)$. Lastly, elliptic curve points are represented in the Kummer projective representation $(X \ : \ Z)$ where $x = X/Z$ and the y-coordinates are dropped.

2-Isogeny Computation. Our new 2-isogeny computation formula is shown in Algorithm 2. The 2-isogeny computation uses an input curve E and point of order 2 on the input curve $P_2 = (X_2 \ : \ Z_2)$ to compute a 2-isogenous elliptic curve E' such that $\phi \ : \ E \to E/\langle P_2 \rangle = E'$. Similar to the SIKE submission's formulas [5], the output curve is represented with coefficients $(A_{24}^+ \ : \ C_{24}) \sim (A' + 2C' \ : \ 4C')$, where A' and C' are the projective coefficients on curve E'. Our new formula adds new computations for constants K_1 and K_2 which are used in the 2-isogeny evaluation formula. The total cost of this computation is $2S + 3A$, where S is the cost of $\mathbb{F}_{p^2}$ squaring and A is the cost of $\mathbb{F}_{p^2}$ addition/subtraction.

2-Isogeny Evaluation. Our new 2-isogeny evaluation formula is shown in Algorithm 3. The 2-isogeny evaluation pushes a point $Q = (X_Q \ : \ Z_Q)$ on elliptic

Algorithm 2: Computing the 2-isogenous curve

function `2_iso_curve`

Input: $P_2 = (X_{P_2}\ Z_{P_2})$, where P_2 has exact order 2 on Montgomery curve $E_{A,C}$

Output: $(A^+_{24}\ :\ C_{24}) \sim (A' + 2C'\ 4C')$ corresponding to $E_{A',C'} = E_{A,C}/\langle P_2 \rangle$, and constants $(K_1, K_2) \in (\mathbb{F}_p^2)^2$

1 $A^+_{24} \leftarrow X^2_{P_2}$,
2 $C_{24} \leftarrow Z^2_{P_2}$,
3 $A^+_{24} \leftarrow C_{24} - A^+_{24}$,
4 $K_1 \leftarrow X_{P_2} - Z_{P_2}$,
5 $K_2 \leftarrow X_{P_2} + Z_{P_2}$,
6 **return** $A^+_{24}, C_{24}, (K_1, K_2)$

Algorithm 3: Evaluating a 2-isogeny at a point

function `2_iso_eval`

Input: Constants $(K_1, K_2) \in (\mathbb{F}_p^2)^2$ from `2_iso_curve`, and $Q = (X_Q\ :\ Z_Q)$ on Montgomery curve $E_{A,C}$

Output: $(X'_Q\ :\ Z'_Q)$ corresponding to $Q' \in E_{A'/C'}$, where $E_{A'/C'}$ is the curve 2-isogenous to $E_{A/C}$ output from `2_iso_eval`

1 $t_0 \leftarrow X_Q + Z_Q$,
2 $t_1 \leftarrow X_Q - Z_Q$,
3 $t_0 \leftarrow t_0 \cdot K_1$,
4 $t_1 \leftarrow t_1 \cdot K_2$,
5 $t_2 \leftarrow t_1 + t_0$,
6 $t_0 \leftarrow t_1 - t_0$,
7 $X'_Q \leftarrow t_2 \cdot X_Q$,
8 $Z'_Q \leftarrow t_0 \cdot Z_Q$,
9 **return** $Q' = (X'_Q\ :\ Z'_Q)$

curve E to its 2-isogenous representation $Q' = (X_{Q'}\ :\ Z_{Q'})$ on elliptic curve E' by using 2-isogeny constants outputted in the 2-isogeny computation. Our new formula no longer needs the point of order 2 to evaluate the 2-isogeny. We remark that this formula requires three temporary registers whereas the previous formula required four. The total cost of this computation is $4M + 4A$, where M is the cost of $\mathbb{F}_{p^2}$ multiplication and A is the cost of $\mathbb{F}_{p^2}$ addition/subtraction.

Table 4. Complexity of newly proposed 2-isogeny formulas versus the state-of-the-art.

Operation	$\#\mathbb{F}_{p^2}$ Add	$\#\mathbb{F}_{p^2}$ Sub	$\#\mathbb{F}_{p^2}$ Mult	$\#\mathbb{F}_{p^2}$ Sqr	#Temporary Values
SIKE Submission [5]					
Compute 2 Isogeny	0	1	0	2	0
Evaluate 2 Isogeny	3	3	4	0	4
This Work					
Compute 2 Isogeny	**1**	**2**	0	2	0
Evaluate 2 Isogeny	**2**	**2**	4	0	**3**

Efficiency of New Formulas. We summarize the cost of our 2-isogeny formulas versus the SIKE submission's [5] in Table 4. Our new formulas swap out an $\mathbb{F}_{p^2}$ addition and $\mathbb{F}_{p^2}$ subtraction from the evaluate 2 isogeny formula to the compute 2 isogeny formula. Since 4 isogenies are more efficient than

2-isogenies, it is expected that we will only ever compute a single 2-isogeny. However, with our new method we will evaluate multiple 2-isogenies, resulting in saving several $\mathbb{F}_{p^2}$ additions and subtractions. Interestingly, our proposed method uses fewer temporary registers in the 2-isogeny evaluation formula. As we see in our stack usage, this actually reduces the total stack usage of the SIKE operations that perform a large-degree isogeny of base degree 2. Given that this revision of formulas reduces stack size, we recommend its use even if our large-degree isogeny algorithm is not employed.

5 Benchmarking and Evaluation

Benchmarking Platforms. With this new method for large-degree isogenies of an odd power, we set out to benchmark it on various platforms. To target different computing capabilities, we chose SIKE implementations based on x86-64, ARM64, and ARM32. Our x86-64 platform was on an X1 Carbon 7th Generation laptop running an Intel Core i7-8565U processor at 1.8 GHz. Our ARM64 platform was a Raspberry Pi 4B device with 8 GB of RAM running an ARM Cortex-A72 processor at 1.5 GHz. Lastly, our ARM32 platform is the STM32F407 Discovery Kit which runs an ARM Cortex-M4 processor at up to 168 MHz. Although we only benchmark on software platforms, our improvements to the large-degree isogenies will most likely provide similar speedups to hardware and software-hardware designs.

x86/ARM64 Benchmarking. For the x86 and ARM64 devices, we used Microsoft's SIDH library [15] v3.4[1] as a base. This provides both SIDH and SIKE implementations for each of the SIKE parameter sets. Importantly, this provides optimized low-level arithmetic for both x86 and ARM64 devices. For the x86 implementation, we disabled TurboBoost. The X1 Carbon 7th Generation device was running Ubuntu 20.04 and the Raspberry Pi 4 is running Raspberry Pi OS (64 bit) release built on May 28, 2021. Although the 64-bit Raspberry Pi OS is still in beta, the ability to employ optimized ARM64 assembly improves performance. Both platforms utilized GNU GCC version 9.3.0 and were compiled with `-O3`.

ARM32 Benchmarking. On the ARM32 side, we used PQCRYPTO's open-source pqm4 [29] library[2]. The pqm4 library provides a benchmarking and testing framework for evaluating post-quantum schemes. It includes ARM Cortex-M4 implementations of schemes such as SIKE with hand-optimized assembly. In addition, there are simple benchmarking capabilities to analyze performance, stack usage, and code-size. Our numbers are obtained with the toolchain `arm-none-eabi-gcc 9.2.1`. We note that for speed benchmarking, the Cortex-M4 is locked to 24 MHz to avoid wait cycles caused by the memory controller.

Library Modifications. Both of these above libraries utilize code that originated in Microsoft's SIDH library and was extended for the SIKE submission [5]. The basic framework of the libraries, such as the directory structure or

[1] Commit @effa607 of https://github.com/microsoft/PQCrypto-SIDH
[2] Commit @844e7ca of https://github.com/mupq/pqm4

function names are generally consistent. Of the four NIST implementations in SIKE's round 3 package, this new method applies only to SIKEp610, which is at NIST Level 3. When upgrading these implementations for our new method of computing a large-degree isogeny, we only edited the `ec_isogeny.c` for new 2-isogeny formulas, `P610.c` for a new large-degree strategy, and `sidh.c` for Alice's large-degree isogeny algorithm. All other files were left untouched.

Table 5. Speedups achieved for SIKEp610 using newly proposed large-degree isogeny method.

Work	Timings [ms]				Timings [$cc \cdot 10^6$]			
	KeyGen	Encap	Decap	E+D	KeyGen	Encap	Decap	E+D
x86-64 i7-8565U @ 1.8 GHz								
Prior	9.5	17.1	17.3	34.5	17.0	30.9	31.2	62.0
This Work	9.5	15.6	16.5	32.0	17.0	28.0	29.7	57.7
Improvement	-	10.5%	4.8%	7.5%	-	10.5%	4.8%	7.5%
ARM64 ARM Cortex-A72 @ 1.5 GHz								
Prior	32.8	60.3	60.6	120.9	49.2	90.4	90.9	181.4
This Work	32.8	54.7	57.9	112.6	49.2	82.1	86.8	168.9
Improvement	-	10.2%	4.8%	7.4%	-	10.2%	4.8%	7.4%
ARM32 ARM Cortex-M4 @ 24 MHz								
Prior	4,972	9,139	9,196	18,335	119.3	219.3	220.7	440.0
This Work	4,972	8,295	8,774	17,069	119.3	199.1	210.6	409.9
Improvement	-	10.2%	4.8%	7.4%	-	10.2%	4.8%	7.4%

Table 6. Stack and code size improvements on ARM Cortex-M4 for SIKEp610 using newly proposed large-degree isogeny method.

Work	Stack [Bytes]			Code Size [Bytes]		
	KeyGen	Encap	Decap	.text	.data	.bss
ARM32 ARM Cortex-M4 @ 24 MHz						
Prior	10,528	10,952	11,416	29,936	0	0
This Work	10,528	10,624	11,088	29,984	0	0
Improvement [value]	-	328	328	-48	-	-
Improvement [%]	-	3.1%	3.0%	-0.16%	-	-

P610 Speedups. First, we summarize our performance improvements for SIKEp610 in Table 5. These results target the three major SIKE operations, including key generation, key encapsulation, and key decapsulation. The "E + D" column is the sum of the key encapsulation and key decapsulation timings. Since key generation generally only needs to be performed once, the "E + D" time has been used as an effective measure of SIKE's performance when deployed. Across

the board, key encapsulation is improved by just over 10% and key decapsulation is improved by just less than 5%, resulting in "E + D" improvement of about 7.5%.

P610 Stack and Code Size. In Table 6, we utilize pqm4 to analyze the stack usage and code size of our implementation. Interestingly, our approach reduces the stack usage by 3% for both key encapsulation and decapsulation. Our modifications do include 48 additional bytes for code size, but this is a neglible amount. Based on the analysis of our new 2-isogeny formulas, we believe our revised 2-isogeny computation and evaluation formulas are the primary reason for stack usage improvement. Furthermore, our changes actually impacted the max stack usage, which means that the large-degree isogeny over base degree 2 consumes the most stack within both key encapsulation and decapsulation.

Table 7. Speedups achieved for SIKEp610_compressed using newly proposed large-degree isogeny method.

Work	Timings [ms]				Timings $[cc \cdot 10^6]$			
	KeyGen	Encap	Decap	E+D	KeyGen	Encap	Decap	E+D
x86-64 i7-8565U @ 1.8 GHz								
Prior	17.2	24.1	18.5	42.6	30.9	43.4	33.3	76.7
This Work	16.3	24.1	17.6	41.7	29.3	43.4	31.7	75.1
Improvement	5.4%	-	5.0%	2.1%	5.4%	-	5.0%	2.1%

Table 8. Speedups achieved for SIKEp377 and SIKEp546 on x86-64 i7-8565U platform @ 1.8 GHz using newly proposed large-degree isogeny method.

Work	Timings [ms]				Timings $[cc \cdot 10^6]$			
	KeyGen	Encap	Decap	E+D	KeyGen	Encap	Decap	E+D
SIKEp377								
Prior	2.7	5.0	5.0	9.9	4.81	8.92	8.93	17.8
This Work	2.7	4.5	4.7	9.2	4.81	8.07	8.52	16.6
Improvement	-	10.5%	4.8%	7.5%	-	10.5%	4.8%	7.5%
SIKEp546								
Prior	7.2	13.1	13.2	26.3	12.9	23.6	23.7	47.3
This Work	7.2	11.8	12.5	24.3	12.9	21.3	22.5	43.8
Improvement	-	10.9%	5.5%	8.1%	-	10.9%	5.5%	8.1%

P610 Compressed Speedups. As a further testament to the robustness of our method, we apply it to the SIKEp610_compressed scheme. SIKE with key compression actually swaps the order of 2 and 3 isogenies. Now, key generation and key decapsulation perform the isogeny of base degree 2. Compared to the uncompressed scheme, SIKEp610_compressed only computes two large-degree

isogenies of base degree 2, one in key generation and one in key decapsulation, whereas SIKEp610(_uncompressed) computes three large-degree isogenies of base degree 2, two in key encapsulation and one in key decapsulation. Our results agree with this fact and show a 5.4% improvement in key generation and a 5.0% improvement in key decapsulation, leading to an "E + D" speedup of about 2.1%.

New P377 and P546 Speedups. Next, we apply our work to the newly proposed SIKE parameters featured in [37]. SIKEp377 and SIKEp546 are new parameter sets targeted at NIST Security Level 1 and 3, respectively. Since this work only provided implementations targeting x86 implementations, we only benchmarked it for our x86 platform. SIKEp377 and SIKEp546 timing results are summarized in Table 8. When comparing these results to the SIKEp610 results, the same 10% improvement for key encapsulation and 5% improvement for key decapsulation are achieved. These numbers are so close most likely because all three implementations feature a similar point multiplication to point evaluation ratio which is used for finding a large-degree isogeny strategy.

Table 9. Evaluation of SIKE performance scaling as prime size increases on x86-64 i7-8565U platform @ 1.8 GHz. Note that SIKEp377, SIKEp546, and SIKEp610 employ our revised large-degree isogeny method.

SIKE Parameter Set	Timings [ms]				Timings [$cc \cdot 10^6$]			
	KeyGen	Encap	Decap	E+D	KeyGen	Encap	Decap	E+D
SIKEp377	2.7	4.5	4.7	9.2	4.81	8.07	8.52	16.6
SIKEp434	3.8	6.2	6.7	12.9	6.90	11.2	12.1	23.2
SIKEp503	5.2	8.5	9.2	17.7	9.37	15.4	16.5	31.9
SIKEp546	7.2	11.8	12.5	24.3	12.9	21.3	22.5	43.8
SIKEp610	12.7	21.4	22.7	44.1	22.9	38.6	40.8	79.4
SIKEp751	15.8	25.4	27.3	52.8	28.4	45.8	49.2	95.0

SIKE Performance Scaling. As a final evaluation of our method, we examine the scaling of SIKE performance as the prime size increases. We summarize the results of all SIKE implementations on our x86 platform in Table 9. Our new large-degree isogeny method applies to SIKEp377, SIKEp546, and SIKEp610. All other schemes have been left unchanged and their performance is only to analyze scaling. For instance, prior SIKE results in the SIKE submission [5] show that key encapsulation is several percent faster than key decapsulation. This trend did not fit for SIKEp610, where key encapsulation and decapsulation results were almost the same. Since our new method accelerates SIKEp610 encapsulation by 10% and decapsulation by 5%, our results now show a several percent gap between encapsulation and decapsulation.

We visualize the SIKE performance scaling results as a graph in Figure 4. As a trendline, we added a polynomial of order 2, which generally fits well. This is to be expected as the size of the prime dictates the low-level modular arithmetic

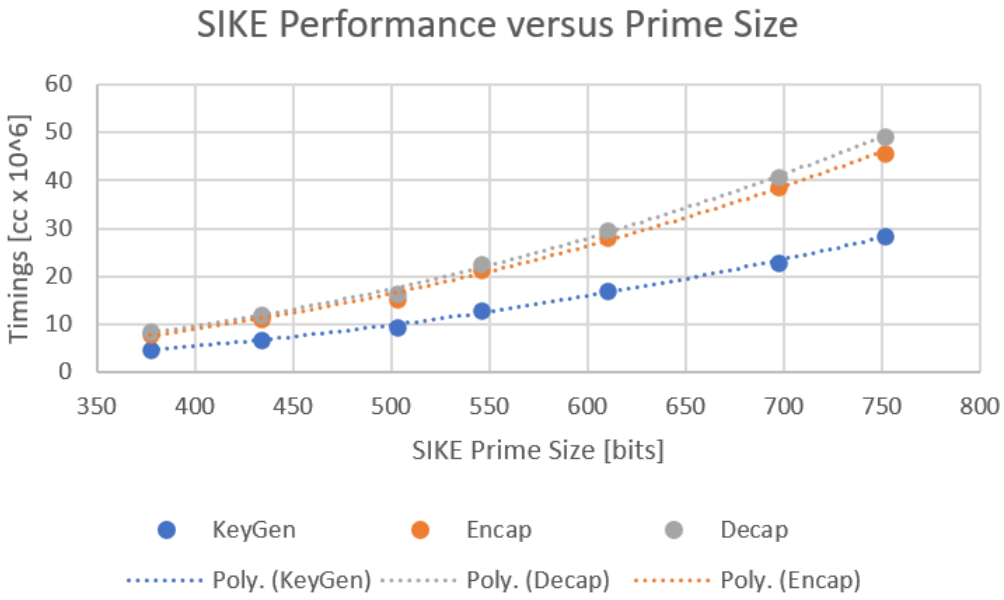

Fig. 4. Performance of SIKE versus the size of its prime on x86 platforms.

as well as the size of the large-degree isogeny. Low-level modular arithmetic is $O(p^2)$ for a prime p and the large-degree isogeny is $O(e\ log\ e)$, where $e = p/2$. Thus, a combined complexity of about $O(p^3\ log\ p)$ is achieved. The only blip in the performance graph is for SIKEp503, which may be because 512 bits fits very well in modern processors when computing modular multiplication.

6 Conclusion

In this paper, we proposed, implemented, and evaluated a new method for computing large-degree isogenies of an odd power. Our results accelerated uncompressed SIKEp610 key encapsulation and decapsulation operations by about 10% and 5%, respectively, also with a 3% improvement in stack usage. This proposed method is both simple to implement and applicable to a variety of SIKE applications. Besides SIKE parameter sets, we applied this method to newly proposed SIKE parameter sets and SIKE key compression. Given the benefit and simplicity of our approach with very few drawbacks, we recommend this method for current and emerging SIKE implementations.

7 Acknowledgment

The authors would like to thank the reviewers for their comments.

References

1. Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography – SAC 2018*, pages 322–343, Cham, 2019. Springer International Publishing.
2. Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process. 2020. NIST IR 8309.
3. Mila Anastasova, Reza Azarderakhsh, and Mehran Mozaffari Kermani. Fast strategies for the implementation of SIKE round 3 on ARM Cortex-M4. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1–13, 2021.
4. Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber: Algorithm specifications and supporting documentation (version 2.0). *Submission to the NIST Post-Quantum Standardization project*, 2019.
5. Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular Isogeny Key Encapsulation. *Submission to the NIST Post-Quantum Standardization project*, 2020.
6. Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular Isogeny Key Encapsulation. *Submission to the NIST Post-Quantum Standardization Project*, 2017.
7. Reza Azarderakhsh, David Jao, Kassem Kalach, Brian Koziel, and Christopher Leonardi. Key compression for isogeny-based cryptosystems. In *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography*, pages 1–10, 2016.
8. Reza Azarderakhsh, David Jao, Brian Koziel, Jason T. LeGrow, Vladimir Soukharev, and Oleg Taraskin. How not to create an isogeny-based PAKE. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *Applied Cryptography and Network Security - 18th International Conference, ACNS 2020, Rome, Italy, October 19-22, 2020, Proceedings, Part I*, volume 12146 of *Lecture Notes in Computer Science*, pages 169–186. Springer, 2020.
9. Reza Azarderakhsh, Rami El Khatib, Brian Koziel, and Brandon Langenberg. Hardware deployment of hybrid PQC. Cryptology ePrint Archive, Report 2021/541, 2021. `https://ia.cr/2021/541`.
10. Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. Classic McEliece: conservative code-based cryptography. *Submission to the NIST Post-Quantum Standardization project*, 2019.
11. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 395–427, Cham, 2018. Springer International Publishing.

12. Denis X. Charles, Kristin E. Lauter, and Eyal Z. Goren. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22(1):93–113, Jan 2009.
13. Andrew M. Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014.
14. Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. Efficient compression of SIDH public keys. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 679–706. Springer, 2017.
15. Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference*, pages 572–601, 2016.
16. Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes, and Fernando Virdia. Improved classical cryptanalysis of SIKE in practice. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 505–534. Springer, 2020.
17. Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006.
18. Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, Sep. 2014.
19. Rami El Khatib, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. Optimized algorithms and architectures for Montgomery multiplication for post-quantum cryptography. In *International Conference on Cryptology and Network Security*, pages 83–98. Springer, 2019.
20. R. Elkhatib, R. Azarderakhsh, and M. Mozaffari-Kermani. Highly optimized Montgomery multiplier for SIKE primes on FPGA. In *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*, pages 64–71, 2020.
21. Rami Elkhatib, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. Accelerated RISC-V for post-quantum SIKE. Cryptology ePrint Archive, Report 2021/597, 2021. `https://ia.cr/2021/597`.
22. Mohammad-Hossein Farzam, Siavash Bayat-Sarmadi, and Hatameh Mosanaei-Boorani. Implementation of supersingular isogeny-based Diffie-Hellman and key encapsulation using an efficient scheduling. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2020.
23. Armando Faz-Hernández, Julio López, Eduardo Ochoa-Jiménez, and Francisco Rodríguez-Henríquez. A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol. *IEEE Transactions on Computers*, 67(11):1622–1636, 2017.
24. Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In *Advances in Cryptology - ASIACRYPT 2016*, pages 63–91, 2016.
25. Steven D. Galbraith, Christophe Petit, and Javier Silva. Identification protocols and signature schemes based on supersingular isogeny problems. In *Advances in Cryptology – ASIACRYPT 2017*, pages 3–33, Cham, 2017.
26. Alexandre Gélin and Benjamin Wesolowski. Loop-abort faults on supersingular isogeny cryptosystems. In *Post-Quantum Cryptography : 8th International Workshop, PQCrypto 2017*, pages 93–106, 2017.

27. David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011*, pages 19–34, 2011.
28. Samuel Jaques and John M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 32–61. Springer, 2019.
29. Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. `https://github.com/mupq/pqm4`.
30. B. Koziel, A. Ackie, R. El Khatib, R. Azarderakhsh, and M. M. Kermani. SIKE'd up: Fast hardware architectures for supersingular isogeny key encapsulation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1–13, 2020.
31. Brian Koziel, Reza Azarderakhsh, and David Jao. An exposure model for supersingular isogeny Diffie-Hellman key exchange. In *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018*, pages 452–469, 2018.
32. Brian Koziel, Reza Azarderakhsh, and David Jao. Side-channel attacks on quantum-resistant supersingular isogeny Diffie-Hellman. In *Selected Areas in Cryptography – SAC 2017, 24th International Conference*, pages 64–81, 2018.
33. Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. Fast hardware architectures for supersingular isogeny Diffie-Hellman key exchange on FPGA. In *Progress in Cryptology – INDOCRYPT 2016: 17th International Conference on Cryptology in India*, pages 191–206, 2016.
34. Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. A high-performance and scalable hardware architecture for isogeny-based cryptography. *IEEE Transactions on Computers*, 67(11):1594–1609, Nov 2018.
35. Brian Koziel, Reza Azarderakhsh, Mehran Mozaffari-Kermani, and David Jao. Post-quantum cryptography on FPGA based on isogenies on elliptic curves. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(1):86–99, Jan 2017.
36. Brian Koziel, Amir Jalali, Reza Azarderakhsh, David Jao, and Mehran Mozaffari-Kermani. NEON-SIDH: Efficient implementation of supersingular isogeny Diffie-Hellman key exchange protocol on ARM. In *Cryptology and Network Security: 15th International Conference, CANS 2016*, pages 88–103, 2016.
37. Patrick Longa, Wen Wang, and Jakub Szefer. The cost to break SIKE: A comparative hardware-based analysis with AES and SHA-3. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 402–431, Cham, 2021. Springer International Publishing.
38. Pedro Maat C Massolino, Patrick Longa, Joost Renes, and Lejla Batina. A compact and scalable hardware/software co-design of SIKE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 245–271, 2020.
39. Michael Naehrig and Joost Renes. Dual isogenies and their application to public-key compression for isogeny-based cryptography. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 243–272, Cham, 2019. Springer International Publishing.
40. Geovandro C. C. F. Pereira and Paulo S. L. M. Barreto. Isogeny-based key compression without pairings. In Juan A. Garay, editor, *Public-Key Cryptography – PKC 2021*, pages 131–154, Cham, 2021. Springer International Publishing.

41. Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145, 2006.
42. Hwajeong Seo, Mila Anastasova, Amir Jalali, and Reza Azarderakhsh. Supersingular isogeny key encapsulation (SIKE) round 2 on ARM Cortex-M4. *IEEE Transactions on Computers*, pages 1–1, 2020.
43. Hwajeong Seo, Pakize Sanal, Amir Jalali, and Reza Azarderakhsh. Optimized implementation of SIKE round 2 on 64-bit ARM Cortex-A processors. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 67-I(8):2659–2671, 2020.
44. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science (FOCS 1994)*, pages 124–134, 1994.
45. Oleg Taraskin, Vladimir Soukharev, David Jao, and Jason T. LeGrow. Towards isogeny-based password-authenticated key establishment. *Journal of Mathematical Cryptology*, 15(1):18–30, 2021.
46. The National Institute of Standards and Technology (NIST). Post-quantum cryptography standardization, 2017–2018. `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization`.
47. Yan Bo Ti. Fault attack on supersingular isogeny cryptosystems. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography*, pages 107–122, Cham, 2017. Springer International Publishing.
48. Jacques Vélu. Isogénies entre courbes elliptiques. *Comptes Rendus de l'Académie des Sciences Paris Séries A-B*, 273:A238–A241, 1971.
49. Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. A post-quantum digital signature scheme based on supersingular isogenies. In *Financial Cryptography and Data Security: 21st International Conference, FC 2017*, pages 163–181, Cham, 2017. Springer International Publishing.