Efficient Implementations of A Quantum-Resistant Key-Exchange Protocol on Embedded systems

Reza Azarderakhsh[1], Dieter Fishbein[2], and David Jao[2]

1. Department of Computer Engineering
Rochester Institute of Technology
Rochester, NY, USA
rxaeec@rit.edu

2. Department of Combinatorics and Optimization
University of Waterloo
Waterloo, ON, CANADA
{djao and dfishbein}@math.uwaterloo.ca

Abstract. Presence of quantum computers is a real threat against the security of currently used public key cryptographic algorithms such as RSA and Elliptic curve cryptography. Isogeny computation on elliptic curves is believed to be difficult even on a quantum computer, and hence isogeny-based protocols represent one of the few truly practical approaches to constructing quantum-resistant cryptosystems. In this paper, we investigate the efficiency of implementing a newly proposed post-quantum key-exchange protocol on PC and ARM-powered embedded platforms. Our implementations on various mass-market emerging embedded devices significantly improve the state-of-the-art of post-quantum cryptographic computations on ARM-powered devices. We provided timing results and compared them to the counterparts available in the literature. For instance, Our timing results on PC platforms are between 18–26% faster than the previous work depending on the security level.

Keywords: Post-quantum cryptography, elliptic curve cryptography, ARM processors, Finite field, Assembly.

1 Introduction

Classical cryptosystems like RSA and Elliptic Curve Cryptography (ECC) [1], [2], which are based on the hardness of factoring and the elliptic curve discrete logarithm problem (ECDLP), respectively, yet unpredictable but possibly will be threatened by quantum computers using Shor's algorithm [3]. Also, it has been widely accepted that relying solely on asymmetric key cryptography dose not guarantee the required security leves in the long term. ECC and RSA are used to protect secure Web pages, encrypted email, and many other types of data. Breaking these would have significant ramifications for electronic privacy and security. Post-quantum cryptography refers to research on cryptographic primitives (usually public-key cryptosystems) that are not efficiently breakable using quantum computers more than classical computer architectures. Therefore, post-quantum secure and practical alternatives are required to replace these cryptosystems. There are some alternatives to be secure against quantum computers threats like the McEliece cryptosystem, Lattice-based cryptosystems, code based cryptosystem, multivariate public key cryptography, and the like. Recently, in [4], [5], [6], and [7], efficient implementations of quantum-safe cryptosystems have been implemented on embedded systems. There is another way of designing quantum-safe cryptosystems based on isogenies which computationally constructs an algebraic map between the curves. In particular, unlike traditional ECC, isogeny computation appears resistant to quantum attacks, and hence such systems are good candidates for quantum-resistant cryptography [8]. Faster isogeny constructions would speed up such cryptosystems, increase the viability of existing proposals, and make new designs feasible. In [8], the use of isogenies in designing, implementing, and analyzing new and existing cryptographic protocols, with particular emphasis on designing and analyzing quantum-resistant cryptosystems is presented. However, their implementations on embedded devices is not 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. As their use increases, more people are using these devices for increasingly sensitive applications such as corporate email, online banking and for the storage of confidential information. As such, the deployment of practical cryptographic protocols for use on mobile devices is of the utmost importance. Since we are faced with the possible development of a large-scale quantum computer in the near future, it is prudent for us to focus our efforts on the deployment of classical protocols that are resistant to attacks from such technology. In this work we present to explore further the applicability of advances in theoretical quantum-resistant algorithms on real-world applications by several efficient implementations on embedded systems. Isogenies over ordinary elliptic curves can be found in subexponential time, meaning that even larger key sizes are required for quantum resistance than previously thought. Our goal is nevertheless to improve the performance of isogeny-based cryptosystems to the point where deployment is practical.

Our contribution: In this paper, we mainly explore the efficiency of implementing recently proposed isogeny-based post-quantum public key cryptography of [8,9] on mobile embedded devices operating on ARM-powered platforms. In [8,9] a quantum-resistant key-exchange protocol proposed by Jao et al. They have proposed a Diffie-Hellman type key-exchange scheme based on computing isogenies between supersingular elliptic curves. The proposed scheme is shown to be quantum-resistant, and the fastest known attacks are exponential time. In this work, we present a practical implementation of the key-exchange protocol suitable for use in mobile (and non-mobile) devices. We have done our implementations on p_{512}, p_{768} and p_{1024} which denote the 512-bit, 768-bit and 1024-bit primes and provide 85 bits, 128 bits, and 170 bits, quantum security levels, respectively. Our implementations are primarily written in C with hand-optimized assembly designed for use with either ARMv7 or x86-64 processors. It uses precomputed public parameters, with all the time-consuming computations offloaded from the device. Compared to the original implementations presented in [9], our code is between 18–26% faster depending on the security level. Moreover, on iOS and Android devices we measured running times around 0.5–1 second for a round of key exchange at the (quantum) 80-bit security level.

The rest of this paper is organized as follows. In Section 2, we review the elliptic curves and isogenies. In Section 3, key exchange protocol is presented. In Section 4, post-quantum algorithms for key exchange protocol is presented. In Section 5, we presented the implementations and timing results are compared to the counterparts available in the literature. Finally, in Section 6 the paper is concluded.

2 Background on Isogeny-Based Cryptography

The idea of isogeny-based cryptography is proposed by J. Silverman [10] and has been investigated for quantum-resistant cryptography by Jao et al. [8,9]. Two elliptic curves over a finite field are isogenous if and only if they have the same number of point [11]. We define an *elliptic curve* over a field K as a projective nonsingular genus-1 algebraic curve E over K together with a distinguished base point ∞ of E defined over K. When the characteristic of K does not equal 2 or 3, which is always the case in this work, one can write E in the form

$$E : y^2 = x^3 + ax + b.$$

Points on an elliptic curve form a group with an efficiently computable group law, with identity element ∞. An elliptic curve E is determined up to isomorphism by its *j-invariant*, defined by

$$j(E) = 1728 \frac{4a^3}{4a^3 + 27b^2}.$$

For any positive integer n, the *n-torsion group* $E[n]$ is defined to be the set of all points P in E defined over the algebraic closure $\overline{K}$ of K such that n times P is the identity:

$$E[n] = \{P \in E(\overline{K}) : nP = \infty\}.$$

As a group, $E[n]$ has $\mathbb{Z}$-rank equal to 2 provided that the characteristic of K does not divide n, and thus when viewed as a module over $\mathbb{Z}/n\mathbb{Z}$ it admits a basis of two elements. Therefore an isogeny

$$\phi \colon E \to E',$$

is defined to be an algebraic map satisfying the property that ϕ is a group homomorphism. The *degree* of ϕ, denoted $\deg \phi$, is its degree as an algebraic map. An isogeny is *separable* if it is separable as an algebraic map. We are interested in separable isogenies defined over finite fields. Assume E and E' are elliptic curves defined over a finite field $\mathbb{F}_q$. In this case, isogenies are determined up to isomorphism by their kernels. Any finite subgroup H of E induces an isogeny $E \to E/H$; conversely, for any isogeny ϕ, the group $\ker \phi$ is a finite subgroup of E. Finite subgroups of E in turn can be specified by identifying a set of generators. Given such a set of generators, the corresponding isogeny can be computed by using Vélu's formulas [12]. Additionally, every isogeny of degree greater than 1 can be factored into a composition of isogenies of prime degree over $\mathbb{F}_q$ [13]. Two curves E and E' are said to be isogenous over $\mathbb{F}_q$ if there exists an isogeny $\phi : E \to E'$ defined over $\mathbb{F}_q$. A theorem of Tate states that E and E' are isogenous over $\mathbb{F}_q$ if and only if the number points on both curves are the same [11]. Let ϕ have degree ℓ. Then ϕ has a dual isogeny $\hat{\phi}$ [10] such that $\phi \circ \hat{\phi} = [\ell]$. The property of being isogenous over $\mathbb{F}_q$ is an equivalence relation on the set of $\mathbb{F}_q$-isomorphism classes of elliptic curves defined over $\mathbb{F}_q$. Thus, we define an isogeny class to be an equivalence class under this equivalence relation. The key-exchange scheme uses isogenies between *supersingular* elliptic curves. An elliptic curve is supersingular if its endomorphism ring (defined as the ring of all isogenies from a curve to itself, under the operations of pointwise addition and functional composition) has $\mathbb{Z}$-rank equal to 4. An elliptic curve is *ordinary* if its endomorphishm ring does not have $\mathbb{Z}$-rank equal to 4 (in this case its endomorphism risk will have $\mathbb{Z}$-rank equal to 1 or 2). Curves in the same isogeny class are either all supersingular or all ordinary.

3 Key Exchange Protocol

Fix 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 some (typically very small) cofactor. Let E be a supersingular elliptic curve defined over $\mathbb{F}_q = \mathbb{F}_{p^2}$. Fix 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 $\mathbb{Z}/\ell_B^b\mathbb{Z}$. All of these parameters are public. The idea of this protocol is a variation of Diffie-Hellman of the commutative diagram shown in Fig. 1, where ϕ and ψ are random walks in the graphs of isogenies of degree ℓ_A andℓ_B respectively. The security of the key exchange is based on the difficulty of finding a path connecting two specified vertices in a graph of supersingular isogenies. We refer the reader to [8,9] for a detailed discussion on the security of the protocol. To have the paper self-contained, the key exchange protocol presented here as follows.

Alice chooses two secret, random elements $m_A, n_A \in_R \mathbb{Z}/\ell_A^a\mathbb{Z}$, not both divisible by ℓ_A, and computes an isogeny $\phi_A \colon E \to E_A$ with kernel $K_A := \langle [m_A]P_A + [n_A]Q_A \rangle$. Alice computes the image $\{\phi_A(P_B), \phi_A(Q_B)\} \subset E_A$ of the basis $\{P_B, Q_B\}$ for $E[\ell_B^b]$ under her secret isogeny ϕ_A. She sends these points to Bob together with E_A. Similarly, Bob selects secret, random elements $m_B, n_B \in_R \mathbb{Z}/\ell_B^b\mathbb{Z}$, not both divisible by ℓ_B and computes an isogeny $\phi_B \colon E \to E_B$ having kernel $K_B := \langle [m_B]P_B + [n_B]Q_B \rangle$. Bob then computes $\{\phi_B(P_A), \phi_B(Q_A)\}$ and sends the values to Alice along with E_B. With this information, Alice computes an isogeny $\phi'_A \colon E_B \to E_{AB}$ having kernel equal to $\{[m_A]\phi_B(P_A), [n_A]\phi_B(Q_A)\}$. Bob proceeds *mutatis mutandis.* Alice and Bob can then use the common j-invariant of

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

as their shared secret key.

4 Algorithmics

4.1 Parameter Generation

The common system parameters for isogeny-based cryptosystem are $\ell_A = 2$ and $\ell_B = 3$, the original implementation performs most of the key exchange protocol in C (as opposed to Cython) making it more efficient. However, even in this case, parts of the key-exchange protocol are done in Cython. Our modifications only concern themselves with the case $\ell_A = 2$ and $\ell_B = 3$ and we assume those parameter values for the remainder of the section. For any fixed choice of a and b one can choose random values of f until one of $p = \ell_A^a \ell_B^b \cdot f + 1$ or $p = \ell_A^a \ell_B^b \cdot f - 1$

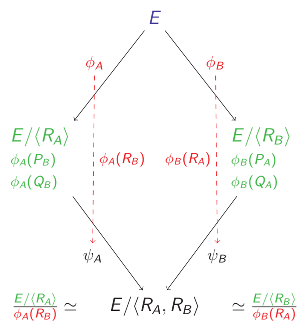

Fig. 1. Key-exchange protocol using isogeneies on supersingular curves. Note that $E[\ell_A^a] = \langle P_B, Q_B\rangle$ and $E[\ell_B^b] = \langle P_A, Q_A\rangle$. Secret data are $R_A = m_A P_A + n_A Q_A$ and $R_B = m_B P_B + n_B Q_B$. Public data are $E/\langle R_A\rangle, \phi_A(P_B), \phi_A(Q_B)$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

is prime. An effective version of the prime number theorem in arithmetic progressions by Lagarias and Odlyzko [14] guarantees that the density of such primes is sufficient. Fixing a prime $p = \ell_A^a \ell_B^b \cdot f \pm 1$ we now need a supersingular curve E_0. A result by Broker [15] recommends that it is computationally easy to find a supersingular curve E over $\mathbb{F}_{p^2}$ with cardinality $(p \mp 1)^2 = (\ell_A^a \ell_B^b \cdot f)^2$. One can either chose $E_0 = E$ or construct the isogeny graph consisting of all supersingular curves defined over $\mathbb{F}_{p^2}$ and choose E_0 via random walks on said isogeny graph. Using either method, we obtain E_0 with group structure $(\mathbb{Z}/(p \mp 1)\mathbb{Z})^2$. To obtain a basis for the torsion group $E_0[\ell_A^a]$, choose a random point $P \in_R E_0(\mathbb{F}_{p^2})$ and set $P' = (\ell_B^b \cdot f)^2 P$ so that P' has order dividing ℓ_A^a. One checks whether P' has order exactly equal to ℓ_A^a by multiplying P' by powers of ℓ_A. If this check succeeds (which it will with high probability) then we set $P_A = P'$. We choose a second point of order ℓ_A^a, Q_A, in the same way. One must check that P_A and Q_A are independent and this is done by computing the Weil pairing $e(P_A, Q_A)$ in $E_0[\ell_A^a]$ and checking that the result has order ℓ_A^a via repeated multiplications of ℓ_A. If this fails we can simply chose another point Q_A and try again.

4.2 Key Exchange

The key exchange is performed in two rounds and in each round Alice and Bob proceed as follows:

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

where E, P, Q, R and S depend on both the round and the player. We now discuss how to implement these three steps.

There are many classical techniques for computing $\langle [m]P + [n]Q\rangle$. In [8], and efficient method for the computation of double point multiplication is presented which is not vulnerable to simple power analysis (SPA) attacks and is employed in this work.

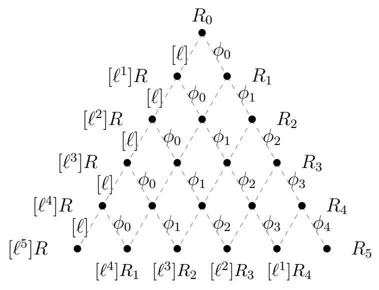

Fig. 2. Diagram of isogeny computation [8].

4.3 Computing Isogenies

Computing the isogenies in the protocol can be accomplished via an iterative process. 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

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

In Fig. 2, the computational structure of computing isogenies for $c = 6$ is illustrated. The bold dots represent points on E. Points on the same left diagonal belong to the same curve and points of the same height on the diagram represent points of the same order. Leftward dashed edges refer to multiplication by ℓ, while rightward dashed edges refer to evaluation of isogenies of degree ℓ. At the beginning of the algorithm, only R_0 is known. In order to compute ϕ, we must compute all the elements at the bottom row of Fig. 2. Using $[\ell^{e-i-1}]R_i$ we can compute the kernel of ϕ_i via $O(\ell)$ point additions. We can then apply Vélu's formulas to compute ϕ_i and E_{i+1}. Since evaluating degree ℓ isogenies is generally twice as expensive as multiplications by ℓ, determining the best approach is a non-trivial combinatorial problem.

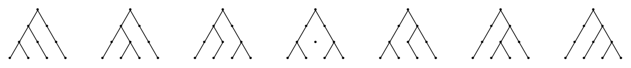

Fig. 3. The seven well-formed full strategies for $n = 4$. Notice that the three middle strategies share the same binary tree topology and the middle one is the canonical strategy.

We are interested in computing the "optimal" full strategy, according to some measure of computational effort. We first note that any well-formed (as shown in Fig. **??**) strategy has a particular binary tree topology obtained by discarding the internal nodes of out-degree less than 2 and preserving the same connectivity structure. Since the optimal strategy can be computed off the device and loaded onto the device as a precomputed parameter, one dose not need to be very concerned with efficiency of the algorithm.[1]

[1] Optimal strategies can be mathematically characterized, though the dynamic programming algorithm is satisfactory from an implementation perspective. See [8,9].

We realized that a straightforward Python implementation computes the optimal strategies for $n = 1024$ in under one second.

4.4 Choice of Models

Or models for the curves and the parameters that have been set in this work are as follow. All the curves employed in this work have group structure of $(\mathbb{Z}/(p \mp 1)\mathbb{Z})^2$. This results to have either the curve itself or its twist to have a point of order 4. In [16], Bernstein et al. prove that any elliptic curve having a point of order 4 is isomorphic to a twisted Edwards curve. They further showed that any twisted Edwards curve is isomorphic to a Montgomery curve [16]. Therefore, over a quadratic extension field, an elliptic curve is isomorphic to its quadratic twist and hence all curves we are concerned with will be isomorphic to a Montgomery curve. To estimate efficiency we count the number of elementary operations in $\mathbb{F}_{p^2}$. We denote I, M, S for the costs of an inversion, a multiplication and a squaring, respectively. We make the assumption that $S \leq M \leq I$. We ignore additions, subtractions and comparisons as they are significantly faster than the operations we include in our estimate. We use the Explicit Formulas Database (EFD) [17] for operation counts on elliptic curves. However, unlike the EFD, we count multiplications by constants (other than small integers) as ordinary multiplications. Montgomery curves [18] have equation as

$$M_{B,A} : By^2 = x^3 + Ax^2 + x. \tag{1}$$

One can represent points on $M_{B,A}$ by coordinates $(X : Z)$ where $x = X/Z$. This is known as a *Kummer* representation. One refers to performing operations on a curve with such a representation as performing operations on the curve's *Kummer line*. Such a representation has the disadvantage of identifying P with $-P$ since the negative of a point only differs in the Y coordinate. However, Montgomery curves have very efficient arithmetic on their Kummer line. The cost of doubling a point is $3M + 2S$ (or $2M + 2S$ when it is scaled to have Z-coordinate equal to 1). Since P is identified with $-P$, it is not possible to add two distinct points. However, if $P - Q$ is known, one can compute $P + Q$ via differential addition. One differential addition has a cost of $4M + 2S$ (or $3M + 2S$ when $P - Q$ is scaled to have Z coordinate equal to 1). Through the use of a Montgomery ladder, along with doublings and differential additions, one can compute any scalar multiplication for a given point P [18]. Since P and $-P$ generate the same subgroup of a group of points on an elliptic curve, and an isogeny can be uniquely identified with such a subgroup, we see that isogenies can be defined and evaluated correctly on the Kummer line.

Isogenies of Montgomery Curves In [8,9], Jao et al. give explicit formulas for isogenies of Montgomery curves and optimize the degree 2 and 3 case. We will not derive those formulas here but instead refer the reader to the original paper for those details. Jao et al.'s original implementation of the key-exchange is suitable when ℓ_A and ℓ_B are arbitrary small primes. However, our implementation is only suitable for $\ell_A = 2$ and $\ell_B = 3$. As in Section 4.3, isogenies of composite smooth degree are computed by composing isogenies of prime degree. In the degree 3^e case, this is done by simply composing degree 3 isogenies. Let $e \geq 3$. In the degree 2^e case, one needs knowledge of a point of order 8 in order to determine the curve F_2 [8,9]. Thus, one cannot simply chain together multiple isogenies of degree 2. This problem is solved by chaining together $e - 2$ isogenies of degree 2 followed by an isogeny of degree 4.

In [8,9], Jao et al. suggested that there may be a small speed advantage in using chains of degree 4-isogenies instead. From Table 1, this certainly appears plausible. However, after further investigation, we found that using 4-isogenies seems to be slightly less efficient in practice ($< 1\%$ disadvantage compared to chains of 2-isogenies). Table 1 compares the cost of isogeny evaluations and scalar multiplications for isogenies of degree 2, 3 and 4. We also report the cost obtained by setting $S = 0.8M$. This figure is roughly based on the fact that squaring in $\mathbb{F}_{p^2}$ requires 2 multiplications (as oppose to 3 for field multiplication). These figures assume certain expressions have been precomputed, and common subexpressions shared. Table 2 lists the total cost of isogeny evaluations and scalar multiplications for an optimal strategy.

Table 1. Comparative costs for multiplication and isogeny evaluation in projective Kummer coordinates, in number of multiplications and squarings, and assuming $S = 0.8M$.

ℓ	2	3	4
Isogenies	$2M + S$	$4M + 2S$	$6M + S$
	2.8	5.6	6.8
Multiplication	$3M + 2S$	$7M + 4S$	$6M + 4S$
	4.6	10.2	9.2

Table 2. Comparative costs of the optimal strategy for computing a degree 2^{514} ($\ell = 2, 4$) or 3^{323} ($\ell = 3$) isogeny, assuming $S = 0.8M$.

ℓ	Optimal Strategy		
	2	3	4
Isogenies	2741	1610	1166
Multiplications	1995	1151	921
Total cost	16852	20756	16402

5 Implementations

In this section, we discuss the techniques that are employed for the implementations of the the isogeny-based quantum resistant protocol presented in the previous sections. Our implementations are done on both ARM powered embedded systems as well as PC platforms.

The original implementation from [8] uses a mixed C/Cython/Python/Sage architecture. Parameter generation is done in Sage, and the computation of the optimal strategy for computing isogenies is done in Python using the dynamic programming algorithm discussed in Section 4.3. Arithmetic in $\mathbb{F}_{p^2}$ is written using C, using GMP to support arithmetic modulo p. Elliptic curve arithmetic is implemented in Cython. In the special case $\ell_A = 2$ and $\ell_B = 3$, the key exchange uses a combination of C and Cython, with the most critical parts done in C. The fact that elliptic curves are implemented using Cython prevents a pure C implementation. For all other values of ℓ_A and ℓ_B, the key exchange is done in Cython.

5.1 Optimizations to the Key-Exchange

In this section we describe the optimizations we made to the key-exchange protocol described in [8,9]. We first describe optimizations made to the C/Cython portion of the program and then discuss our assembly optimizations.

Porting into C The original implementation of the key-exchange has elliptic curves implemented in Cython. Cython is designed to provide an interface between C and Python code so that both can be used in the same software. This allows for the convenience of Python while still implementing critical portions in C. However, a pure C implementation is generally more efficient. We created a C-implementation of elliptic curves and re-wrote the key-exchange in pure C for the $\ell_A = 2$ and $\ell_B = 3$ cases. In the original implementation, public parameters are generated immediately before the key-exchange is executed. This is impractical as these computations are done in Sage and are time-consuming. Furthermore, it is not necessary to have new public parameters for each run of the key-exchange. We separated parameter generation from the actual execution of the key-exchange. Parameter generation is done in Sage and the output is written to a text file. This allows parameter generation to be done on a different device than the key-exchange protocol. The pure C implementation of the key-exchange takes the text file as input.

Assembly Optimizations Assembly languages are low-level programming languages in which there is a strong correspondence between the language and the architecture's machine code instructions. Our ARM assembly code modifications are designed to work with ARMv7 instruction set with a 32-bit word size. The *word size* refers to the register size. The ARMv7 instruction set has 16 registers `r0,... ,r15`. The registers `r13`, `r14` and `r15` are referred to as the stack pointer, link register and program counter, respectively. They have each have a specific purpose and should not used for general-purpose calculations by the user. However, registers `r0` to `r12` can be used freely by the user. Similarly, our x86-64 assembly code modifications are designed to work with the x86-64 architecture. This architecture offers sixteen 64-bit registers `r8,...,r15, rax, rcx, rdx, rbx, rsp, rbp, rsi` and `rdi`. The registers `rbp` and `rsp` are called the base pointer and stack pointer respectively. They should not be used for general-purpose calculations by the user.

We used assembly code to speedup multi-precision integer arithmetic in $\mathbb{F}_q = \mathbb{F}_{p^2}$. Multi-precision integers are integers that are larger than a given machine's word-size. The majority of programming languages do not provide direct support for multi-precision numbers. One generally needs to use a specific software package, such as GMP, in order to use them efficiently. However, if one knows the approximate size of the number in advance, one can sometimes design hand-optimized assembly routines that are more efficient than routines in these software packages. The disadvantage of this approach is that hand-optimized assembly code is platform-specific and non-portable. We employ the following techniques to optimize our assembly implementation:

- **Loop Unrolling:** Since we know the maximal size in bits of the operands, we can unroll all loops. This gives us the ability to avoid conditional branches.
- **Instruction re-ordering:** Often times instructions can compete for the same processor and data resources, causing the code to be slower. By re-ordering non-dependent instructions we can allow for a more optimal scheduling of the instructions in the microprocessor's pipeline. For example, for integer multiplication, loop unrolling makes it possible to load the data required for the next multiplication while the processor is performing the current one.
- **Register Allocation:** All available registers were used in order to minimize moving data from memory to registers.
- **Multiple stores:** ARM processors are capable of loading and storing multiple words from or to the memory by one instruction. By storing the final result at once instead of writing a word back to memory each time when a new result is ready, we minimize the number of memory access instructions. Also, we do some register clean-ups (cost-free) when the pipeline is performing the multiple store instruction. It is worth mentioning that while it was possible to write 8 words at once, only 4 words are written to memory at each time because the available non-dependent instructions to re-order after the multiple store instruction are limited.

The key-exchange software uses GMP as an arithmetic backend. GMP stores numbers in consecutive memory locations and uses the *little endian* method which stores the least-significant word at the smallest memory address. We implemented field addition in ARMv7 assembly language and implemented both field addition and field multiplication in x86-64 assembly language. Field multiplication for $\mathbb{F}_q$ requires several additions, 3 integer multiplications and two modular reductions for which Barrett reduction was used. We present details of these implementations below. We will not attempt to explain each assembly instruction in detail but rather try to present the idea of each algorithm we discuss.

Field Addition: 512-bit $\mathbb{F}_q$ addition was implemented on ARMv7 platform and 768-bit $\mathbb{F}_q$ addition was implemented on the x86-64 platform. When passing three or fewer parameters to a function, the function will place those parameters in registers `r1` $-$ `r3` and expect to receive any possible output at the memory address in register `r0`. Our implementation of 768-bit field addition on the x86-64 platform is much simpler. It was found that is was just as efficient to use the built-in GMP function `mpz_add` to add the two numbers rather than using an assembly routine. This is likely due to `mpz_add` being coded in assembly for the ARMv7 platform. After adding the two numbers, we then check to see if a subtraction is required by using `mpz_cmp` to compare the size of the result to the prime order of the underlying field. If a subtraction is required, it is done using an assembly routine to subtract the prime from the result. This routine essentially consists of the `SUB` instruction followed by several `SBB` instructions to subtract the correct pieces of the prime from the result and adjust for any borrow flags.

Integer Multiplication: 768-bit field multiplication was implemented on the x86-64 platform. The technique we use is known as *column-wise multiplication* and is analogous to the method school-book multiplication taught to young children. First, the MUL instruction takes one register operand and multiplies the value of that operand by the value of the number in register rax. It stores the double-word result of this multiplication in the registers rax (lowest word) and rdx (highest word). Second, register labels are preceded by the % character. Third the MOV instruction moves data between different data-storage locations. MOV is much like ARMv7's LDR and STR instructions.

Barrett Reduction: 768-bit Barrett reduction was implemented on the x86-64 platform for use in $\mathbb{F}_q$ multiplication. The algorithm is implemented using a combination of C and assembly with the aspects that control the flow of the algorithm implemented in C and the arithmetic portions implemented in assembly. In 1986, P. D. Barrett introduced Barrett reduction to compute $c = a \pmod n$ [19]. Barrett is an improvement on naive division algorithms and works assuming that $a < n^2$. The main idea is to replace expensive divisions by multiplications that can be performed much cheaper. We use Barrett reduction to compute $c = a \pmod p$ where $\mathbb{F}_q = \mathbb{F}_{p^2}$. We are able to make several pre-computations at the parameter generation stage before the key-exchange is executed and stored in the same text file containing the public parameters for the encryption system. First we compute the minimal k such that $2^k > p$ and then m such that $m = \lfloor 4^k/p \rfloor$.

5.2 Implementation Results and Comparisons

The original implementations of the computations of post-quantum cryptography presented in [8], is only for PC platforms for 85, 128, and 170 bits security levels. The timing results for our pure C implementation are presented in Table 3. Notice that our pure C implementation is approximately 20% faster than the original implementations on the Mac OS platform. Due to the variety of software packages required, the original implementations presented in [8] was not suitable to run on the iOS or Android platform. Thus, a comparison of running times cannot be made on these platforms, even though our implementations support them.

Let p_{512}, p_{768} and p_{1024} denote the 512-bit, 768-bit and 1024-bit primes (respectively) that we use to compute running times. They are defined as follows:

$$\begin{aligned} p_{512} &= 186 \cdot (2^{258} 3^{161}) - 1, \\ p_{768} &= 2 \cdot (2^{386} 3^{242}) - 1, \\ p_{1024} &= 353 \cdot (2^{514} 3^{323}) - 1. \end{aligned}$$

For pure C implementations, our results are 18%, 26%, and 19% faster than the previous work presented in [8], for 85, 128, and 170 bits security levels, respectively. Implementing field addition in x86-64 assembly gives a speedup of 4% (in comparison to our pure C implementations) on the Mac OS X platform for 768-bit values of p (128-bit security level). Also, our assembly result is 28% faster than the the one presented in [8] as one can see in Table 3.

For targeting embedded devices, we implement the post-quantum algorithms on iOS2 iPad 2 ARM Cortex-A9 operating at 1 GHz and Arndale ARM Cortex-A15 operating at 1.7 GHz.

The timing results for various security levels are reported in Table [8]. Pure C timing implementations on ARM Cortex-A15 are 0.629s, 1.77s, and 3.81s on 85, 128, and 170 bits security levels. Having only addition implemented in assembly gives us an speed up of 1.4% in comparison to the pure C implementations. The assembly results are primarily done using inline assembly which allows assembly code to be written in the same file as regular C code. Although not readily apparent, this method is prone to bugs and inherent inefficiencies. A better approach would be to write separate files containing the assembly code and link them to the C program.

There are few research work that have considered implementations of post-quantum cryptosystems on embedded systems mainly on FPGAs and 8-bit Microcontrollers including the ones given in [5] (lattice-based cryptography), [7] (code-based cryptography), and [4], [6] (McEllice cryptosystems). As our implementations are mainly focused on ARM-powered embedded systems and PC platforms, comparison with those works are infeasible.

Table 3. Timings for our C implementation of key exchange for $\ell_A = 2$ and $\ell_B = 3$.

Mac OS Macbook Pro Intel Core i5-2415M @ 2.4 GHz [8]			
Field Size (Prime)	p_{512}	p_{768}	p_{1024}
Quantum Security	85 bits	128 bits	170 bits
Pure C Timing (s)	0.113	0.303	0.529
Mac OS Macbood Pro Intel Core i5-2415M @ 2.4 GHz [This work]			
Pure C Timing (s)	0.093 (18%)	0.226 (26%)	0.429 (19%)
C with ASM (ADD/Mult)	–	0.217 (28%)	–
iOS2 iPad 2 ARM Cortex-A9 @ 1 GHz dual core [This work]			
Pure C Timing (s)	1.06	2.68	5.30
Arndale ARM Cortex-A15 @ 1.7 GHz dual core [This work]			
Pure C Timing (s)	0.629	1.77	3.81
C with ARM ASM (Add)	0.620	–	–

6 Conclusion

In this paper, we find that the key-exchange protocol presented in [8] can be realistically implemented and used on mobile communication devices at reasonable security levels. We first describe in detail a quantum-resistent key-exchange scheme [8], based on isogeny computations on elliptic curves. Two implementations of the scheme, one targeting PC platforms (x86-64 architecture), and the second targeting embedded devices (ARM Cortex-A9 and ARM Cortex-A15 processors), are developed. On a PC Platform the we demonstrated the speed-ups in the range between 18% and 26% compared to the original implementations. Our implementations on ARM processors are the first implementations of this particular post-quantum algorithm on embedding devices reported to date. With execution times between 1 and 5 seconds, these implementations demonstrate the feasibility of using the scheme on portable devices, such as smartphones and tablets, running iOS and Android.

References

1. Koblitz, N.: Elliptic Curve Cryptosystems. Mathematics of Computation 48, 203–209 (1987)
2. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Proceedings of Advances in Cryptology (CRYPTO 1985). 417–426 (1986)
3. 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)
4. 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)
5. 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)
6. 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)
7. 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)
8. 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)
9. Feo, L.D., Jao, D., Plut, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Cryptology ePrint Archive, Report 2011/506 (2011) http://eprint.iacr.org/.
10. Silverman, J.H.: The Arithmetic of Elliptic Curves. Volume 106 of GTM. Springer, New York (1992)
11. Tate, J.: Endomorphisms of abelian varieties over finite fields. Inventiones Mathematicae 2, 134–144 (1966)
12. Vélu, J.: Isogénies entre courbes elliptiques. Comptes Rendus de l'Académie des Sciences Paris Séries A-B 273, A238–A241 (1971)
13. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006) http://eprint.iacr.org/.
14. 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)
15. Bröker, R.: Constructing supersingular elliptic curves. Journal of Combinatorics and Number Theory 1(3), 269–273 (2009)
16. Bernstein, D., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted Edwards Curves. In: Progress in Cryptology–AFRICACRYPT 2008. Volume 5032 of LNCS. 389–405 (2008)
17. Bernstein, D.J., Lange, T.: Explicit-Formulas Database (2007) http://www.hyperelliptic.org/EFD/index.html.
18. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation 48(177), 243–264 (1987)
19. Barrett, P.: Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: Advances in Cryptology–CRYPTO' 86. Volume 263 of LNCS. 311–323 (1987)