NEON-SIDH: Efficient Implementation of Supersingular Isogeny Diffie-Hellman Key-Exchange Protocol on ARM

Brian Koziel[1], Reza Azarderakhsh[1], Amir Jalali[1], Mehran Mozaffari Kermani[2] and David Jao[3]

[1] Computer Engineering Department
[2] Electrical and Microelectrical Engineering Department
Rochester Institute of Technology, Rochester, NY 14623, USA
{bck650, rxaeec, aj2628@, mmkeme}@rit.edu.
[3]Combinatorics and Optimization, University of Waterloo, CANADA
djao@math.uwaterloo.ca

Abstract. In this paper, we investigate the efficiency of implementing a post-quantum key-exchange protocol over isogenies (PQCrypto 2011) on ARM-powered embedded platforms. We propose to employ new primes to speed up constant-time finite field arithmetic and perform isogenies quickly. Montgomery multiplication and reduction are employed to produce a speedup of 3 over the GNU Multiprecision Library. We analyze the recent projective isogeny formulas presented in [1] and conclude that affine isogeny formulas are much faster in ARM devices. We provide fast affine SIDH libraries over 512, 768, and 1024-bit primes. We provide timing results for emerging embedded ARM platforms using the ARMv7A architecture for 85-, 128-, and 170-bit quantum security levels. Our assembly-optimized arithmetic cuts the computation time for the protocol by 50% in comparison to our portable C implementation and performs approximately 3 times faster than the only other ARMv7 results found in the literature. The goal of this paper is to show that isogeny-based cryptosystems can be implemented further and be used as an alternative to classical cryptosystems on embedded devices.

Keywords: Elliptic curve cryptography, post-quantum cryptography, isogeny-based cryptosystems, ARM embedded processors, finite field, assembly

1 Introduction

Post-quantum cryptography (PQC) refers to research on cryptographic primitives (usually public-key cryptosystems) that are not efficiently breakable using quantum computers more than classical computer architectures. Notably, Shor's algorithm [2] can be efficiently performed with a quantum computer to break standard Elliptic Curve Cryptography (ECC) and RSA cryptosystems. There are some alternatives to be secure against quantum computer threats like the McEliece cryptosystem, lattice-based cryptosystems, code-based cryptosystems, multivariate public key cryptography, and the like. Recently, in [3], [4], [5], and [6], efficient implementations of quantum-safe cryptosystems have been implemented on embedded systems. None of these works consider making the current cryptosystems based on elliptic curves to be quantum-resistant and hence they introduce and implement new cryptosystems with different performance metrics.

To avoid quantum computing concerns, an elliptic curve based alternative to Elliptic Curve Diffie-Hellman (ECDH) which is not susceptible to Shor's attack is the Supersingular Isogeny Diffie-Hellman (SIDH) key exchange. Isogeny computations construct an algebraic map between elliptic curves, which appear resistant to quantum attacks. Thus, this system improves upon traditional ECC and is a strong candidate for quantum-resistant cryptography [7]. Faster isogeny constructions would speed up such cryptosystems, increase the viability of existing proposals,

and make new designs feasible. In [7], the use of isogenies to create new and existing cryptographic protocols with quantum-resistance is presented. However, their implementations on emerging embedded devices have not been investigated yet. It is expected that the use of mobile devices, such as smartphones, tablets, and emerging embedded systems, will become further widespread in the coming years for increasingly sensitive applications. In this work, we further investigate the applicability of advances in theoretical quantum-resistant algorithms on real-world applications by several efficient implementations on emerging embedded systems. Our goal is to improve the performance of isogeny-based cryptosystems to the point where deployment is practical.

In a recent announcement at PQC 2016 [8], NIST announced a preliminary plan to start the gradual transition to quantum-resistant protocols. As such, there is a tremendous need to discover and implement new proposed methods that are resistant to both classical computers and quantum computers. NIST will evaluate these PQC schemes based on security, speed, size, and tunable parameters. Isogeny-based cryptography provides a suitable replacement for standard ECC or RSA protocols because it provides small key sizes, provides forward secrecy, and has a Diffie-Hellman key-exchange available. A protocol is forward secret if it only has to generate one random public key per session for key agreement. Furthermore, isogeny-based cryptography utilizes standard ECC point multiplication schemes, but take it a step further by computing large isogenies to provide quantum-resistance.

Our contribution:

- We provide efficient libraries for the key-exchange protocol presented in [7] to highly optimized C and ASM.
- We present fast and secure prime candidates for 85-bit, 128-bit, and 170-bit quantum security levels.
- We provide hand-optimized finite field arithmetic computations over various ARM-powered processors to produce constant-time arithmetic that is three times as fast as GMP's.
- We analyze the effectiveness of projective [1] and affine [9] isogeny schemes.
- We provide implementation results for embedded devices running a Cortex-A8 and a Cortex-A15. For the latter, an entire quantum-resistant key-exchange with 85-bit quantum security operates in approximately a tenth of a second. Further, our Cortex-A15 assembly optimized results are 3 times faster than [10].

2 SIDH Protocol

This serves as a quick introduction to elliptic curves and isogenies. For a full mathematical background, we point the reader to [9] for a full explanation of isogenies or [11] for a complete look at elliptic curve theory.

2.1 Isogenies on Elliptic Curves

Isogeny-based cryptography utilizes unique algebraic maps between elliptic curves that satisfy group homomorphism. The idea of isogeny-based cryptography was first introduced by J. Silverman [11] and has been investigated for quantum-resistant cryptography by Jao et al. [7,9]. For two elliptic curves over a finite field to be isogenous, they must have the same number of points [12] and have the same j-invariant. We define an isogeny to be $\phi : E \to E'$ such that ϕ satisfies group homomorphism. The degree of an isogeny, $\deg\{\phi\}$, is its degree as an algebraic map. We are particularly interested in computing isogenies of high degree.

A curve's endomorphism ring is defined as the ring of all isogenies from a curve to itself, under point addition and functional composition. A curve is considered supersingular if this endomorphism ring has $\mathbb{Z}$-rank equal to 4. Supersingular curves can be defined over $\mathbb{F}_{p^2}$ or $\mathbb{F}_p$. Therefore, a common field that includes all isogenous curves is $\mathbb{F}_{p^2}$. Supersingular curves have the property that for every prime $\ell \neq p$, there exist $\ell + 1$ isogenies of degree ℓ from a base curve. An isogeny can be computed over a kernel, κ, such that $\phi : E \to E/\langle\kappa\rangle$ by using Vélu's formulas [13]. By specifying curves with a smooth shape, rational points of smooth order act as a generator for the isogeny, allowing efficient computations of isogenies.

2.2 Computing Large Degree Isogenies

The degree of an isogeny is its degree as an algebraic map. As shown in [14], isogeny computations can be done iteratively. Given an elliptic curve E and a point R of order ℓ^e, we compute $\phi\colon E \to E/\langle R\rangle$ by decomposing ϕ into a chain of degree ℓ isogenies, $\phi = \phi_{e-1} \circ \cdots \circ \phi_0$, as follows. Set $E_0 = E$ and $R_0 = R$, and define

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

Essentially, point additions are used to compute the kernel at each iteration and Vélu's formulas are used to compute ϕ_i and E_{i+1}. An optimal strategy is found based on the least number of walks on the isogeny graph to determine the desired isogeny. Finding an optimal strategy is essentially a non-trivial combinatorial problem that only has to be done once for particular degrees of isogenies. One can refer to [9] for further details calculating an optimal strategy over isogeny maps.

2.3 Key-Exchange Protocol Based on Isogenies

Two parties, Alice and Bob, want to exchange a secret key over an insecure channel in the presence of malicious third-parties. They agree on a prime p of the form $\ell_A^a \ell_B^b \cdot f \pm 1$ where ℓ_A and ℓ_B are small primes, a and b are positive integers, and f is a small cofactor to make the number prime. They define a supersingular elliptic curve, $E(\mathbb{F}_q)$ where $q = p^2$. Lastly, they agree on four points on the curve that form two independent bases. These are a basis $\{P_A, Q_A\}$ of $E[\ell_A^a]$ over $\mathbb{Z}/\ell_A^a\mathbb{Z}$ and a basis $\{P_B, Q_B\}$ of $E[\ell_B^b]$ over $\mathbb{Z}/\ell_B^b\mathbb{Z}$. Essentially, each party takes seemingly random walks in the graphs of isogenies of degree ℓ_A^a and ℓ_B^b to both arrive at a curve with the same j-invariant, similar to a Diffie-Hellman key-exchange. In a graph of supersingular isogenies, the infeasibility to discover a path that connects two particular vertices provides security for this protocol.

Alice chooses two private keys $m_A, n_A \in \mathbb{Z}/\ell_A^a\mathbb{Z}$ with the stipulation that both are not divisible by ℓ_A^a. On the other side, Bob chooses two private keys $m_B, n_B \in \mathbb{Z}/\ell_B^b\mathbb{Z}$, where both private keys are not divisible by ℓ_B^b. From there, the key-exchange protocol can be broken down into two rounds of the following:

1. Compute $R = \langle [m]P + [n]Q\rangle$ for points P, Q.
2. Compute the isogeny $\phi : E \to E/\langle R\rangle$ for a supersingular curve E.
3. Compute the kernels $\phi(P)$ and $\phi(Q)$ for the first round.

The key exchange protocol is shown in Figure 1. For a better illustration, we provide a step-by-step example of this protocol in Section 8.1. Alice performs the double point multiplication with her private keys to obtain a kernel, $R_A = \langle [m_A]P + [n_A]Q\rangle$ and computes an isogeny $\phi_A : E \to E_A = E/\langle [m_A]P + [n_A]Q\rangle$. She performs the isogeny quickly by performing many small isogenies of size ℓ_A. She then computes the projection $\{\phi_A(P_B), \phi_A(Q_B)\} \subset E_A$ of the basis $\{P_B, Q_B\}$for

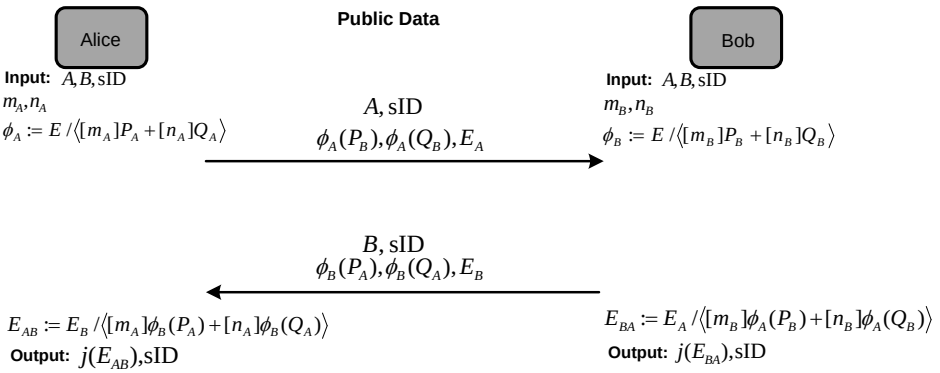

Fig. 1. (a) Key-exchange protocol using isogenies on supersingular curves. Note that $E[\ell_A^a] = \langle P_B, Q_B \rangle$ and $E[\ell_B^b] = \langle P_A, Q_A \rangle$. The secret data is $R_A = m_A P_A + n_A Q_A$ and $R_B = m_B P_B + n_B Q_B$. The public data is $E/\langle R_A \rangle, \phi_A(P_B), \phi_A(Q_B)$ and $E/\langle R_B \rangle, \phi_B(Q_A), \phi_B(Q_A)$.

$E[\ell_B^b]$ under her secret isogeny ϕ_A, which can be done efficiently by pushing the points P_B and Q_B through the isogeny at each smaller isogeny. Over a public channel, she sends these points and curve E_A to Bob. Likewise, Bob performs his own double-point multiplication and computes his isogeny over the supersingular curve E with $\phi_B : E \to E_B = E/\langle [m_B]P + [n_B]Q \rangle$. He also computes his projection $\{\phi_B(P_A), \phi_B(Q_A)\} \subset E_B$ of the basis $\{P_A, Q_A\}$for $E[\ell_A^a]$ under his secret isogeny ϕ_B and sends these points and curve E_B to Alice. For the second round, Alice performs the double point multiplication to find a second kernel, $R_{AB} = \langle [m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A) \rangle$, to compute a second isogeny $\phi'_A : E_B \to E_{AB} = E_B/\langle [m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A) \rangle$. Bob also performs a double point multiplication and computes a second isogeny $\phi'_B : E_A \to E_{BA} = E_A/\langle [m_B]\phi_A(P_B) + [n_B]\phi_A(Q_B) \rangle$. Alice and Bob now have isogenous curves and can use the common j-invariant as a shared secret key.

$$
\begin{aligned}
E_{AB} &= \phi'_B(\phi_A(E)) = \phi'_A(\phi_B(E)) = \\
&= E/\{[m_A]P_A + [n_A]Q_A, [m_B]P_B + [n_B]Q_B\}, \\
&j(E_{AB}) \equiv j(E_{BA}).
\end{aligned}
$$

2.4 Protocol Optimizations

There are many optimizations that have been proposed in [9] and [1]. Notably, all arithmetic is on Montgomery curves as they have fast scalar point multiplication and fast isogeny formulas. Furthermore, the optimal path to compute large-degree isogenies involves finding an optimal strategy of point multiplications and isogeny evaluations. The general trend has been to use isogeny graphs of base 2 and 3, since fast isogenies between Montgomery curves and fast scalar point multiplications can be performed over these isogeny graphs.

Our implementation style closely follows the methods of [9]. We use a 3-point Montgomery differential ladder (also presented in [9]) for a constant set of operations for double point multiplcations and their "affine" isogeny formulas for computing and evaluating large degree isogenies. We note that [9] does not scale the Z-coordinates of the inputs to the ladder to 1. This would decrease the cost of a 3-point step by 2 multiplications per step. [1] recently proposed "projective" isogeny formulas that represent the curve coefficients of a Montgomery curve in projective space (i.e. a numerator and denominator), so that isogeny calculations do not need inversion

until the very end of a round of a key exchange. We also note that [1] proposes sending over isogenies evaluated over points P, Q, and PQ to the other party and that isogenies of a base size 4 are faster than isogenies of base size 2.

3 Proposed Choice of SIDH-Friendly Primes

The primes used in the key-exchange protocol are the foundation of the underlying arithmetic. Since supersingular curves are used, it is necessary to generate primes to allow the curve to have smooth order so that the isogenies can be computed quickly. For this purpose, primes of the form $p = \ell_A^a \ell_B^b \cdot f \pm 1$ are selected. Within that group of primes, [9] and [1] specifically chose isogeny-based cryptosystem parameters of $\ell_A = 2$ and $\ell_B = 3$. These isogeny graph bases provides efficient formulas for isogenies of size 2 and 3, as shown in [9] and [1].

Smooth isogeny primes do not feature the distinct shape of a Mersenne prime (e.g. $2^{521} - 1$) or pseudo-Mersenne prime, but the choice of $\ell_A = 2$ does provide for several optimizations to finite-field arithmetic, which will be covered in more detail in Section 4.

The security of the underlying isogeny-based cryptosystem is directly related to the relative size of ℓ_A^a and ℓ_B^b, or rather $\min(\ell_A^a, \ell_B^b)$. Whichever isogeny graph spanned by the prime is smaller is easier to attack. Therefore, a prime must be chosen where these isogeny graphs are approximately equal. The classical security of the prime is approximately its size in bits divided by 4 and quantum security of a prime is approximately its size in bits divided by 6. In terms of a prime's smooth isogeny graphs, the classical security is thus the size of the smaller isogeny graph divided by 2 and the quantum security is thus the size of the smaller isogeny graph divided by 3. Based on this security analysis, the SIDH protocol over a 512-bit, 768-bit, and 1024-bit prime feature approximately 85, 128, and 170 bits of quantum security, respectively.

3.1 Proposed Prime Search

Primes were searched for by setting balanced isogeny orders ℓ_A^a and ℓ_B^b for $\ell_A = 2$ and $\ell_B = 3$ and searching for factors f that produce a prime ± 1. However, using $+1$ in the form of the prime produced a prime where $-1 \bmod p$ was a quadratic residue. This was a result of using $\ell_A = 2$ as the first isogeny graph, which is non-optimal for the extension field $\mathbb{F}_{p^2}$. Thus, primes of the form $p = 2^a 3^b \cdot f - 1$ were primarily investigated. The primes were found by using a Sage script that changes f to find such primes. The prime number theorem in arithmetic progressions in [15] holds that the density of such primes is sufficient. We did not search for primes with an f value greater than 100. The primes that we discovered were compared and selected based on the following parameters:

- **Security**: The relative security of SIDH over a prime is based on $\min(\ell_A^a, \ell_B^b)$. Therefore, the prime should have balanced isogeny graphs and a small f term.
- **Size**: These primes are designed to be used in ARM processors, some that are limited in memory. These primes should feature a size slightly less than a power of 2, while featuring a high quantum security.
- **Speed**: These primes efficiently use space to reduce the number of operations per field arithmetic, but also have nice properties for the field arithmetic. Notably, all primes of the form $p = 2^a \ell_B^b \cdot f - 1$ will have the Montgomery friendly property [16] because the least significant half of the prime will have all bits set to '1'.

Table 1 contains a list of strong primes for 512, 768, and 1024-bit SIDH implementations. Each of these primes feature approximately balanced isogeny graphs. Each prime requires the least

Table 1. Proposed smooth isogeny primes

Security Level	Prime Size (bits)	$p = \ell_A^a \ell_B^b \cdot f \pm 1$	$\min(\ell_A^a, \ell_B^b)$	Classical Security	Quantum Security
p_{512}	499	$2^{251}3^{155}5 - 1$	3^{155}	123	82
	503	$2^{250}3^{159} - 1$	2^{250}	125	83
	510	$2^{252}3^{159}37 - 1$	2^{252}	126	84
p_{768}	751	$2^{372}3^{239} - 1$	2^{372}	186	124
	758	$2^{378}3^{237}17 - 1$	3^{237}	188	125
	766	$2^{382}3^{238}79 - 1$	3^{238}	189	126
p_{1024}	980	$2^{493}3^{307} - 1$	3^{307}	243	162
	1004	$2^{499}3^{315}49 - 1$	2^{499}	249	166
	1008	$2^{501}3^{316}41 - 1$	3^{316}	250	167
	1019	$2^{508}3^{319}35 - 1$	3^{319}	253	168

number of total bits for a quantum security level. We provide a prime with the f term to be 1 for each security level, but that is not a requirement.

We provide several primes within each security level to give tunable parameters for an SIDH implementation. Extra bits of freedom in the radix representation allow for speed optimizations such as lazy reduction and carry cancelling. [1] proposes using the prime $2^{372}3^{239}-1$ for a 768-bit implementation. This prime is actually 751 bits, allowing for 17 bits of freedom. However, as Table 1 shows, the prime $2^{378}3^{237}17 - 1$ is a 758-bit prime that gives 1 more bit of quantum security. 10 bits of freedom leaves plenty of room for optimizations, so it is merely a question of how many bits of freedom are necessary, while still achieving a high security level.

Lastly, we note that it is better to have a larger isogeny graph of 2^a, since this ensures that the chain of '1's on the end of the prime is longer. This could result in an additional register that is all '1's for reduction and it could also be used to generate a faster addition chain for exponentiations. Regular form improves computations and larger isogeny graphs of 2^a allows for more form in a prime.

For our design, we chose to implement over the primes:

$$\begin{aligned} p_{512} &= 2^{250}3^{159} - 1 \\ p_{768} &= 2^{372}3^{239} - 1 \\ p_{1024} &= 2^{501}3^{316}41 - 1 \end{aligned}$$

4 Proposed Finite-Field Arithmetic

For any cryptosystem featuring large finite-fields, the finite-field arithmetic lies at the heart of the computations. This work is no exception. The critical operations are finite-field addition, squaring, multiplication, and inversion. The abundance of the operations throughout the entire key-exchange protocol calls for numerous optimizations to the arithmetic, even at the assembly level. This work targets the ARMv7-A architectures. All operations are done in the Montgomery domain to take advantage of the extremely fast Montgomery reduction for the primes above.

Finite-field addition performs $A + B = C$, where $A, B, C \in \mathbb{F}_p$. Essentially, this just means that there is a regular addition of elements A and B to produce a third element C. If $C \geq p$, then

$C = C - p$. For ARMv7, this can be efficiently done by using the *ldmia* and *stmia* instructions, which load and store multiple registers at a time, incrementing the address each time. The operands are loaded into multiple registers and added with the carry bit. If the resulting value is larger than the prime for a field, then a subsequent subtraction by the prime occurs. For a constant-time implementation, the conditional flags are used to alter a mask that is applied to the prime as the subtraction occurs. In the case that the value is not larger than the prime, the masked prime becomes 0. Finite-field subtraction is nearly identical to addition, but subtraction with borrow is used and if the borrow flag is set at the end of the subtraction, then the prime is added to the resulting value.

Introduced in [1], the smooth isogeny form $2^a\ell^b f - 1$ features a fast reduction based on simplifying the Montgomery reduction formula [17]:

$$c = (a + (aM' \bmod R)p)/R = (a - aM' \bmod R)/R + ((p+1)(aM' \bmod R))$$

where m is slightly larger than the size of the prime (e.g. $R = 2^{512}$ for p_{512}), a is a result of a multiplication and less than $2m$ bits long, $M' = -p^{-1} \bmod 2^m$, and c is $a \bmod p$. In this equation, $p+1$ has many least-signficant limbs of '0', since approximately half of the least-significant limbs of p are all '1'. Thus, many partial products can be avoided for reduction over this scheme. An alternative to the above scheme is to leave the Montgomery reduction in its standard form, but perform the first several partial products as subtractions since $0\text{xFF} \times A = A \times 2^8 - A$ and the least significant limbs are all '1'.

The typical scheme for Montgomery multiplication is to use $M' = -p^{-1} \bmod 2^w$, where w is the word size. We note that the form of the prime $2^a\ell^b f - 1$ guarantees that $p' = 1$ as long as $2^a > 2^{64}$, for our ARMv7 implementation. This reduces the complexity of Montgomery reduction from $k^2 + k$ to k^2 single-precision multiplication operations, where k is the number of words of an element within the field that must be multiplied.

We utilize the ARM-NEON vector unit to perform the multiplications because it can hold many more registers and parallelize the multiplications. We adopt the multiplication and squaring scheme of [18] to perform large multiplications efficiently. This scheme utilizes a transpose of individual registers within NEON to reduce data dependency stalls. This same technique was employed in this work to perform the multiplication for 512-bit multiplication with the Cascade Operand Scanning (COS) method. In general, multiplications are performed as 32×256 bit multiplications, which is applied several times to produce a 512×512 multiplication. A separated reduction scheme was used. 1024-bit multiplication are composed of three 512×512 multiplications, based on a 1 level additive Karatsuba multiplication. Squaring can reuse the input operands and several partial products for multiplication and reduce approximately 75% of the cycles for a multiplication.

Finite-field inversion finds some A^{-1} such that $A \cdot A^{-1} = 1$, where $A, A^{-1} \in \mathbb{F}_p$. There are many schemes to perform this efficiently. Fermat's little theorem exponentiates $A^{-1} = A^{p-2}$. This requires many multiplications and squarings, but is a constant set of operations. The Extended Euclidean Algorithm (EEA) has a significantly lower time complexity of $O(\log^2 n)$ compared to $O(\log^3 n)$ for Fermat's little theorem. EEA uses a greatest common divisor algorithm to compute the modular inverse of elements a and b with respect to each other, $ax + by = \gcd(a, b)$. With the abundance of field inversions used in this key-exchange protocol, the EEA was chosen. The GMP library already employs a highly optimized version of EEA for various architectures. The algorithm performs the inversion very quickly, but does leak some information about the value being inverted from the timing information. Therefore, to take advantage of this algorithm and provide some protections against simple power analysis and timing attacks, a random value was multiplied to the element before and after the inversion, effectively obscuring what value was initially being inverted. This requires two extra multiplications, but the additional

Table 2. Affine isogeny formulas vs. projective isogenies formulas

Computation	Affine Cost [9]	Projective Cost [1]
Mult-by-3	$7\tilde{M} + 4\tilde{S}$	$8\tilde{M} + 5\tilde{S}$
Iso-3 Computation	$1\tilde{I} + 5\tilde{M} + 1\tilde{S}$	$3\tilde{M} + 3\tilde{S}$
Iso-3 Evaluation	$4\tilde{M} + 2\tilde{S}$	$6\tilde{M} + 2\tilde{S}$
Mult-by-4	$6\tilde{M} + \tilde{S}$	$8\tilde{M} + 4\tilde{S}$
Iso-4 Computation	$1\tilde{I} + 3\tilde{M}$	$5\tilde{S}$
Iso-4 Evaluation	$6\tilde{M} + 4\tilde{S}$	$9\tilde{M} + 1\tilde{S}$

defense against timing and simple power analysis attacks is absolutely necessary for a secure key-exchange protocol.

Since isogenies can be defined over $\mathbb{F}_{p^2}$, a reduction modulus must be defined to simplify the multiplication between elements of $\mathbb{F}_{p^2}$. With the prime choice of $p = 2^a \ell_B^b \cdot f - 1$, -1 is never a quadratic residue of the prime and $x^2 + 1$ can be used as a modulus for the extension field. Addition/subtraction in $\mathbb{F}_{p^2}$ require 2 additions in $\mathbb{F}_p$, squaring in $\mathbb{F}_{p^2}$ requires 2 multiplications and 3 additions in $\mathbb{F}_p$, multiplicaiton in $\mathbb{F}_{p^2}$ requires 3 multiplications and 5 additions in $\mathbb{F}_p$, and inversion in $\mathbb{F}_{p^2}$ requires an inversion, 2 multiplications, 2 squarings, and 2 additions in $\mathbb{F}_p$. This arithmetic required 3 temporary registers in $\mathbb{F}_p$. Two extra multiplications by a random value were added to finite-field inversion to provide side-channel resistance. An inversion over the Montgomery domain also has an additional multiplication ($\text{MM}(\frac{1}{ar}, r^3) = \frac{r}{a}$) in $\mathbb{F}_p$ to keep the result in the Montgomery domain.

5 Affine or Projective Isogenies

Here, we analyze the complexity of utilizing the new "projective" isogeny formulas presented by Costello et al. in [1] to the "affine" isogeny formulas presented by De Feo et al. in [9]. Notably, the projective formulas allow for constant-time inversion implementations without greatly increasing the total time of the protocol. However, in terms of non-constant inversion, we will show that the affine isogeny formulas are still much faster for ARMv7 target device. For cost comparison between these formulas, let I, M, and S refer to inversion, multiplication, and squaring in $\mathbb{F}_p$, respectively. A tilde above the letter indicates that the operation is in $\mathbb{F}_{p^2}$.

We introduce the idea of the inversion/multiplication ratio, or for SIDH over $\mathbb{F}_{p^2}$, $\tilde{I}/\tilde{M}$. This is dependent on the size of elements in $\mathbb{F}_p$, the processor, as well as the inversion used. For a constant-time inversion, the ratio is most likely several hundred since it requires several hundred multiplications and squarings for an exponentiation based on Fermat's little theorem. However, for non-constant time inversion, such as EEA or Kaliski's almost inverse, the ratio is much smaller. For instance, on ARMv7 platforms, as in [19], the ratio ranges from approximately 10 for a 254-bit number to approximately 7 for a 638-bit number, both over $\mathbb{F}_p$. For personal computers, the ratio is much larger for non-constant inversion, typically greater than 20 for optimized arithmetic.

In Table 2 we compare the relative computational costs of affine isogeny formulas and projective isogeny formulas. Evaluations and point multiplications should be reduced as much as possible as they are performed many times when computing large degree isogenies. Isogeny computations compute the map between two points. Affine isogeny computations cost more than their projective counterpart because certain calculations are performed that are reused across each affine isogeny evaluation.

Table 3. Relative costs of traversing isogeny pyramid based on affine vs. projective isogenies

Prime	$\#3P$	#3eval	#3comp	LargeIso3Cost	$\#4P$	#4eval	#4comp	LargeIso4Cost
Affine Isogeny Computations								
p_{512}	496	698	159	$159\tilde{I} + 9417\tilde{M}$	457	410	124	$124\tilde{I} + 6966\tilde{M}$
p_{768}	780	1176	239	$239\tilde{I} + 15163\tilde{M}$	771	638	185	$185\tilde{I} + 11215\tilde{M}$
p_{1024}	1123	1568	316	$316\tilde{I} + 21005\tilde{M}$	1061	942	250	$250\tilde{I} + 15974\tilde{M}$
Projective Isogeny Computations								
p_{512}	500	691	159	$11525\tilde{M}$	423	441	124	$9182\tilde{M}$
p_{768}	811	1124	239	$18623\tilde{M}$	638	771	185	$14865\tilde{M}$
p_{1024}	1129	1558	316	$25792\tilde{M}$	981	1013	250	$21076\tilde{M}$

Table 4. Comparison of break-even inversion/multiplication ratios for large isogenies at different security levels. When the inversion over multiplication ratio is at the break-even point, affine isogenies require approximately the same cost as projective isogenies. Ratios smaller than these numbers are faster with affine formulas. The tilde indicates an operation in $\mathbb{F}_{p^2}$ and no tilde indicates an operation in $\mathbb{F}_p$.

Prime	Alice Round 1 Iso	Bob Round 1 Iso	Alice Round 2 Iso	Bob Round 2 Iso
p_{512}	$\tilde{I} = 20.87\tilde{M}$	$\tilde{I} = 19.26\tilde{M}$	$\tilde{I} = 17.87\tilde{M}$	$\tilde{I} = 13.26\tilde{M}$
p_{768}	$\tilde{I} = 22.73\tilde{M}$	$\tilde{I} = 20.48\tilde{M}$	$\tilde{I} = 19.73\tilde{M}$	$\tilde{I} = 14.48\tilde{M}$
p_{1024}	$\tilde{I} = 23.41\tilde{M}$	$\tilde{I} = 21.15\tilde{M}$	$\tilde{I} = 20.41\tilde{M}$	$\tilde{I} = 15.15\tilde{M}$
p_{512}	$I = 52.62M$	$I = 47.78M$	$I = 43.62M$	$I = 29.78M$
p_{768}	$I = 58.20M$	$I = 51.44M$	$I = 49.20M$	$I = 33.46M$
p_{1024}	$I = 60.23M$	$I = 53.46M$	$I = 51.23M$	$I = 35.46M$

From this table, we created optimal strategies for traversal of the isogeny pyramid. The affine and projective optimal strategy differed because the ratio of point multiplication over isogeny evaluation differed. Similar to the method proposed by [9] and also implemented in [1], we created an optimal strategy to traverse the pyramid. We based the cost of traversing the pyramid with the relationship $\tilde{S} = 0.66\tilde{M}$, since there are 2 multiplications in $\mathbb{F}_p$ for $\tilde{S}$ and 3 multiplications in $\mathbb{F}_p$ for $\tilde{M}$. We performed this experiment for our selected primes in the 512-bit, 768-bit, and 1024-bit categories, shown in Table 3.

We note that the difference in performance is also much greater for the first round of the SIDH protocol, as the other party's basis points are pushed through the isogeny mapping. This includes 3 additional isogeny evaluations per isogeny evaluation, as P, Q, and $P-Q$ are pushed through the isogeny. In Table 4, we compare the break-even points for when the cost of affine and projective isogenies are the same. Alice operates over base 4 isogenies and Bob operates over base 3 isogenies. We utilize $\tilde{I} = I + 3.33\tilde{M}$ for a Karatsuba-based inversion, $I/M = 3(\tilde{I}/\tilde{M} - 3.33)$. For example, for Alice's round 1 isogeny, $I = 53M$ is the break-even point at the 512-bit level. Thus, even with conservative estimates for the cost of using projective coordinates, affine isogenies trump projective coordinates for small I/M ratios.

6 Implementation Results and Discussion

In this section, we review the ARM architectures that were used as testing platforms, how we optimized the assembly code around them, and present our results.

6.1 ARM Architectures

As the name Advanced RISC Machines implies, ARM implements architectures that feature simple instruction execution. The architectures have evolved over the years, but this work will focus on the ARMv7-A. The ARMv7-A family employs a 32-bit architecture that uses 16 general-purpose registers, although registers 13, 14, and 15 are reserved for the stack pointer, link register, and program counter, respectively. ARM-NEON is an Single-Instruction Multiple-Data (SIMD) engine that provides vector instructions for the above architecture. ARMv7's NEON features 32 registers that are 64-bits wide or alternatively viewed as 16 registers that are 128-bits wide. NEON provides nice speedups over standard register approaches by taking advantage of data paralellism in the large register sizes. This comes in handy primarily in multiplication, squaring, and reduction.

We benchmarked the following boards running various ARM architectures:

- A BeagleBoard Black running a single ARMv7 Cortex-A8 processor operating at 1.0 GHz. Conforming to the ARMv7-A architecture, this processor is among the more basic architectures within the family, supporting a decode width of 2, a pipeline depth of 13, a split Harvard L1 cache with 32 KB each, and an L2 cache of 512 KB.
- A Jetson TK1 running 4 ARMv7 Cortex-A15 cores operating at 2.3 GHz. This processor provides more performance than the Cortex-A8 with a decode width of 3, multiple depths of pipeline, and out-of-order execution. The L1 cache is the same as the Cortex-A8, but the L2 cache is shared among cores with up to 8 MB per chip.

We utilize loop unrolling, instruction re-ordering, register allocation, and multiple stores to hand-optimize our assembly used for finite-field arithmetic in the above boards.

6.2 Testing Methodology

The key-exchange was written in the standard C language. We used GMP version 6.1.0. The code was compiled using the standard operating system and development environment on the given device. A parameters file defining the agreed upon curve, basis points, and strategies for the key exchange was generated externally using Sage. The strictly C code with GMP is fairly portable and can be used with primes of any size, as long as it is provided with a valid parameters file. There are separate versions which include the 512-bit and 1024-bit assembly optimizations that only work with primes up to these sizes. The protocols are identical in both the C and ASM implementations. The timings were found by taking the median of a sufficient number of trials for each cell in the table. The primes that were used can be found in Table 3. Some constants critical to the computation were precomputed in the Montgomery domain.

6.3 Results and Comparison

The results for this experiment are presented in Table 6. This provides the timings of individual finite field operations in $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ as well as the total computation time of each party for the protocol. The expected time to run this protocol is roughly Alice or Bob's computation time and some transmission cost.

The Jetson TK1 achieved a speedup of 1.94 when using the hand-optimized assembly for 512-bit primes and a speedup of 1.59 for 1024-bit primes. The Beagle Board Black achieved a speedup of 2.27 over the 512-bit primes and a speedup of 2.00 over 1024-bit primes when using the assembly code. These speedups came as a result of the optimized finite field arithmetic over $\mathbb{F}_p$. Addition is generally a fraction of the cost. Multiplication and squaring are almost twice as fast

Table 5. Timing results of key-exchange on Beagle Board Black ARMv7 device for different security levels

Beagle Board Black (ARM v7) Cortex-A8 at 1.0 GHz using C												
Field	$\mathbb{F}_p$ [cc]						$\mathbb{F}_{p^2}$ [cc]				Key Exchange [cc $\times 10^3$]	
Size	A	S	M	R	I	I/M	$\tilde{A}$	$\tilde{S}$	$\tilde{M}$	$\tilde{I}$	Alice	Bob
p_{512}	115	1866	2295	3429	40100	7.0	1241	12229	14896	72400	483,968	514,786
p_{768}	142	3652	4779	6325	71500	6.4	1404	23167	28459	135400	1,406,381	1,525,215
p_{1024}	168	5925	8202	10150	111900	6.1	1558	38046	46891	211400	3,135,526	3,367,448
Beagle Board Black (ARM v7) Cortex-A8 at 1.0 GHz using ASM and NEON												
Field	$\mathbb{F}_p$ [cc]						$\mathbb{F}_{p^2}$ [cc]				Key Exchange [cc $\times 10^3$]	
Size	A	S	M	R	I	I/M	$\tilde{A}$	$\tilde{S}$	$\tilde{M}$	$\tilde{I}$	Alice	Bob
p_{512}	70	718	953	962	40100	20.9	279	4445	6736	52756	216,503	229,206
p_{1024}	120	2714	3723	3956	111900	14.6	375	15714	23682	150795	1,597,504	1,708,383

Table 6. Timing results of key-exchange on NVIDIA Jetson TK-1 ARMv7 device for different security levels

Jetson TK-1 Board (ARM v7) Cortex-A15 at 2.3 GHz using C												
Field	$\mathbb{F}_p$ [cc]						$\mathbb{F}_{p^2}$ [cc]				Key Exchange [cc $\times 10^3$]	
Size	A	S	M	R	I	I/M	$\tilde{A}$	$\tilde{S}$	$\tilde{M}$	$\tilde{I}$	Alice	Bob
p_{512}	83	926	1152	2271	24302	7.1	877	7256	8776	42481	285,026	302,332
p_{768}	99	1679	2403	4024	39100	6.1	982	13467	16216	73922	783,303	848,461
p_{1024}	117	2955	4144	6053	59800	5.7	1122	21558	26286	115437	1,728,183	1,851,782
Jetson TK-1 Board (ARM v7) Cortex-A15 at 2.3 GHz using ASM and NEON												
Field	$\mathbb{F}_p$ [cc]						$\mathbb{F}_{p^2}$ [cc]				Key Exchange [cc $\times 10^3$]	
Size	A	S	M	R	I	I/M	$\tilde{A}$	$\tilde{S}$	$\tilde{M}$	$\tilde{I}$	Alice	Bob
p_{512}	39	516	640	732	24302	17.7	158	3025	4579	34049	148,003	154,657
p_{1024}	73	1856	2464	2961	59800	11.0	273	11273	17007	97594	1,118,644	1,140,626

with the ASM. The most significant improvement is reduction around 3-3.5 times as fast with the ASM. Addition in $\mathbb{F}_{p^2}$ is approximately 5-7 times faster with assembly because the intermediate elements were guaranteed to be in the field, only requiring a subtraction with a mask as a modulus. With the assembly optimizations, the Jetson TK1 performs one party's computations in approximately 0.066 seconds and 0.491 seconds over 85-bit and 170-bit quantum security, respectively. The Beagle Board Black performs one party's computations in approximately 0.223 seconds and 1.65 seconds over 85-bit and 170-bit quantum security, respectively.

This implementation follows the algorithms and formulas of the key exchange protocol is in [9]. We note that [10] utilizes assembly-optimized arithmetic over Barrett reduction and reduces the computation time of [9] by approximately 25%, whereas our optimizations reduce the computation time by 50% over the standard C implementation. Further, we also stress that our implementation includes side-channel resistance. Our finite-field arithmetic is constant-time, except for inversion which applies extra multiplications for protection, and we utilize a constant set of operations that deal with the secret keys. Lastly, our implementation is much more portable because it only requires a C compiler and the GNU library.

The only other implementations of SIDH for ARMv7 are [10] and [1]. [1] only operates with projective isogeny formulas over the 751-bit prime, $2^{372}3^{239} - 1$, and uses a generic implementation with Montgomery reduction. [10] uses the same affine formulas as our implementation,

Table 7. Comparison of affine and projective isogeny implementations on ARM Cortex-A15 embedded processors. Our work and [1] was done on a Jetson TK1 and [10] was performed on an Arndale ARM Cortex-A15.

Work	Language	Field size	Quantum Security	Isogeny formulas	Timings $[cc \times 10^6]$				
					Alice R1	Bob R1	Alice R2	Bob R2	Total
Costello et al. [1][1]	C	751	124	Proj.	1,794	2,120	1,665	2,001	7,580
Azarderakhsh et al. [10]	C	521	85	Affine	N/A	N/A	N/A	N/A	1,069
	C	771	128		N/A	N/A	N/A	N/A	3,009
	C	1035	170		N/A	N/A	N/A	N/A	6,477
This work	ASM	503	83	Affine	83	87	66	68	302
	C	751	124		437	474	346	375	1,632
	ASM	1008	167		603	657	516	484	2,259

1. Targeted x86-64 architectures, but is portable on ARM. All arithmetic is in generic C.

Table 8. Comparison of I/M ratios for various computer architectures based on GMP library

Architecture	Device	I/M ratio		
		p_{512}	p_{768}	p_{1024}
ARMv7 Cortex-A8	Beagle Board Black	7.0	6.4	6.1
ARMv7 Cortex-A15	Jetson TK1	7.1	6.1	5.9
ARMv8 Cortex-A53	Linaro HiKey	8.2	7.3	6.5
Haswell x86-64	i7-4790k	14.9	14.7	13.8

but uses primes that are not as efficient. Table 7 contains a comparison of these implementations for ARM Cortex-A15. We note that the assembly optimizations are not applied for our 768-bit version. Similarly, [1] has a generic implementation with Montgomery reduction. Our assembly optimized implementation is approximately 3 times faster than the implementation in [10] and the portable C implementation is about 5 times faster than the projective isogeny implementation in [1].

We were surprised to find that the performance of the SIDH library in [1] suffered on ARMv7, so we investigated this further. Table 8 compares the I/M ratio for different computer architectures over GMP computations. We note that with optimized multiplication, this would generally be higher, but it is an idea of the relative difference between I/M ratios for ARM architectures and x86 architectures. As Table 8 shows, the I/M ratio for a PC is much greater than ARM architectures, by a factor of 2. This shows that ARM implementations benefit much more from using affine coordinates ($I < 20M, \tilde{I} < 8\tilde{M}$ with our optimizations). We primarily attribute the slowness experienced by the SIDH library to the generic arithmetic and low I/M ratios for ARMv7 devices.

There are several other popular post-quantum cryptosystems that have been implemented in the literature. The ones that consider embedded system have typically used FPGA's or 8-bit microcontrollers, such as the lattice-based system in [4], code-based system in [6], or McEllice system in [3] and [5]. The comparison with any of these works is difficult because the algorithms are extremely different and the implementations did not use ARM-powered embedded devices.

7 Conclusion

In this paper, we proved that isogeny-based key exchanges proposed in [7] can be implemented efficiently on emerging ARM embedded devices and represent a new alternative to classical

cryptosystems. Both efficient primes and the impact of projective isogeny formulas were investigated. Without transmission overhead, a party can compute their side of the key exchange in fractions of a second. We hope that the initial investigation of this protocol on embedded devices will inspire other researchers to continue looking into isogeny-based implementations as a strong candidate for NIST's call for post-quantum resistant cryptosystems. As a future work, we plan to apply our assembly optimizations to the projective isogeny formulas presented in [1] for a constant-time implementation. We note that robust and high-performance implementations provide critical support for industry adoption of isogeny-based cryptosystems.

References

1. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny diffie-hellman. Cryptology ePrint Archive, Report 2016/413 (2016)
2. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science (FOCS 1994). 124–134 (1994)
3. Eisenbarth, T., Güneysu, T., Heyse, S., Paar, C.: Microeliece: Mceliece for embedded devices. In Clavier, C., Gaj, K., eds.: 11th International Workshop Cryptographic Hardware and Embedded Systems - CHES 2009. Volume 5747 of Lecture Notes in Computer Science., Springer 49–64 (2009)
4. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: A signature scheme for embedded systems. In Prouff, E., Schaumont, P., eds.: 14th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2012. Volume 7428 of Lecture Notes in Computer Science., Springer 530–547 (2012)
5. Heyse, S.: Implementation of mceliece based on quasi-dyadic goppa codes for embedded devices. In Yang, B.Y., ed.: 4th International Workshop on Post-Quantum Cryptography, PQCrypto 2011. Volume 7071 of Lecture Notes in Computer Science., Springer 143–162 (2011)
6. Heyse, S., von Maurich, I., Güneysu, T.: Smaller keys for code-based cryptography: Qc-mdpc mceliece implementations on embedded devices. In Bertoni, G., Coron, J.S., eds.: 15th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2013. Volume 8086 of Lecture Notes in Computer Science., Springer 273–292 (2013)
7. Jao, D., Feo, L.D.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Post-Quantum Cryptography–PQCrypto 2011. Volume 7071 of LNCS. 19–34 (2011)
8. Chen, L., Jordan, S.: Report on post-quantum cryptography, (2016)
9. De Feo, L., Jao, D., Plut, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Journal of Mathematical Cryptology 8(3), 209–247 (September 2014)
10. Reza Azarderakhsh, Dieter Fishbein, D.J.: Efficient implementations of a quantum-resistant key-exchange protocol on embedded systems. Technical report, University of Waterloo (2014)
11. Silverman, J.H.: The Arithmetic of Elliptic Curves. Volume 106 of GTM. Springer, New York (1992)
12. Tate, J.: Endomorphisms of abelian varieties over finite fields. Inventiones Mathematicae 2, 134–144 (1966)
13. Vélu, J.: Isogénies entre courbes elliptiques. Comptes Rendus de l'Académie des Sciences Paris Séries A-B 273, A238–A241 (1971)
14. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006)
15. Lagarias, J., Odlyzko, A.: Effective versions of the Chebotarev density theorem. In: Algebraic number fields: L-functions and Galois properties. Symposium Proceedings of the University of Durham 409–464 (1975)
16. Gueron, S., Krasnov, V.: Fast prime field elliptic-curve cryptography with 256-bit primes. Journal of Cryptographic Engineering 5(2), 141–151 (2014)
17. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44(170), 519–521 (1985)
18. Seo, H., Liu, Z., Grobschadl, J., Kim, H.: Efficient arithmetic on arm-neon and its application for high-speed rsa implementation. Cryptology ePrint Archive, Report 2015/465 (2015) http://eprint.iacr.org/.

19. Grewal, G., Azarderakhsh, R., Longa, P., Hu, S., Jao, D.: Efficient Implementation of Bilinear Pairings on ARM Processors. In: Selected Areas in Cryptography: 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg (2013) 149–165

8 Appendix

8.1 Sample Key Exchange

For reference, we provide a toy example of a key-exchange under the supersingular isogeny Diffie-Hellman. Assume Alice and Bob agree on parameters $\ell_A = 2$, $\ell_B = 3$, $e_A = 22$, $e_B = 15$, and $f = 1$, for a prime $p = 2^{22}3^{15} - 1$. For reference, $F.1$ will refer to the most signficant element in $\mathbb{F}_{p^2}$. Alice and Bob also choose to use the Montgomery curve $y^2 = x^3 + (46990356857824 \times F.1 + 51574400922824)x^2 + x$, with cardinality $(p+1)^2 = 362207510070449535829929984$. We note that we push the opposite party's basis points P and Q through projective isogenies, but the same could be done with Kummer isogenies of the Kummer coordinates P, Q, and PQ. Withe the above curve, both parties determine the following coordinates for bases A and B:

$$P_A = (16435726748426 \times F.1 + 25857855444783 \ : \ 3311090673801 \times F.1 + 49904885532715 \ : \ 22924753146307 \times F.1 + 4141333668847)$$

$$Q_A = (42745493793169 \times F.1 + 25640910747935 \ : \ 26024857998918 \times F.1 + 45400075813831 \ : \ 33869819798953 \times F.1 + 27450566627723)$$

$$P_B = (46241491872403 \times F.1 + 24929181970748 \ : \ 13155975969868 \times F.1 + 35327236081211 \ : \ 24346083709932 \times F.1 + 23936773895502)$$

$$Q_B = (37509538200603 \times F.1 + 24025255492771 \ : \ 11657749987300 \times F.1 + 34486050347794 \ : \ 14039740366919 \times F.1 + 57274374824856)$$

Alice picks secret keys $m_A = 1$, $n_A = 12072833$ and Bob picks secret keys $m_B = 21430664$, $n_B = 1$. Alice and Bob use these secret keys compute a double point multiplication to generate a kernel over Montgomery Kummer lines:

$$R_A = (32854839462681 \times F.1 + 26587109381295 \ : \ 47388368863397 \times F.1 + 39769792436883)$$

$$R_B = (42919642960505 \times F.1 + 54910988865013 \ : \ 56563006858480 \times F.1 + 34171044701950)$$

Alice computes an isogeny over E_0 with her kernel and applies points P_B and Q_B over the isogeny:

$$E_A \ : \ (17031078681609 \times F.1 + 14923000136871)y^2 = x^3 + (5281450620597 \times F.1 + 2844666389140)x^2 + x$$

$$\phi_A(P_B) = (52979143687140 \times F.1 + 11666207223688 \ : \ 23042001578843 \times F.1 + 53541680755863 \ : \ 56030208016172 \times F.1 + 59571693043722)$$

$$\phi_A(Q_B) = (7059088422233 \times F.1 + 8723030024031 \ : \ 11278511729527 \times F.1 + 29422098398745 \ : \ 59190951474805 \times F.1 + 40212020037679)$$

Bob also computes an isogeny over E_0 with his kernel and applies points P_A and Q_A over the isogeny:

$$E_B \ : \ (53963752958850 \times F.1 + 33110507945965)y^2 = x^3 + (39099266670887 \times F.1 + 48127419907737)x^2 + x$$

$$\phi_B(P_A) = (29159200437206 \times F.1 + 10075659528496 \ : \ 33517540157349 \times F.1 + 41752911486261 \ : \ 45911701422116 \times F.1 + 14451873864646)$$

$$\phi_B(Q_A) = (21760272334158 \times F.1 + 34388982665251 \ : \ 22055074323229 \times F.1 + 9308750125658 \ : \ 44090484346682 \times F.1 + 25501363801352)$$

Alice and Bob exchange this information over an insecure channel. They perform Round 2 containing a second double point multiplication and isogeny over the other party's transferred information:

$$R_{AB} = (48372798995204 \times F.1 + 49380570296163 \ : \ 11901326566357 \times F.1 + 46036001374066)$$

$$R_{BA} = (39806120200001 \times F.1 + 32797858325871 \ : \ 21820883729659 \times F.1 + 53933942074447)$$

$$E_{AB} \ : \ (15444749931456 \times F.1 + 50948818889259)y^2 = x^3 + (8068193289649 \times F.1 + 1323974738625)x^2 + x$$

$$E_{BA} \ : \ (15444749931456 \times F.1 + 50948818889259)y^2 = x^3 + (8068193289649 \times F.1 + 1323974738625)x^2 + x$$

In this case, the curve each party found was the same, but applying the isogenies in a different way could produce curves that differ but are still isomorphic. In any case, the agreed upon private key is the j-invariant of the curve:

$$j(E_{AB}) = j(E_{BA}) = 34563663181277 \times F.1 + 2312423318543$$