Towards Optimized and Constant-Time CSIDH on Embedded Devices

Amir Jalali[1](✉), Reza Azarderakhsh[1], Mehran Mozaffari Kermani[2], and David Jao[3]

[1] Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL, USA
{ajalali2016,razarderakhsh}@fau.edu

[2] Department of Computer Science and Engineering, University of South Florida, Tampa, FL, USA
mehran2@usf.edu

[3] Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, Canada
djao@uwaterloo.ca

Abstract. We present an optimized, constant-time software library for commutative supersingular isogeny Diffie-Hellman key exchange (CSIDH) proposed by Castryck et al. which targets 64-bit ARM processors. The proposed library is implemented based on highly-optimized field arithmetic operations and computes the entire key exchange in constant-time. The proposed implementation is resistant to timing attacks. We adopt optimization techniques to evaluate the highest performance CSIDH on ARM-powered embedded devices such as cellphones, analyzing the possibility of using such a scheme in the quantum era. To the best of our knowledge, the proposed implementation is the first constant-time implementation of CSIDH and the first evaluation of this scheme on embedded devices. The benchmark result on a Google Pixel 2 smartphone equipped with 64-bit high-performance ARM Cortex-A72 core shows that it takes almost 12 s for each party to compute a commutative action operation in constant-time over the 511-bit finite field proposed by Castryck et al. However, using uniform but variable-time Montgomery ladder with security considerations improves these results significantly.

Keywords: Commutative supersingular isogeny · Constant-time · Embedded devices · Post-quantum cryptography

1 Introduction

The construction of public-key cryptography schemes based on the elliptic curves isogeny problem was proposed by Couveignes in 1997 [11] which described a non-interactive key exchange based on the isogeny classes of ordinary elliptic curves defined over a finite field $\mathbb{F}_p$. In 2004, Rostovtsev and Stolbunov [26]

I. Polian and M. Stöttinger (Eds.): COSADE 2019, LNCS 11421, pp. 215–231, 2019.
https://doi.org/10.1007/978-3-030-16350-1_12

Table 1. Comparison of SIDH and CSIDH over NIST's level 1 quantum security [12]

Scheme	Speed	Key size (Bytes)	Constant-time	Quantum attack	Active attacks	Non-interactive
SIDH	∼10 ms	378 B	Yes	$p^{1/6}$	Yes	No
CSIDH	∼100 ms	64 B	Not yet	Subexponential	Not known	Yes

independently came up with the same construction which later led to the design of other primitives such as isogeny-based digital signature [29]. Although the isogeny-based public-key cryptography construction by Couveignes-Rostovtsev-Stolbunov is attractive in many aspects such as key size, in 2010, Childs, Jao and Soukharev [7] showed that there exists a subexponential quantum algorithm that can solve the ordinary curve isogeny underlying problem. The proposed attack targeted the commutative ideal class group $\mathrm{cl}(\mathcal{O})$ for isogeny of ordinary curves; thus made this primitive unsuitable for the post-quantum era.

In 2006, Charles-Lauter-Goren [6] proposed a set of secure cryptographic hash functions from the supersingular curves isogeny graphs. Inspired by their work, in 2011, Jao and De Feo [22] proposed a Diffie-Hellman key exchange protocol from the isogeny of supersingular elliptic curves which was not vulnerable to Childs's quantum attack because of the non-commutative ring of endomorphisms in supersingular curves. Their interactive Supersingular Isogeny Diffie-Hellman (SIDH) key exchange is the fundamental basis of CCA secure Supersingular Isogeny Key Encapsulation (SIKE) mechanism [21] which was submitted to NIST PQC standardization project.

Due to the Child's quantum attack and impractical performance results of Couveignes-Rostovtsev-Stolbunov scheme, this primitive has been disregarded by community. Even the recent effort by De Feo-Kieffer-Smith [15] still takes several minutes to perform a single commutative action, in spite of using optimized state-of-the-art techniques.

Recently, Castryck et al. [5] proposed a new modification on the Couveignes-Rostovtsev-Stolbunov original scheme by adopting it to supersingular elliptic curves. However, instead of defining the supersingular curve over full ring of endomorphisms, the proposed scheme is restricted to the prime field $\mathbb{F}_p$ which preserves the commutative action of isogeny. They named the Diffie-Hellman key exchange scheme constructed over the commutative action as CSIDH (Commutative Supersingular Isogeny Diffie-Hellman). The main motivation behind using supersingular curves is to accelerate the commutative action rather than to address the security concerns on the Couveignes-Rostovtsev-Stolbunov raised by Child's quantum attack. In fact, CSIDH proposal can still be solved *theoretically* in subexponential time using the quantum algorithm as it is discussed by Biasse-Jao-Sankar [3] which targets the abelian hidden shift problem in the context of isogeny of supersingular curves [5]. However, as it is stated in [5], since the CSIDH public-key, in contrast to SIDH, contains only a single curve coefficient, it is not vulnerable to torsion point images attacks presented by Petit [25]. Furthermore, CSIDH simple key validation makes it inherently secure against CCA attacks proposed by Galbraith et al. [17]. Table 1 provides an abstract comparison between CSIDH and SIDH over NIST's level 1 security. The performance metrics provided in this table are based on optimized implementations on Intel Skylake processors. The performance of SIDH scheme on embedded devices is also investigated in detail in [18–20,23,27].

Recent detailed analysis in [4] shows that the CSIDH may have some security concerns with respect to quantum attacks. However, it offers efficient and fast key validation as well as extremely small key size. Moreover, other cryptography applications can be derived from the commutative group action similar to traditional Diffie-Hellman. For instance, De Feo and Galbraith [14] recently proposed SeaSign, a set of compact signatures from supersingular isogeny group action. Therefore, it is important to evaluate and analyze different aspects of this scheme such as performance and security on the practical settings.

The initial performance report of the commutative group action in [1,5] is based on a variable-time, mixed C and ASM implementation on Intel Skylake processors. Recent performance improvement of CSIDH by Meyer et al. [24] was also designed on top of the proof of concept implementation of CSIDH [5] and thus is variable-time. Note that as it is stated clearly in [5], the proof of concept implementation of CSIDH is unfit for production and it is totally vulnerable to timing and power analysis attacks.

In this work, we present a constant-time software library for CSIDH which targets 64-bit ARM-powered embedded devices. The main motivation behind this work is to evaluate the performance and the feasibility of using CSIDH in the real setting while the proposed software is secure against timing analysis attacks due to the constant-time implementation. We provide a set of modifications to the initial implementation of CSIDH in [5] over different layers from field arithmetic to group operations.

Since the proposed commutative action operation is implemented in constant-time, it can be simply adopted inside other applications of commutative supersingular isogeny to evaluate their performance in real settings.

The paper is organized as follows. Section 2 provides preliminaries on the isogeny of supersingular elliptic curves and explains the CSIDH scheme in a nutshell. Section 3 describes our approach to implement the entire commutative Diffie-Hellman key exchange efficiently and constant-time on embedded devices. The CSIDH benchmark results on two popular cellphones are presented in Sect. 4 and a comparison of constant- and variable-time implementations is provided. We conclude this work in Sect. 5.

2 Background

In this section, a brief description of supersingular curves isogeny and its application to construct a Diffie-Hellman key exchange protocol is presented. We refer the readers to [5,13,16,28] for more details.

2.1 Isogeny of Supersingular Curves

An ℓ-degree isogeny ϕ_ℓ is a rational function that maps an elliptic curve E defined over a field K to another curve E'. E' is a unique curve up to isomorphism and the map can be defined by a kernel which is a point P of order ℓ on E.

$$\phi_\ell : E \to E'/\langle P \rangle, P \in E[\ell].$$

The ring of endomorphisms of E is defined over the algebraic closure of K and denoted as $\mathrm{End}(E)$. Considering an elliptic curve E defined over a finite field $\mathbb{F}_p$, in case of ordinary elliptic curves, the $\mathrm{End}(E)$ is defined only over the base field $\mathbb{F}_p$, while for supersingular curves, it is defined over some extension field.

In contrast to Couveignes-Rostovtsev-Stolbunov scheme, CSIDH is constructed on supersingular curves. Therefore, only the subring of $\mathrm{End}(E)$ which is defined over $\mathbb{F}_p$, i.e., $\mathrm{End}_{\mathbb{F}_p}(E)$, is considered [5]. We have $\mathrm{End}_{\mathbb{F}_p}(E) \cong \mathcal{O}$ where $\mathcal{O}$ is an order in an imaginary quadratic field [14].

2.2 Class Group Action

For a supersingular elliptic curve E defined over $\mathbb{F}_p$ with $\mathrm{End}_{\mathbb{F}_p}(E) \cong \mathcal{O}$, the $\mathrm{cl}(\mathcal{O})$ is the ideal class group of $\mathcal{O}$. The action of an $\mathcal{O}$-ideal $\mathfrak{a}$ can be defined by an isogeny $\phi : E \to E'$ and denoted as $\mathfrak{a} * E$; its kernel $\ker(\phi)$ is presented by a torsion subgroup of points $E[\mathfrak{a}]$ on the curve E. The j-invariant, $j(E)$, of a curve E divides the $\mathrm{End}(E)$ into isomorphism classes where the isomorphic curves share the j-invariant value. Moreover, according to [11], the set of $j(E)$ constructs a hard homogeneous space which immediately implies the construction of a Diffie-Hellman like protocol.

2.3 Commutative Isogeny Diffie-Hellman Key Exchange

Considering the isogeny group action as described above on supersingular curves, CSIDH is defined over a finite field $\mathbb{F}_p$ with the prime p of the form $p = 4.\ell_1 \cdots \ell_n - 1$. Here ℓ_i are small odd primes and in the proposed parameter setting in [5] contain 74 primes which together construct a 511-bit prime value. Since the curve is supersingular, all ℓ_i are Elkies primes.

Using supersingular curves in CSIDH in contrast to Couveignes-Rostovtsev-Stolbunov original scheme makes it easy to find a curve with cardinality equal to $\#E(\mathbb{F}_p) = p+1$ and $\ell_i | p+1$. Thus $\#E(\mathbb{F}_p)$ is congruent to 0 modulo all primes.

Similar to SIDH efficient implementation [10], in order to take advantage of fast and compact Montgomery arithmetic, the starting curve is defined as $E_0 : y^2 = x^3 + x$ which is an instance of Montgomery curve and therefore all the corresponding isomorphic curves are also in Montgomery form.

Private Key. The private key is defined as an n-tuple $(e_1, \cdots, e_n)$ of integers chosen from $[-m, m]$ where m is a small integer. Note that the value of m is defined by the provided security level as it is discussed in details in [5, Section 7]. According to their security analysis, $m = 5$ is sufficient to provide 125-bit and

61-bit of classical and quantum security, respectively. De Feo and Galbraith also provide the estimation of range for m regarding the NIST higher security levels [14, Table 1].

Public Key. Each n-tuple private key represents the ideal class $[\mathfrak{a}] = [\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}]$ on the ideal class group $\mathrm{cl}(\mathcal{O})$ and generates a public key by applying the group action on the base curve E_0. The isomorphic curve generated by this group action is a Montgomery curve $[\mathfrak{a}]E : y^2 = x^3 + Ax^2 + x$ which whose coefficient $A \in \mathbb{F}_p$ is the corresponding public key.

Shared Secret. Since the group action is commutative, Alice and Bob compute the shared secret in a non-interactive procedure. They generate their key pairs as $([\mathfrak{a}], E_A)$ and $([\mathfrak{b}], E_B)$. Alice applies the action using her secret $[\mathfrak{a}]$ on the Bob's public key E_B and computes $[\mathfrak{a}]E_B$. Conversely, Bob computes $[\mathfrak{b}]E_A$ using his action and Alice's public key. The shared secret is the final curve coefficient $[\mathfrak{a}][\mathfrak{b}]E_0 = [\mathfrak{a}]E_B = [\mathfrak{b}]E_A$. Figure 1 demonstrates the key exchange procedure in a nutshell.

Alice		**Bob**
$\mathtt{SK}_A = (e_{A1}, \cdots, e_{An})$		$\mathtt{SK}_B = (e_{B1}, \cdots, e_{Bn})$
$[\mathfrak{a}] = [\mathfrak{l}_1^{e_{A1}} \cdots \mathfrak{l}_n^{e_{An}}]$		$[\mathfrak{b}] = [\mathfrak{l}_1^{e_{B1}} \cdots \mathfrak{l}_n^{e_{Bn}}]$
$\mathtt{PK}_A = [\mathfrak{a}]E_0 = E_A$		$\mathtt{PK}_B = [\mathfrak{b}]E_0 = E_B$
	$\xleftarrow{E_B}$	
	$\xrightarrow{E_A}$	
$\mathtt{Shared}_A = [\mathfrak{a}]E_B = [\mathfrak{a}][\mathfrak{b}]E_0$		$\mathtt{Shared}_B = [\mathfrak{b}]E_A = [\mathfrak{b}][\mathfrak{a}]E_0$

Fig. 1. CSIDH key exchange.

3 Constant-Time CSIDH Implementation

In this section, we outline our strategy for implementing an optimized and constant-time CSIDH on 64-bit ARMv8 processors. We engineered the underlying finite field arithmetic for the proposed field size and adopted different optimizations to evaluate a high performance and constant-time implementation of CSIDH on embedded devices.

All the field arithmetic operations described in this work are implemented using hand-written ARMv8 assembly to reduce the compiler overhead and provide the most-optimized results.

3.1 Field Arithmetic Modulo p_{511}

Starting from simple field arithmetic, modular addition and subtraction modulo p_{511} are implemented in constant-time. In contrast to the CSIDH proof of concept implementation [5] in which the modulo operation is only performed when the results are required to be corrected, our addition result is always subtracted from p_{511} and tested for borrow overflow. Based on the last borrow value, either p_{511} or 0 is added to the result for final correction. The same strategy is adopted for modular subtraction in order to have constant-time modular addition/subtraction.

Montgomery Modular Multiplication. Following the optimized SIDH implementation by Castello-Longa-Naehrig [10], Castryck et al. designed the entire curve operations in x-only arithmetic to take advantage of the optimized and compact Montgomery formulas. Accordingly, field multiplication and reduction are implemented using the Montgomery multiplication which is expected to offer the optimal performance since the p_{511} does not have any special form to enable further optimization techniques.

Since the prime is 511-bit, there is only one bit space left for any overflows and it is impossible to use optimization techniques such as lazy reduction to postpone the reduction operation. Therefore, using the Montgomery multiplication seems to be a better option rather than separate multiplication and Montgomery reduction due to the optimal memory usage and compactness.

The p_{511} is not a *Montgomery-friendly* prime which means that $p' = -p^{-1} \bmod r$ is not equal to 1 for the target radix, i.e., $r = 2^{64}$. This adds extra multiplication operations to the reduction part.

Because of the special shape of the CSIDH prime $p = 4.\ell_1 \cdots \ell_n - 1$, a straightforward strategy to find a Montgomery-friendly primes suggests the form

$$p = r.\ell_1 \cdots \ell_n - 1,$$

where r is the implementation target radix. This adds a considerable length (an extra word) to the field size without enhancing security level. Therefore, searching for a Montgomery-friendly prime in the context of CSIDH does not seem to add any performance improvement and the prime p_{511} is a proper choice for the target security level.

Since the filed elements over $\mathbb{F}_{p_{511}}$ are stored in an array of 8×64-bit words, 32 available 64-bit general registers inside the ARMv8 cores are adequate for implementing the modular multiplication efficiently using operand-scanning method. Therefore, we implement a compact and constant-time operand-scanning Montgomery multiplication using ARMv8 assembly, taking advantage of 64-bit wide general registers. Similar to our constant-time addition, the final result of the multiplication is always subtracted from p_{511} and according to the borrow overflow it adds to either p_{511} or 0 for the final correction.

Field Inversion. The constant-time field inversion is implemented using FLT algorithm in which for a field element a, the inverse of the element is computed as $a^{-1} = a^{p-2} \bmod p$. Surprisingly, the variable-time CSIDH proof of concept implementation uses the same approach for computing the inverse of an operand, while faster non-constant algorithm such as Extended Euclidean

Algorithm (EEA) could have been utilized. Moreover, they used binary square-and-multiply method to compute such exponentiation which is not an efficient approach in terms of performance, but it offers slightly less memory usage.

We implemented the exponentiation using fixed-window method with precomputed table. We set the window length to 6-bit which led to a table with 28 $\mathbb{F}_p$ elements. This consumes roughly 1.8 KB of memory which is negligible to the obtained performance improvement. The proposed addition chain is highly-optimized for the 6-bit window and it costs $29\mathbf{M} + 2\mathbf{S}$ operations[1] for generating the table, and $73\mathbf{M} + 510\mathbf{S}$ for computing addition chain. Therefore, a field inversion costs $102\mathbf{M} + 512\mathbf{S}$ in our window method, while it costs $255\mathbf{M} + 510\mathbf{S}$ for binary method which is used in the CSIDH proof of concept.

Square Root. The square root test over $\mathbb{F}_{p_{511}}$ for a field element a is implemented in constant-time by computing $a^{\frac{p-1}{2}} \bmod p$. Instead of using binary method in the CSIDH proof of concept implementation, we adopted the window method. The proposed addition chain computes the square root test using $71\mathbf{M} + 510\mathbf{S}$ in addition to $27\mathbf{M} + 2\mathbf{S}$ for the precomputed table generation. This leads to the total $98\mathbf{M} + 512\mathbf{S}$ computations, in contrast to $255\mathbf{M} + 510\mathbf{S}$ cost of binary method.

Note that in contrast to projective implementation of SIDH which only requires one inversion at the very end of each round, projective CSIDH requires several field inversion and square root computations inside each group action. Therefore, the above optimizations provide considerable enhancement in overall performance and efficiency of the protocol.

3.2 Scalar Multiplication

The Montgomery curve and x-only arithmetic offer a set of fast and compact formulas for computing curve arithmetic and isogeny computations. The CSIDH proof of concept implementation [5] is implemented based on the Montgomery group arithmetic. However, since the proposed implementation is non-constant time, it is entirely vulnerable to DPA and SPA attacks. In particular, the Montgomery ladder implementation for computing scalar multiplication is totally vulnerable to the power attacks and the exact value of scalar can be retrieved easily by power trace analysis [8].

To mitigate this vulnerability, we adopted the constant-time Montgomery ladder using the constant-time conditional `cswap` function. Since the point scalars in the CSIDH scheme have variable length, the constant-time ladder adds a significant extra operations to the scheme compared to non-constant version, but since the scalar in the commutative action operation is directly related to the private key, this modification is necessary.

The constant-time left-to-right Montgomery ladder is illustrated in Algorithm 1. It computes the scalar multiplication using $n - 1$ number of operations for different bit-length scalars where n is the finite field bit-length.

[1] **M** and **S** stand for field multiplication and field squaring, respectively.

Algorithm 1. Constant-time variable length scalar multiplication

```
Input  : $k = \sum_{i=0}^{n-1} k_i 2^i$ and $\mathtt{x}(P)$ for $P \in E(\mathbb{F}_p)$.
Output: $(X_k, Z_k) \in \mathbb{F}_p^2$ s.t. $(X_k : Z_k) = \mathtt{x}([k]P)$.
1: $X_R \leftarrow X_P, Z_R \leftarrow Z_P$
2: $X_Q \leftarrow 1, Z_Q \leftarrow 0$
3: for $i = n-2$ downto $0$ do
4:     $(Q, R) \leftarrow \mathtt{cswap}(Q, R, (k_i \ \mathtt{xor}\ k_{i+1}))$
5:     $(Q, R) \leftarrow \mathtt{xDBLADD}(Q, R, P)$
6: end for
7: $(Q, R) \leftarrow \mathtt{cswap}(Q, R, k_0)$
8: return $Q$
```

As it is already implemented in [10] and pointed out in [24], the `xDBLADD` function inside Algorithm 1 computes the simultaneous point addition and doubling using precomputed $(A + 2C : 4C)$ values to reduce the number of operations. We state that the effect of this optimization on the overall performance of our constant-time CSIDH is negligible.

Remark 1. The constant-time Montgomery ladder in Algorithm 1 is computationally expensive. However, using this algorithm guarantees the DPA and SPA resistance. Alternatively, to achieve significant better performance results, we can adopt a uniform ladder with various number of iterations for different scalars such that $k_{n-1} = 1$ as it is outlined in Algorithm 2. Since the algorithm is uniform, it does not reveal any information about the scalar bit values. However, the scalar bit-length can still be exposed by DPA. We included both implementations in our software to illustrate the difference in performance results of the CSIDH scheme. Further details are provided in Sect. 4.

Remark 2. In order to be resistant against DPA, Coron [8] proposed different countermeasures for scalar multiplication in the context of elliptic curve cryptography. According to his analysis, the countermeasures do not significantly impact efficiency. However, they also do not thwart all kinds of power attacks. Moreover, adopting such techniques results in variable-time software which is dependent to the inputs. Therefore, we choose to use fully constant-time Montgomery ladder inside our software to be resistant against all kinds of timing and power attacks.

The rest of Montgomery arithmetic such as `xDBL`, `xADD`, are constant-time and therefore no modifications are needed. However, we note that depending on the inputs, the number of group operations and subsequently field arithmetic counts can vary in the CSIDH variable-time implementation. Accordingly, in order to make the scheme entirely constant-time, we need to modify the key exchange operations and make them independent of inputs. In the next section, we describe these modifications.

Algorithm 2. Uniform and variable-time scalar multiplication

```
Input  : $k = \sum_{i=0}^{n-1} k_i 2^i$ with $k_{n-1} = 1$ and $\mathtt{x}(P)$ for $P \in E(\mathbb{F}_p)$.
Output: $(X_k, Z_k) \in \mathbb{F}_p^2$ s.t. $(X_k : Z_k) = \mathtt{x}([k]P)$.
1: $X_R \leftarrow X_P, Z_R \leftarrow Z_P$
2: $Q \leftarrow \mathtt{xDBL}(P)$
3: for $i = n-2$ downto $0$ do
4:     $(Q, R) \leftarrow \mathtt{cswap}(Q, R, (k_i \text{ xor } k_{i+1}))$
5:     $(Q, R) \leftarrow \mathtt{xDBLADD}(Q, R, P)$
6: end for
7: $(Q, R) \leftarrow \mathtt{cswap}(Q, R, k_0)$
8: return $Q$
```

3.3 Key Exchange Operations

As it is discussed in details in [5, Section 8], the most prominent operation inside CSIDH is the commutative group action. This operation computes the resulting curve coefficient given a starting curve and an n-tuple private key. The provided proof of concept implementation of group action in [5] is fast and optimized. However, as it is discussed before, its timing and performance directly depend on the input which makes it impossible to utilize in the practical settings. In this section, we provide a set of modifications to make key exchange operations constant-time. These modifications result in notable performance degradation to the scheme. However, they are necessary to be resistant against timing and power attacks.

Constant-Time Commutative Action. In order to compute an ℓ-degree isogeny using Vélu's formulas [30], we need to find a kernel point of order ℓ from torsion subgroup $E[\ell]$ on the curve. On Montgomery curves, a set of projective x-only formulas for arbitrary degree isogenies were proposed by Costello and Hisil [9] which the CSIDH proof of concept implementation is constructed upon.

In the context of SIDH, since the exact degree of isogeny is defined prior to the key exchange (2^{e_A} and 3^{e_B} for Alice and Bob, respectively), two pairs of base points are chosen from each torsion subgroups $P_A, Q_A \in E[2^{e_A}]$ and $P_B, Q_B \in E[3^{e_B}]$ as public parameters. Using these bases, Alice and Bob simply compute their secret isogeny kernel points $R_A = Q_A + [n_A]P_A$ and $R_B = Q_B + [n_B]P_B$ which accelerate the computation. However, this is not the case in the CSIDH scheme since the degree of isogeny action $[\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}]$ is directly related to the each party's secret key $(e_1, \cdots, e_n)$. Therefore, as it is noted in [5] the kernel of each small degree isogeny $\ell_i^{e_i}$ in each step of isogeny computation is retrieved by sampling a random x-coordinate followed by a square root test (to check whether it is defined over $\mathbb{F}_p$ or imaginary $\mathbb{F}_{p^2}$) and a multiplication by $(p+1)/\ell_i$ which because of the special shape of p outputs a point of order ℓ_i or the point at infinity $\mathcal{O}$ with the probability of $1 - 1/\ell_i$ and $1/\ell_i$, respectively. After finding a kernel for each small degree isogeny ϕ_{ℓ_i}, the isogeny map is computed on the

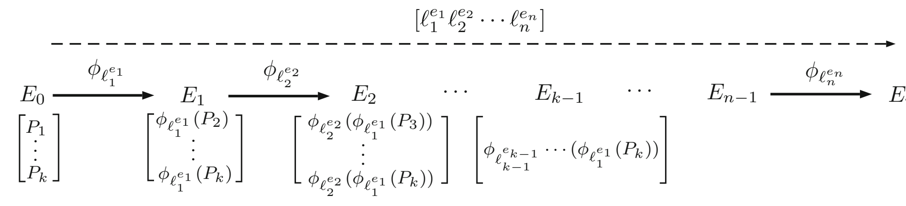
$$\begin{tikzcd}
E_0 \arrow[r, "\phi_{\ell_1^{e_1}}"] \arrow[rrrrrrr, dashed, bend left, "{[\ell_1^{e_1}\ell_2^{e_2}\cdots\ell_n^{e_n}]}"] & E_1 \arrow[r, "\phi_{\ell_2^{e_2}}"] & E_2 & \cdots & E_{k-1} & \cdots & E_{n-1} \arrow[r, "\phi_{\ell_n^{e_n}}"] & E_n \\
\begin{bmatrix} P_1 \\ \vdots \\ P_k \end{bmatrix} & \begin{bmatrix} \phi_{\ell_1^{e_1}}(P_2) \\ \vdots \\ \phi_{\ell_1^{e_1}}(P_k) \end{bmatrix} & \begin{bmatrix} \phi_{\ell_2^{e_2}}(\phi_{\ell_1^{e_1}}(P_3)) \\ \vdots \\ \phi_{\ell_2^{e_2}}(\phi_{\ell_1^{e_1}}(P_k)) \end{bmatrix} & & \begin{bmatrix} \phi_{\ell_{k-1}^{e_{k-1}}}\cdots(\phi_{\ell_1^{e_1}}(P_k)) \end{bmatrix} & & &
\end{tikzcd}$$

Fig. 2. Computing action using auxiliary base points on E_0.

current curve. This procedure is consecutively performed for all the small degree isogeny maps which together construct the action.

The random sampling procedure is expensive and it significantly affects the performance of the action operation, especially when it fails to provide an ℓ_i order point. It is possible to define a set of base points with predefined orders $\{P_1 \in E[\ell_1], \cdots, P_k \in E[\ell_k]\}$ such that $\{\ell_1, \cdots, \ell_k\}$ are a subgroup of primes from $\{\ell_1, \cdots, \ell_n\}$ on the base curve E_0 similar to SIDH. Furthermore, in each step of isogeny computations, the image of these base points can be computed on the next curve. Therefore, for the small degree isogenies $(\ell_1, \cdots, \ell_k)$ with higher probability of failure in random sampling $(1/\ell_1, \cdots, 1/\ell_k)$, the kernel points are ready at each step to use for the isogeny computations.

For the larger degrees, since the failure probability is relatively small, we can stick to the random sampling to reduce the memory usage. Figure 2 illustrates this procedure for some predefined value of k[2]. In each step of isogeny computation, one of the image points is dropped and its image is not needed for the further isogeny computations.

Therefore, at the beginning of the procedure, the x-coordinate of k points is stored while at the k-th step, only one point is required. As a result, the isogeny evaluations of the auxiliary points are reduced as the algorithm steps forward.

However, this adds some security concerns to the scheme since the isogeny kernels are the image of some public base points through small degree isogenies. Moreover, a set of extra isogeny evaluations in each step is added to the scheme. Therefore, more investigation on the security and performance of the proposed method is needed. We leave the possibility of using such a technique for the future work.

To be able to practically evaluate the variation of the main loop in variable-time group action implementation, we performed a statistical analysis on the number of required iterations for uniformly random inputs. We conducted 10^6 experiments of variable-time group action from random inputs and recorded the number of iterations. Figure 3 presents the result of this experiment. We observed that the number of iterations for different private keys can be as large as 60 (only once in 10^6 experiments), while some inputs only require 9 iterations. This is

[2] The optimal value for k is directly related to the prime and the trade-off between memory usage and performance.

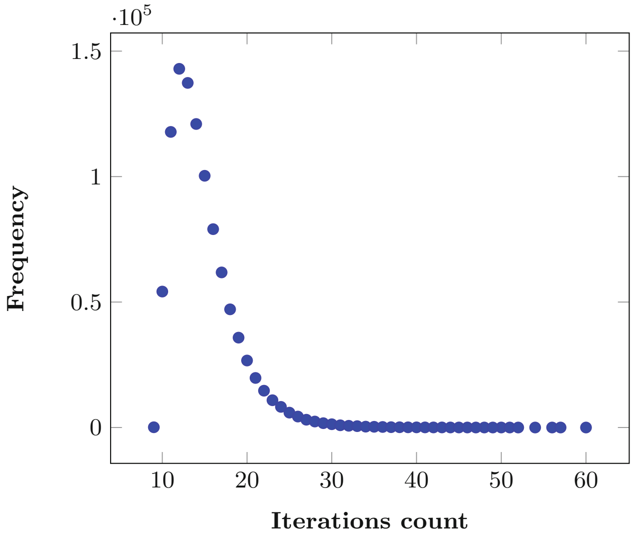

Fig. 3. The frequency of different iterations count over 10^6 experiments of group action.

the result of the variation in the n-tuple secret key and the failure probability of computing the point of order ℓ_i for some ℓ_is.

To mitigate this variation, the variant number of iterations should be replaced with an upper bound value. The straightforward value for the upper bound is n (the number of e_is). However, this is very conservative which significantly degrades the performance of the software. In fact, the recent detailed analysis of CSIDH in [2] shows that it is sufficient to iterate $r = 59$ iterations of the main loop to obtain a negligible failure probability ($<2^{-32}$) of the group action, considering the range of $e \in \{-5, \cdots, 5\}$.

To evaluate the performance of constant-time CSIDH on the target embedded devices, we modified the CSIDH variable-time action operation algorithm by removing all the conditional and `while` loop statements in an efficient way to provide a constant-time implementation of this algorithm inside our software. We outline the procedure in Algorithm 3. We refer the readers to our implementation for further details.

Using `cswap` in Algorithm 3 implies useless point multiplication and isogeny computations. However, the frequency of these operations may directly reveal the sign and the value of private key in a detailed power analysis trace. In fact, the detailed analysis of variable-time implementation of CSIDH in the Fig. 3 indeed demonstrates that the main loop iteration can have notable variation depending on the value of private key.

Algorithm 3. Constant-time commutative class group action

```
Input  : $A \in \mathbb{F}_p$ and a list of integers $(e_1, \cdots, e_n)$.
Output: $B \in \mathbb{F}_p$ s.t. $[\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}]E_A = E_B$.
 1: // Decoding private key
 2: for $i = 0$ to $n - 1$ do
 3:     Set $s \leftarrow 1$ if $e_i$ is negative, otherwise $s \leftarrow 0$.
 4:     Set $v \leftarrow 0$ if $e_i$ is 0, otherwise $v \leftarrow 1$.
 5:     $e_i(s) \leftarrow e_i - (2 \cdot s \cdot e_i)$.
 6:     $e_i(\bar{s}) \leftarrow 0$.
 7:     $k(\bar{s}) \leftarrow \ell_i \cdot k(\bar{s})$.
 8:     $k(\bar{v}) \leftarrow (\ell_i - v \cdot (\ell_i - 1)) \cdot k(\bar{v})$.
 9: end for
10: // Action
11: for $i = 0$ to $n - 1$ do
12:     $A' \leftarrow A$.
13:     Sample a random $x = \mathtt{x}(P) \in \mathbb{F}_p$.
14:     Set $u \leftarrow 0$ if $x^3 + Ax^2 + x$ is a square in $\mathbb{F}_p$, otherwise $u \leftarrow 1$.
15:     $R \leftarrow [k(u)]P$.
16:     $d(u) \leftarrow 1$.
17:     for $j = 0$ to $n - 1$ do
18:         $f \leftarrow 1$, $r \leftarrow e_j(u)$.
19:         for $z = j + 1$ to $n - 1$ do
20:             $f \leftarrow f \times (\ell_z - (\bar{e_z}(u).(\ell_z - 1)))$.
21:         end for
22:         $Q \leftarrow [f]R$.
23:         Set $t \leftarrow 1$ if $Z_Q = 0$, otherwise $t \leftarrow 0$.
24:         Compute $\phi_{\ell_j} : A \rightarrow B$ s.t. $\ker(\phi_{\ell_j}) = Q$.
25:         cswap$(B, A, (\bar{t} \vee e_j(u)))$.
26:         $e_j(u) \leftarrow e_j(u) - 1$.
27:         $m \leftarrow (e_j(u) \vee t \vee \bar{r})$.
28:         $e_j(u) \leftarrow e_j(u) + t$.
29:         $k(u) \leftarrow k(u) \times (\ell_j - (m \cdot (\ell_j - 1)))$.
30:         $d(u) \leftarrow (d(u) \wedge e_j(u))$.
31:     end for
32:     $q \leftarrow q \oplus q$.
33:     cswap$(A', A, q)$.
34:     $q \leftarrow (d(0) \wedge d(1))$.
35: end for
36: return $A$
```

We observe that the scalar multiplications in lines 15 and 22 of Algorithm 3 directly generate the secret isogeny kernel. Therefore, from the security viewpoint, the scalar multiplication indeed requires to be resistant against side-channel attacks. However, as we see in this algorithm, the point P is randomly generated in each iteration.

This makes it very hard for an attacker to retrieve the value of the kernel point using power analysis. Therefore, using a uniform but variable-time Montgomery ladder can be an option since it improves the performance results notably. We leave the further investigations for future work.

Constant-Time Key Generation. The CSIDH key generation algorithm is straightforward. First, private keys are randomly generated as an n-tuple integers from $[-m, m]$ interval, where in case of our implementation $m = 5$. Next, the public key is computed by performing the group action using the generated private key on the base curve E_0.

The CSIDH proof of concept implementation generates both private and public keys in variable-time. Since our proposed group action in Sect. 3.3 is constant-time, the public key generation is indeed constant-time. We made some trivial changes using constant-time conditional instructions on the private key random generation procedure to make the entire key generation resistant to timing attacks. We refer the reader to our implementation for further details.

Public Key Validation. The CSIDH scheme offers a fast and straightforward public key validation by examining whether the given curve is supersingular or not. Based on [5, Proposition 8], the supersingularity check justifies that the represented curve (public key) has the right endomorphism ring and therefore is a valid public key.

Since the public key validation procedure only uses public values, it does not require to be resistant against power and timing attacks and therefore it can be designed and implemented in variable-time as it is already implemented efficiently in [5]. Accordingly, we adopted the same implementation of the public key validation inside our software and used the fast variable-time Montgomery ladder algorithm for point multiplication inside the cofactor multipliers algorithm. We refer the readers to [5] for further details on the implementation of public key validation.

4 Performance Results and Discussion

In this section, we present our implementation[3] results on the two popular cellphones, Google Pixel 2 and Huawei Nexus 6P equipped with 64-bit ARM Cortex-A72 and Cortex-A57, respectively. Our software is designed in a way that can be simply compiled to either constant-time or variable-time executable using `gcc` preprocessors.

We used `aarch64-linux-gnu-gcc` compiler for cross-compiling the executable with `-static -O3` flags and ran it using `adb shell` on the cellphones. Table 2 presents the performance of our constant-time and variable-time software on target platforms. Note that the variable-time implementation is also based on our optimized hand-written assembly field arithmetic and provides an optimized performance estimation of CSIDH proof of concept implementation on embedded devices. Moreover, the total CSIDH results are obtained by running the entire protocol, containing key generations and key validation on the target processors. The difference between timing and the number of clock cycles for target

[3] Our library is publicly available at: https://github.com/amirjalali65/ARMv8-CSIDH.

Table 2. Performance results of constant-time (with constant-time Montgomery ladder) and variable-time CSIDH. (Benchmarks were obtained on 1.95 GHz Cortex-A57 and 2.4 GHz Cortex-A72 cores running Android 7.1.1 and 8.1.0, respectively)

		Constant-time		Variable-time [5]	
		Cortex-A57	Cortex-A72	Cortex-A57	Cortex-A72
Key validation	Cycles $\times 10^6$	-	-	38	23
	Seconds	-	-	0.02	0.01
Group action	Cycles $\times 10^6$	30,459	28,872	624	552
	Seconds	**15.6**	**12.03**	**0.32**	**0.23**
Total CSIDH	Cycles $\times 10^6$	61,054	57,912	1,326	1,224
	Seconds	31.3	24.1	0.68	0.51

Table 3. Performance results of constant-time (with uniform but variable-time Montgomery ladder) CSIDH.

Operation	Cortex-A57	Cortex-A72
Group action	$11{,}286 \cdot 10^6$ cc **5.94 s**	$10{,}824 \cdot 10^6$ cc **4.51 s**
Total CSIDH	$22{,}819 \cdot 10^6$ cc 12.01 s	$21{,}744 \cdot 10^6$ cc 9.06 s

platforms refers to the processor's working frequency and its micro-architecture technology. Cortex-A72 core is the new high-performance 64-bit ARM core with optimized pipeline and micro-architecture which is used inside many embedded devices recently.

We also benchmarked our constant-time software with uniform but variable-time Montgomery ladder as it is discussed in Sect. 3.2. Table 3 presents the performance results of this experiment on our target platforms. We observed more than 2.5 times performance improvement just by using uniform variable-time ladder. This implies that the main challenge for designing a constant-time isogeny group action is to find an optimized and secure way of computing scalar multiplication. Considering the countermeasure techniques for uniform variable-time ladder, the performance results become more practical.

4.1 Discussion

Although the performance results of the constant-time CSIDH is not extremely promising, but since it is constructed on a commutative action, it offers a set of cryptographic applications such as digital signature which can be very useful in the quantum era. Based on the estimations in [14], such signatures are not very high-performance even by using fast and variable-time commutative action.

Therefore, in order to be able to practically adopt the isogeny commutative group action, its performance should be enhanced. The main bottleneck lies in the constant-time Montgomery ladder for computing point multiplication. Furthermore, useless computations inside the constant-time group action is undesirable.

One significant improvement to the algorithm can be achieved by using a faster but insecure ladder as it is discussed in the previous section. We can also reduce the number of useless operations and point sampling inside the group action by defining a set of base points as CSIDH public parameters and compute the image of these points in each step. While this may improve the performance notably, it adds some concerns regarding the security of the scheme. We believe that using the above suggestions and imposing security countermeasures can make the CSIDH and isogeny group action a suitable candidate for different applications, specifically because of its small key sizes and fast key validation.

5 Conclusion

In this work, we presented an efficient and constant-time implementation of CSIDH scheme on embedded devices. We engineered a set of constant-time and highly-optimized field and group arithmetic implementation using ARM assembly and provided a CSIDH software which is secure against SPA and DPA attacks. We benchmarked our software on two popular cellphones equipped with 64-bit high-performance ARM Cortex-A57 and Cortex-A72 cores. To the best of our knowledge, this work is the first constant-time implementation of CSIDH and the first evaluation of this scheme on embedded devices.

The implementation results imply that the fully constant-time implementation of the scheme may not be practical for many applications and it needs more investigations on the performance improvement and security analysis. However, because of many advantages of the isogeny commutative group action, the proposed software can still be used inside the applications with static keys and restricted band-width, taking advantage of fast key validation and small key size of CSIDH. Since side-channel attacks resistance is one of the fundamental requirements for any cryptographic scheme, we hope this work attracts engineers and researchers to investigate the performance improvement and security of the constant-time isogeny group action as it seems to be one of the promising candidates for designing different cryptographic applications in the future.

Acknowledgment. This work is supported in parts by NSF CNS-1801341, NIST-60NANB17D184, NIST-60NANB16D246, and ARO W911NF-17-1-0311, as well as NSERC, CryptoWorks21, Public Works and Government Services Canada, Canada First Research Excellence Fund, and the Royal Bank of Canada.

References

1. An Efficient Post-quantum Commutative Group Action. https://csidh.isogeny.org/software.html
2. Bernstein, D.J., Lange, T., Martindale, C., Panny, L.: Quantum Circuits for the CSIDH: Optimizing Quantum Evaluation of Isogenies. https://quantum.isogeny.org/qisog-20181031.pdf
3. Biasse, J.-F., Jao, D., Sankar, A.: A quantum algorithm for computing isogenies between supersingular elliptic curves. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 428–442. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13039-2_25
4. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH and ordinary isogeny-based schemes. IACR Cryptology ePrint Archive (2018). https://eprint.iacr.org/2018/537
5. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. IACR Cryptology ePrint Archive (2018). https://eprint.iacr.org/2018/383
6. Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic hash functions from expander graphs. J. Cryptol. **22**(1), 93–113 (2009)
7. Childs, A.M., Jao, D., Soukharev, V.: Constructing elliptic curve isogenies in quantum subexponential time. J. Math. Cryptol. **8**(1), 1–29 (2014)
8. Coron, J.-S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_25
9. Costello, C., Hisil, H.: A simple and compact algorithm for SIDH with arbitrary degree isogenies. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 303–329. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_11
10. 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). https://doi.org/10.1007/978-3-662-53018-4_21
11. Couveignes, J.M.: Hard Homogeneous Spaces. IACR Cryptology ePrint Archive (2006). http://eprint.iacr.org/2006/291
12. Feo, L.D.: Isogeny Graphs in Cryptology. http://defeo.lu/docet/assets/slides/2018-05-31-gdr-securite.pdf
13. Feo, L.D.: Mathematics of isogeny based cryptography. CoRR abs/1711.04062 (2017). http://arxiv.org/abs/1711.04062
14. De Feo, L., Galbraith, S.D.: SeaSign: compact isogeny signatures from class group actions. IACR Cryptology ePrint Archive (2018). https://eprint.iacr.org/2018/824
15. Feo, L.D., Kieffer, J., Smith, B.: Towards practical key exchange from ordinary isogeny graphs. CoRR (2018). http://arxiv.org/abs/1809.07543
16. Galbraith, S.D.: Mathematics of Public Key Cryptography. Cambridge University Press, Cambridge (2012)
17. Galbraith, S.D., Petit, C., Shani, B., Ti, Y.B.: On the security of supersingular isogeny cryptosystems. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 63–91. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_3

18. Jalali, A., Azarderakhsh, R., Mozaffari-Kermani, M.: Efficient post-quantum undeniable signature on 64-Bit ARM. In: Adams, C., Camenisch, J. (eds.) SAC 2017. LNCS, vol. 10719, pp. 281–298. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72565-9_14
19. Jalali, A., Azarderakhsh, R., Kermani, M.M.: NEON SIKE: supersingular isogeny key encapsulation on ARMv7. In: Security, Privacy, and Applied Cryptography Engineering - 8th International Conference, SPACE, pp. 37–51 (2018)
20. Jalali, A., Azarderakhsh, R., Kermani, M.M., Jao, D.: Supersingular isogeny Diffie-Hellman key exchange on 64-bit ARM. IEEE Trans. Depend. Secure Comput. (2017)
21. Jao, D., et al.: Supersingular isogeny key encapsulation. Submission to the NIST Post-Quantum Standardization project (2017). https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions
22. Jao, D., De 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). https://doi.org/10.1007/978-3-642-25405-5_2
23. Koziel, B., Jalali, A., Azarderakhsh, R., Jao, D., Kermani, M.M.: NEON-SIDH: efficient implementation of supersingular isogeny Diffie-Hellman key exchange protocol on ARM. In: Cryptology and Network Security - 15th International Conference, CANS, pp. 88–103 (2016)
24. Meyer, M., Reith, S.: A faster way to the CSIDH. IACR Cryptology ePrint Archive, p. 782 (2018). https://eprint.iacr.org/2018/782
25. Petit, C.: Faster algorithms for isogeny problems using torsion point images. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 330–353. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_12
26. Rostovtsev, A., Stolbunov, A.: Public-key cryptosystem based on isogenies. IACR Cryptology ePrint Archive (2006). http://eprint.iacr.org/2006/145
27. Seo, H., Liu, Z., Longa, P., Hu, Z.: SIDH on ARM: faster modular multiplications for faster post-quantum supersingular isogeny key exchange. IACR Trans. Cryptogr. Hardw. Embed. Syst. **3**, 1–20 (2018)
28. Silverman, J.H.: The Arithmetic of Elliptic Curves. GTM, vol. 106. Springer, New York (2009). https://doi.org/10.1007/978-0-387-09494-6
29. Stolbunov, A.: Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. Adv. Math. Commun. **4**(2), 215–235 (2010). https://doi.org/10.3934/amc.2010.4.215
30. Vélu, J.: Isogénies entre courbes elliptiques. CR Acad. Sci. Paris, Séries A **273**, 305–347 (1971)