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

Brian Koziel[1(✉)], Amir Jalali[2], Reza Azarderakhsh[3], David Jao[4], and Mehran Mozaffari-Kermani[5]

[1] Texas Instruments, Wylie, USA
kozielbrian@gmail.com
[2] CE Department, RIT, Rochester, USA
amirjalali65@gmail.com
[3] CEECS Department and I-SENSE FAU, Boca Raton, USA
razarderakhsh@fau.edu
[4] C&O Department, University of Waterloo, Waterloo, Canada
djao@uwaterloo.ca
[5] EME Department, RIT, Rochester, USA
mmkeme@rit.edu

Abstract. We investigate the efficiency of implementing the Jao and De Feo isogeny-based post-quantum key exchange protocol (from PQCrypto 2011) on ARM-powered embedded platforms. In this work we propose 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 Costello et al. (Crypto 2016) 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 the 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 arithmetic · Assembly implementation

1 Introduction

Post-quantum cryptography (PQC) refers to research on cryptographic primitives (usually public-key cryptosystems) that are not efficiently breakable using

S. Foresti and G. Persiano (Eds.): CANS 2016, LNCS 10052, pp. 88–103, 2016.
DOI: 10.1007/978-3-319-48965-0_6

quantum computers. Most notably, Shor's algorithm [1] can be efficiently implemented on a quantum computer to break standard Elliptic Curve Cryptography (ECC) and RSA cryptosystems. There are some alternatives secure against quantum computing threats, such as the McEliece cryptosystem, lattice-based cryptosystems, code-based cryptosystems, multivariate public key cryptography, and others. Recent work such as [2–4] demonstrates efficient implementations of such quantum-safe cryptosystems on embedded systems. None of these works consider an approach based on quantum-resistant elliptic curve cryptosystems. Hence, they introduce and implement new cryptosystems with different security metrics and performance characteristics.

To avoid quantum computing attacks, Jao and De Feo [5] proposed an elliptic curve based alternative to Elliptic Curve Diffie-Hellman (ECDH) which is not susceptible to Shor's attack, namely the Supersingular Isogeny Diffie-Hellman (SIDH) key exchange protocol. Isogeny computations constitute an algebraic map between elliptic curves, which appear to be resistant to quantum attacks. Thus, this system improves upon traditional ECC and represents a strong candidate for quantum-resistant cryptography. Faster isogeny constructions would speed up such cryptosystems, increase the viability of existing proposals, and make new designs feasible. Existing results on the implementation of isogeny-based key exchange include De Feo et al. [5,6] and Costello et al. [7]. However, implementations on emerging embedded devices have not been fully investigated. It is expected that mobile devices, such as smartphones, tablets, and emerging embedded systems, will become more widespread in the coming years for increasingly sensitive applications. In this work, we investigate the applicability of advances in theoretical quantum-resistant algorithms on real-world applications by providing 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-like key exchange available. Furthermore, key compression schemes have been proposed in [7,9] to aid in the storage and transmission of ephemeral keys. Lastly, isogeny-based cryptography utilizes standard ECC point multiplication schemes, allowing for re-use of existing ECC libraries and even hybrid schemes that simultaneously use ECC and isogenies to provide quantum resistance, such as the hybrid scheme proposed in [7].

Our contributions:

- We provide efficient libraries[1] for the key exchange protocol presented in [5] using 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 3 times as fast as GMP.
- We analyze the effectiveness of projective [7] and affine [6] isogeny computation schemes.
- We provide implementation results for embedded devices running Cortex-A8 and 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], the fastest results available in the literature.

2 SIDH Protocol

This serves as a quick introduction to the Supersingular Isogeny Diffie-Hellman key exchange. For a full mathematical background of the protocol, we point the reader to the original works proposing it in [5,6] or [11] for a complete look at elliptic curve theory.

2.1 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 smooth isogeny 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, $E_0(\mathbb{F}_q)$ where $q = p^2$. Lastly, they agree on four points on the curve that form two independent bases. Over a starting supersingular curve E_0, these are a basis $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ which generate $E_0[\ell_A^{e_A}]$ and $E_0[\ell_B^{e_B}]$, respectively, such that $\langle P_A, Q_A \rangle = E_0[\ell_A^{e_A}]$ and $\langle P_B, Q_B \rangle = E_0[\ell_B^{e_B}]$.

As first noted in [12], consider a graph of all supersingular elliptic curves of a fixed isogeny graph under $\mathbb{F}_{p^2}$. In this graph, the vertices represent each isomorphism class of supersingular elliptic curves and the edges represent the degree-ℓ isogenies of a particular isomorphism class. Essentially, each party takes seemingly random walks in the graph of supersingular isogenies of degree ℓ_A^a and ℓ_B^b to both arrive at supersingular elliptic curves with the same isomorphism class and 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.

[1] Code is available at https://github.com/kozielbrian/NEON-SIDH_ARMv7.

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 images $\phi(P)$ and $\phi(Q)$ for the basis of the opposite party for the first round.

The key exchange protocol proceeds as follows. 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 $\phi_A : E_0 \to E_A = E_0/\langle [m_A]P + [n_A]Q \rangle$. She performs the large degree isogeny efficiently by performing many small isogenies of degree 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 $E_0[\ell_B^b]$ under her secret isogeny ϕ_A, which can be done efficiently by pushing the points P_B and Q_B through each isogeny of degree ℓ_A. 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 $\phi_B : E_0 \to E_B = E_0/\langle [m_B]P + [n_B]Q \rangle$. He also computes his projection $\{\phi_B(P_A), \phi_B(Q_A)\} \subset E_B$ of the basis $\{P_A, Q_A\}$ for $E_0[\ell_A^a]$ 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.

$$\begin{aligned} E_{AB} &= \phi'_B(\phi_A(E_0)) = \phi'_A(\phi_B(E_0)) \\ &= E_0/\{[m_A]P_A + [n_A]Q_A, [m_B]P_B + [n_B]Q_B\}, \\ j(E_{AB}) &\equiv j(E_{BA}). \end{aligned}$$

2.2 Protocol Optimizations

Many optimizations have been proposed in [6,7] for computing isogenies. To begin with, all arithmetic is performed on Montgomery curves [13] as they have been shown to have fast scalar point multiplication and fast isogeny formulas. We refer to the Explicit Formulas Database (EFD) [14] for the fastest operation counts on elliptic curves. The Kummer representation for Montgomery curves provides extremely fast curve arithmetic by performing operations on the curve's Kummer line [13]. Points are represented as $(X : Z)$, where $x = X/Z$. Under this scheme, there is no difference between points P and $-P$. The EFD provides explicit formulas for differential addition and point doubling. Note that P and $-P$ generate the same subgroup of points on the elliptic curve, so isogenies can be evaluated correctly on the Kummer line. Lastly, 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 [6]. We use a 3-point Montgomery differential ladder (also presented in [6]) 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 [6] 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. An alternative approach to the double-point multiplication is to utilize a uniform double-point multiplication algorithm, such as those proposed in [15] or [16]. Costello et al. [7] 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 [7] proposes sending isogenies evaluated over the points P, Q, and PQ in Kummer coordinates to the other party in the first round and that isogenies of degree 4 have been shown to be faster than isogenies of degree 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, smooth isogeny primes of the form $p = \ell_A^a \ell_B^b \cdot f \pm 1$ are selected. Within that group of primes, [6,7] specifically chose isogeny-based cryptosystem parameters of $\ell_A = 2$ and $\ell_B = 3$. These isogeny graph bases provides efficient formulas for isogenies of degree 2 and 3, as shown in [6,7].

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, covered in more detail in Sect. 4.

The security of the underlying isogeny-based cryptosystem is directly related to the relative magnitude of ℓ_A^a and ℓ_B^b, or rather $\min(\ell_A^a, \ell_B^b)$. Whichever isogeny graph is spanned by the smaller prime power is easier to attack. Therefore, a prime should be chosen where these prime powers are approximately equal. As demonstrated in [6], 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. Based on this security assessment, 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

We searched for primes 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 produces a prime where −1 mod p is a quadratic residue, which is not optimal for performing arithmetic in the extension field $\mathbb{F}_{p^2}$. Therefore, we primarily investigated only primes of the form $p = 2^a 3^b \cdot f - 1$. Our primes were found by using a Sage script that changes f to find such primes. 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 to allow for some speed optimizations such as lazy reduction and carry cancelling, while still 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 [17] because the least significant half of the prime will have all bits set to '1'.

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

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

We provide several primes within each security level to give tunable parameters for an SIDH implementation. Costello et al. [7] propose 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 for speed optimizations in systems using 32 or 64-bit

words. 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 and still has 10 bits of freedom to allow for speed optimizations. We find it useful to have several strong primes to work with, which could allow for a variety of speed techniques.

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

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

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 these 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 [18] to take advantage of the extremely fast Montgomery reduction for the primes above.

All arithmetic below is for $\mathbb{F}_p$. Since supersingular curves 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. We utilized reduced arithmetic in $\mathbb{F}_{p^2}$ based on fast arithmetic in $\mathbb{F}_p$.

4.1 Field Addition

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.

4.2 Field Multiplication and Squaring

Finite-field multiplication performs $A \times B = C$, where $A, B, C \in \mathbb{F}_p$. This equates to a regular multiplication of A and B to produce a third element C. However,

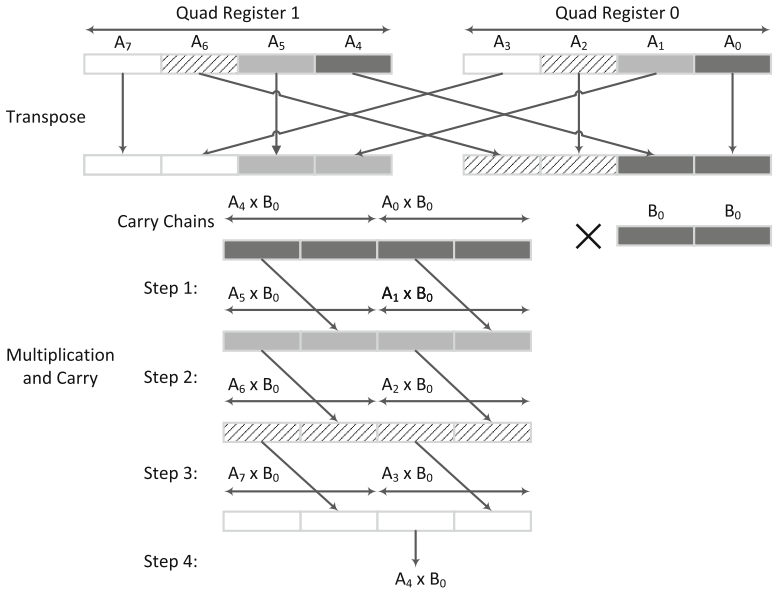

Fig. 1. Finite-field Multiplication using NEON

if elements A and B are both m-bits, then the result, C, is $2m$-bits. A reduction must be made so that the result is still within the field. Montgomery multiplication and reduction [18] was chosen because of its fast reduction method. Introduced in [7], smooth isogeny primes of the form $2^a\ell^b f - 1$ feature a fast reduction based on simplifying the Montgomery reduction formula [18]:

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

where $R = 2^m$ 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, $M' = -p^{-1} \bmod R$, and $c = a \bmod p$. 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 $M' = 1$ 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 [19] 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, as shown in Fig. 1. By using a transposed quad register in NEON, the partial products can be determined out of order and the carries applied later, reducing data dependencies in the multiplication sequence. Figure 1 demonstrates an example of a 32×256 bit multiplication, which is applied several times to produce a 512×512 multiplication. A separated reduction scheme was used. A 1024-bit multiplication is 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 requires approximately 75 % of the cycles for a multiplication.

4.3 Field Inversion

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, $ax + by = \gcd(a, b)$. Based on the analysis presented in Sect. 5, the EEA was chosen because it made affine SIDH much faster than projective SIDH. The GMP library already employs a highly optimized version of EEA for various architectures. EEA performs an inversion quickly, but does leak some information about the value being inverted from the timing information. Therefore, to take advantage of this fast inversion 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 defense against timing and simple power analysis attacks is necessary for a secure key exchange protocol.

5 Affine or Projective Isogenies

Here, we analyze the complexity of utilizing the new "projective" isogeny formulas presented by Costello et al. in [7] to the "affine" isogeny formulas presented by De Feo et al. in [6]. 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 devices. 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 $\mathbb{F}_{p^2}$.

Table 2. 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

Table 3. Affine isogeny formulas vs. projective isogenies formulas

Computation	Affine cost [6]	Projective cost [7]
Point 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	$4\tilde{M} + 2\tilde{S}$	$6\tilde{M} + 2\tilde{S}$
Point 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	$6\tilde{M} + 4\tilde{S}$	$9\tilde{M} + 1\tilde{S}$

We introduce the idea of the inversion/multiplication ratio, or for SIDH over $\mathbb{F}_{p^2}$, $\tilde{I}/\tilde{M}$, as a metric to compare the relative cost of inversion and multiplication and decide between the effectiveness of affine or projective formulas. This inversion/multiplication ratio is dependent on the size of elements in $\mathbb{F}_p$, the processor, as well as the inversion used. For a constant-time inversion using Fermat's little theorem, the ratio is most likely several hundred since it requires several hundred multiplications and squarings for the inversion exponentiation. However, for non-constant time inversion, such as EEA or Kaliski's almost inverse [20], the ratio is much smaller. Table 2 compares the I/M ratio for different computer architectures over the GNU Multiprecision Library (GMP). We note that with optimized multiplication, this ratio would generally be higher, but it gives an idea of the relative difference between I/M ratios for ARM architectures and x86 architectures. As Table 2 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 isogeny computations.

In Table 3, we compare the relative computational costs of affine isogeny formulas presented in [6] and projective isogeny formulas presented in [7] over isogenies of degree 3 and 4. Point multiplications by ℓ are over Kummer coordinates with affine or projective curve coefficients. Isogeny computations compute the map between two points and isogeny evaluations push a point through the mapping, both of these are of degree ℓ. Affine isogeny computations cost more than their projective counterpart because certain calculations are performed that are reused across each affine isogeny evaluation.

Table 4. Relative costs of computing large-degree isogenies based on affine vs. projective isogeny formulas

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	$21076\tilde{M}$

From this table, we created optimal strategies for traversing the large-degree isogeny graphs. The affine and projective optimal strategy differed because the ratio of point multiplication over isogeny evaluation differed. Similar to the method proposed by [6] and also implemented in [7], we created an optimal strategy to traverse the graph. We based the cost of traversing the graph 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 4. In Table 4, we count the total number of point multiplications by ℓ as $\#\ell P$, the total number of $\ell-$isogeny evaluations as $\#\ell$eval, and the total number of $\ell-$isogeny computations as $\#\ell$comp. From the cost of these operations in affine or projective coordinates, shown in Table 3, we calculated the total cost of the large-degree isogeny in terms of multiplications and inversions in $\mathbb{F}_{p^2}$ under LargeIsoℓCost.

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 computation, as P, Q, and $P - Q$ are pushed through the isogeny. In Table 5, we compare the break-even points for when the cost of affine and projective isogenies are the same. If the ratio is smaller than the break-even point, then the large-degree isogeny computation is faster with affine isogeny formulas. Alice operates over degree 4 isogenies and Bob operates over degree 3 isogenies. We utilize $\tilde{I} = I + 3.33\tilde{M}$ to get the break-even points for operations in $\mathbb{F}_p$ since we used a Karatsuba-based inversion. Thus, $I/M = 3(\tilde{I}/\tilde{M} - 3.33)$. As an example, the break-even point for Alice's round 1 isogeny is $I = 53M$ at the 512-bit level. Thus, even with conservative estimates for the cost of using projective coordinates, affine coordinates trump projective coordinates for small I/M ratios.

Table 5. Comparison of break-even inversion/multiplication ratios for large-degree 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.

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}	$I = 60.23M$	$I = 53.46M$	$I = 51.23M$	$I = 35.46M$

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 a Single-Instruction Multiple-Data (SIMD) engine that provides vector instructions for the ARMv7 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 these ARM architectures:

- A BeagleBoard Black running a single ARMv7 Cortex-A8 processor operating at 1.0 GHz.
- A Jetson TK1 running 4 ARMv7 Cortex-A15 cores operating at 2.3 GHz.

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 primes that were used can be found in Table 3.

6.3 Results and Comparison

The results for this experiment are presented in Tables 6 and 7 for the BeagleBoard Black and Jetson TK1, respectively. This provides the timings, in clock cycles, 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 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 our hand-optimized assembly code over our generic C code. The Jetson TK1 achieved a speedup of 1.94 for 512-bit primes and a speedup of 1.59 for 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 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 Beagle Board Black performs one party's computations in approximately 0.223 s and 1.65 s over 85-bit and 170-bit quantum security, respectively. The Jetson TK1 performs one party's computations in approximately 0.066 s and 0.491 s over 85-bit and 170-bit quantum security, respectively.

Table 6. 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 size	$\mathbb{F}_p$ [cc]					$\mathbb{F}_{p^2}$ [cc]				Key Exch. [cc $\times 10^3$]	
	A	S	M	mod	I	$\tilde{A}$	$\tilde{S}$	$\tilde{M}$	$\tilde{I}$	Alice	Bob
p_{512}	115	1866	2295	3429	40100	1241	12229	14896	72400	483,968	514,786
p_{768}	142	3652	4779	6325	71500	1404	23167	28459	135400	1,406,381	1,525,215
p_{1024}	168	5925	8202	10150	111900	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 size	$\mathbb{F}_p$ [cc]					$\mathbb{F}_{p^2}$ [cc]				Key Exch. [cc $\times 10^3$]	
	A	S	M	mod	I	$\tilde{A}$	$\tilde{S}$	$\tilde{M}$	$\tilde{I}$	Alice	Bob
p_{512}	70	718	953	962	40100	279	4445	6736	52756	216,503	229,206
p_{1024}	120	2714	3723	3956	111900	375	15714	23682	150795	1,597,504	1,708,383

Table 7. 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 size	$\mathbb{F}_p$ [cc]					$\mathbb{F}_{p^2}$ [cc]				Key Exch. [cc $\times 10^3$]	
	A	S	M	mod	I	$\tilde{A}$	$\tilde{S}$	$\tilde{M}$	$\tilde{I}$	Alice	Bob
p_{512}	83	926	1152	2271	24302	877	7256	8776	42481	285,026	302,332
p_{768}	99	1679	2403	4024	39100	982	13467	16216	73922	783,303	848,461
p_{1024}	117	2955	4144	6053	59800	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 size	$\mathbb{F}_p$ [cc]					$\mathbb{F}_{p^2}$ [cc]				Key Exch. [cc $\times 10^3$]	
	A	S	M	mod	I	$\tilde{A}$	$\tilde{S}$	$\tilde{M}$	$\tilde{I}$	Alice	Bob
p_{512}	39	516	640	732	24302	158	3025	4579	34049	148,003	154,657
p_{1024}	73	1856	2464	2961	59800	273	11273	17007	97594	1,118,644	1,140,626

Our implementation follows the algorithms and formulas of the affine key exchange protocol given in [6]. Our implementation also 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 C implementation is portable because it only requires a C compiler and the GNU library.

The only other portable implementations of SIDH for ARMv7 are [7,10]. Of these, [7] only operates with projective isogeny formulas over the 751-bit prime, $2^{372}3^{239}-1$, and uses a generic, constant-time, implementation with Montgomery reduction. [10] uses the same affine formulas as our implementation, but uses primes that are not as efficient. Table 8 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, [7] has generic arithmetic 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 [7].

Table 8. Comparison of affine and projective isogeny implementations on ARM Cortex-A15 embedded processors. Our work and [7] was done on a Jetson TK1 and [10] was performed on an Arndale ARM Cortex-A15.

Work	Lang	Field size [bits]	PQ Sec. [bits]	Iso. Eq	Timings [$cc \times 10^6$]				
					Alice R1	Bob R1	Alice R2	Bob R2	Total
Costello et al. [7][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.

Moreover, [10] does not consider side-channel attacks, but [7] is a constant-time implementation, which is inherently protected against simple power analysis and timing attacks.

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 [2], code-based system in [4], or McEllice system in [3]. 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 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 investigate redundant arithmetic schemes with NEON and apply our assembly optimizations to the projective isogeny formulas for a constant-time implementation. We note that robust and high-performance implementations provide critical support for industry adoption of isogeny-based cryptosystems.

Acknowledgment. The authors would like to thank the reviewers for their constructive comments. This material is based upon work supported by the National Science Foundation under grant No. CNS-1464118 awarded to Reza Azarderakhsh.

References

1. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science (FOCS 1994), pp. 124–134 (1994)
2. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: a signature scheme for embedded systems. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 530–547. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33027-8_31
3. Heyse, S.: Implementation of Mceliece based on quasi-dyadic goppa codes for embedded devices. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 143–162. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25405-5_10
4. Heyse, S., Maurich, I., Güneysu, T.: Smaller keys for code-based cryptography: QC-MDPC McEliece implementations on embedded devices. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 273–292. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40349-1_16

5. Jao, D., Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25405-5_2
6. De Feo, L., Jao, D., Plut, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. J. Math. Cryptology **8**(3), 209–247 (2014)
7. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 572–601. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53018-4_21
8. Chen, L., Jordan, S.: Report on post-quantum cryptography, NIST IR 8105 (2016)
9. Azarderakhsh, R., Jao, D., Kalach, K., Koziel, B., Leonardi, C.: Key compression for isogeny-based cryptosystems. In: Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography, AsiaPKC 2016, pp. 1–10. ACM, New York (2016)
10. Azarderakhsh, R., Fishbein, D., Jao, D.: 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. GTM, vol. 106. Springer, New York (1992)
12. Mestre, J.F.: La méthode des graphes. Exemples et applications. In: Proceedings of the International Conference on Class Numbers and Fundamental Units of Algebraic Number Fields (Katata, 1986), pp. 217–242. Nagoya Univ., Nagoya (1986)
13. Montgomery, P.: Speeding the pollard and elliptic curve methods of factorization. Math. Comput. **48**, 243–264 (1987)
14. Bernstein, D.J., Lange, T.: Explicit-formulas database (2007). http://www.hyperelliptic.org/EFD/
15. Bernstein, D.J.: Differential addition chains. Technical report (2006). http://cr.yp.to/ecdh/diffchain-20060219.pdf
16. Azarderakhsh, R., Karabina, K.: A new double point multiplication algorithm and its application to binary elliptic curves with endomorphisms. IEEE Trans. Comput. **63**(10), 2614–2619 (2014)
17. Gueron, S., Krasnov, V.: Fast prime field elliptic-curve cryptography with 256-bit primes. J. Cryptograph. Eng. **5**(2), 141–151 (2014)
18. Montgomery, P.L.: Modular multiplication without trial division. Math. Comput. **44**(170), 519–521 (1985)
19. 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)
20. Kaliski, B.S.: The montgomery inverse and its applications. IEEE Trans. Comput. **44**(8), 1064–1065 (1995)