18 Seconds to Key Exchange: Limitations of Supersingular Isogeny Diffie-Hellman on Embedded Devices

Philipp Koppermann[1], Eduard Pop[1], Johann Heyszl[1], and Georg Sigl[1,2]

[1] Fraunhofer Institute for Applied and Integrated Security, Munich, Germany
{philipp.koppermann,eduard.pop,johann.heyszl}@aisec.fraunhofer.de
[2] Technische Universität München, Munich, Germany
sigl@tum.de

Abstract. The quantum secure supersingular isogeny Diffie-Hellman (SIDH) key exchange is a promising candidate in NIST's on-going post-quantum standardization process. The evaluation of various implementation characteristics is part of this standardization process, and includes the assessment of the applicability on constrained devices. When compared to other post-quantum algorithms, SIDH appears to be well-suited for the implementation on those constrained devices due to its small key sizes. On the other hand, SIDH is computationally complex, which presumably results in long computation times. Since there are no published results to test this assumption, we present speed-optimized implementations for two small microcontrollers and set a first benchmark that can be of relevance for the standardization process. We use state-of-the art field arithmetic algorithms and optimize them in assembly. However, an ephemeral key exchange still requires more than 18 seconds on a 32-bit Cortex-M4 and more than 11 minutes on a 16-bit MSP430. Those results show that even with an improvement by a factor of 4, SIDH is in-fact impractical for small embedded devices, regardless of further possible improvements in the implementation. On a positive note, we also analyzed the implementation security of SIDH and found that appropriate DPA countermeasures can be implemented with little overhead.

Keywords: Post-quantum cryptography, supersingular, isogeny, SIDH, ECC, embedded devices, efficient implementation, side-channel analysis.

1 Introduction

It is well known that future large-scale quantum computers can efficiently compute Shor's algorithm [29], and thus threaten public-key cryptosystems that rely on the (elliptic curve) discrete logarithm problem or the integer factorization problem. Even though full-fledged quantum computers are yet to arrive, today's recorded encrypted communication could be broken with a quantum computer years later. In the past few years, this led to intensive research and a large number of published papers dealing with post-quantum cryptography

(PQC) i.e. cryptographic algorithms that are considered to be secure against an attack by a quantum computer. The National Institute of Standards and Technology (NIST) [3] published a report on PQC providing an overview of existing algorithms including an announcement for standardization. In this report, NIST distinguishes between five approaches: lattice-based cryptography, code-based cryptography, multivariate polynomial cryptography, hash-based signatures, and *other* which include isogeny-based cryptography. We are interested in how such cryptosystems can be used in the *Internet of Things* (IoT). When analyzing different PQC approaches, it becomes apparent that most of them require large private and public keys. Large key sizes imply at least two problems for smaller embedded devices: First, since the transmission of data requires the majority of the energy budget, the size of the public parameters including the public key must be kept small. Second, small embedded devices often possess less than ten kilobytes of memory. Therefore, PQC schemes that feature large key pairs, as for example the McEliece cryptosystem [23] that needs about 220 kB for a single public key at a 128-bit quantum security level, are impractical on such devices.

With public keys as small as 330 bytes [7], the quantum-secure supersingular isogeny Diffie-Hellman (SIDH) key exchange [16] is a promising candidate to secure the communication on embedded devices. SIDH is based on elliptic curves and shares similarities with traditional elliptic curve cryptography (ECC); however, the underlying number-theoretic problem is the isogeny-graph problem. An isogeny ϕ is an algebraic map between two elliptic curves, which are defined over a finite field of characteristic p. The point multiplication $Q = [n]P$ of a point P with some scalar n, which is well known in traditional ECC, can be seen as a special case of an isogeny where $Q = \phi(P)$ for identical curves. Finding the isogeny between the known domain and co-domain (in case of distinct elliptic curves) constitutes the isogeny-graph problem, which is an instance of the so-called claw problem [16]. This isogeny-graph consists of vertices representing isomorphism classes of elliptic curves that are connected by edges representing isogenies. Alice and Bob start from the vertex that is the public curve and traverse this graph via a seemingly random walk. Ultimately, they end up on two curves sharing some value that is used as the shared secret. So far, the best known available algorithm for a quantum computer can solve this problem with a time complexity of $\mathcal{O}(p^{1/6})$. In this work, we investigate on SIDH, which was introduced in 2011 by Jao and De Feo [16]. Costello et al. [8] published the first constant-time implementation on Intel Sandy Bridge and Haswell processors using projective coordinates. In terms of speed, their results were recently surpassed by Hernández et al. [10]. In 2016, Koziel et al. [20] presented a highly-optimized implementation in affine coordinates on a comparably less powerful 32-bit Cortex-A8 and Cortex-A15 architecture using the NEON SIMD architecture extension. However, until now it remains unclear how SIDH performs on microcontrollers which possess less computational power and lack dedicated SIMD accelerators.

Our contributions. We present speed-optimized SIDH implementations for two popular microcontrollers, namely the 32-bit ARM Cortex-M4 and the 16-bit TI MSP430X, targeting a 128-bit quantum and 192-bit classical security level. For our implementation, we optimized the field arithmetic functions i.e. modular addition, subtraction, multiplication, and reduction in assembly. Compared to the generic C-implementation, this reduced thy cycle count for each field operation by more than a factor of 15. However, our results show that an ephemeral key exchange still requires more than 18 seconds on the ARM Cortex-M4 and more than 11 minutes on the MSP430X, which is clearly too long for most real-life applications. On a positive note, we show that randomized projective coordinates, as a countermeasure to thwart differential power analysis (DPA), can be implemented for only 3% computational overhead and perform a leakage detection test to demonstrate the effectiveness as part of a case study.

Outline. In Sect. 2 we describe the preliminaries followed by a brief description of the SIDH protocol in Sect. 3. In Sect. 4 we present our implementation for the Cortex-M4 and the MSP430X with special emphasis on the prime field operations. We summarize our performance results in Sect. 5 and discuss randomized projective coordinates and public key validation in Sect. 6. Finally, we conclude in Sect. 7.

2 Preliminaries

The quantum-secure SIDH key exchange protocol uses elliptic curve arithmetic, i.e. elliptic curves as mathematical structures and its associated point arithmetic, which is well understood in the ECC domain. However, in order to describe SIDH, further preliminary definitions need to be introduced. Therefore, we provide the reader with a brief description of isogenies, supersingular curves, and ℓ-torsion subgroups. A more detailed description can be found in [13, 14, 32].

Isogenies. Suppose $\mathbb{E}_1$ and $\mathbb{E}_2$ are two elliptic curves with the same cardinality, i.e. $\#\mathbb{E}_1 = \#\mathbb{E}_2$, and with identity elements $\mathcal{O}_1$ and $\mathcal{O}_2$, respectively. Then an isogeny is a surjective mapping $\phi : \mathbb{E}_1 \mapsto \mathbb{E}_2$ if and only if $\phi(\mathcal{O}_1) = \mathcal{O}_2$. This mapping is also a group homomorphism, i.e. $\forall P, Q \in \mathbb{E}_1 : \phi(P + Q) = \phi(P) + \phi(Q)$. Two elliptic curves are called isogenous if there exists an isogeny between them. The kernel of an isogeny is defined as the set of points on the domain curve that map to the identity element: $\ker(\phi) = \{P \in \mathbb{E}_1 \mid \phi(P) \mapsto \mathcal{O}_2\}$. There is a one to one correspondence between isogenies and their kernels, and an isogeny can be computed from its kernel. Using the kernel of an isogeny to store it as a data structure is common in SIDH. If $\mathbb{E}_1$ is an elliptic curve, then for any subgroup $H \subseteq \mathbb{E}_1$ there exists a unique (up to isomorphism) elliptic curve $\mathbb{E}_2$ with an associated isogeny $\phi : \mathbb{E}_1 \mapsto \mathbb{E}_2$ with $\ker(\phi) = H$. This isogeny is a natural map: its image is isomorphic to the quotient of the kernel in the domain, i.e. $\mathbb{E}_2 \cong \mathbb{E}_1 / \ker(\phi)$. Parts of the protocol deal with the computation of an isogeny of a certain degree. For the purpose of this paper, the degree of an isogeny is the cardinality of its kernel.

Supersingular curves. Elliptic curves can be either ordinary or supersingular. An elliptic curve $\mathbb{E}(\mathbb{F}_q)$ with $q = p^a$, where p is a prime and $a \in \mathbb{Z}$, is called supersingular if $\#\mathbb{E}(\mathbb{F}_q) \equiv 1 \bmod p$. Supersingular curves were proven to reduce the computational complexity of the elliptic curve discrete logarithm problem [24], which restricts their application in ECC. However, Childs et al. [4] showed that solving the isogeny problem for ordinary elliptic curves, i.e. finding an isogeny between two known ordinary curves, can be done in quantum-polynomial time. This fact implies that cryptographic protocols based on the ordinary isogeny problem are insecure in the post-quantum world. The opposite is considered to be true regarding the supersingular case [13].

$\boldsymbol{\ell}$-torsion subgroups. Let $\mathbb{E}/\mathbb{F}_q$ be an elliptic curve defined over a finite field of prime characteristic p. For any integer ℓ we can define the ℓ-torsion subgroup of $\mathbb{E}$ as $\mathbb{E}[\ell] := \{P \in \mathbb{E} \mid [\ell]P = \mathcal{O}\}$. The ℓ-torsion subgroup of an elliptic curve also has a special structure: $\mathbb{E}[\ell] \cong \mathbb{Z}/\ell\mathbb{Z} \oplus \mathbb{Z}/\ell\mathbb{Z}$. In other words, $\mathbb{E}[\ell]$ can be generated by two independent points $P, Q \in \mathbb{E}$ of order ℓ, i.e. $\mathbb{E}[\ell] = \langle P, Q\rangle := \{[m]P + [n]Q \mid m, n \in \mathbb{Z}\}$. SIDH takes advantage of this structure as will be described in Sect. 3.

3 The supersingular isogeny Diffie-Hellman key exchange

Jao and De Feo [16] proposed SIDH as a variant of the Diffie-Hellman key exchange based on the isogeny-graph problem. Similarly to standard Diffie-Hellman, SIDH has a number of public parameters, as described in Sect. 3.1, and is separated into two phases: the key pair and shared secret key computation as presented in Sect. 3.2 and Sect. 3.3, respectively. We shortly describe algorithms for the large degree isogeny computation. This operation is analogous to the scalar multiplication in traditional ECC, and is computed iteratively as detailed in Sect. 3.4.

3.1 Public parameters

Before keys can be exchanged, SIDH requires to fix the base field, the supersingular elliptic curve and some points on this curve.

Base field. A finite field $\mathbb{F}_q := \mathbb{F}_{p^2}$ is fixed where p is some large prime with the form $p = \ell_A^{e_A} \cdot \ell_B^{e_B} \cdot f \pm 1$. The values ℓ_A and ℓ_B are small primes, and $e_A, e_B, f \in \mathbb{N}$, with f being a cofactor chosen in such a way that p is prime. Alice will compute isogenies of degree $\ell_A^{e_A}$ and Bob will compute isogenies of degree $\ell_B^{e_B}$. Note that it is recommended to chose $\ell_A^{e_A} \approx \ell_B^{e_B}$ to achieve a similar security level and computational complexity for both parties.

Elliptic curve and bases. Alice and Bob define a supersingular elliptic curve $\mathbb{E}_0/\mathbb{F}_{p^2}$. Next, four points are chosen $P_A, Q_A, P_B, Q_B \in \mathbb{E}_0$ fixing the bases $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ generating the $\ell_A^{e_A}$-, and $\ell_B^{e_B}$-torsion subgroups, respectively: $\mathbb{E}_0[\ell_A^{e_A}] = \langle P_A, Q_A\rangle$ and $\mathbb{E}_0[\ell_B^{e_B}] = \langle P_B, Q_B\rangle$.

3.2 Key generation

Alice chooses two secret random integers $m_A, n_A \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$, not both divisible by ℓ_A and computes $R_A \coloneqq [m_A]P_A + [n_A]Q_A$. It holds that $R_A \in \langle P_A, Q_A\rangle = \mathbb{E}_0[\ell_A^{e_A}]$ and thus $\#\langle R_A\rangle = \ell_A^{e_A}$. Alice can then compute the isogeny ϕ_A with $\ker(\phi_A) = \langle R_A\rangle$ and thus $\deg(\phi_A) = \ell_A^{e_A}$ taking $\mathbb{E}_0$ to a new elliptic curve $\mathbb{E}_A$. The isogeny ϕ_A is the quotient map, so the curve $\mathbb{E}_A$ is isomorphic to $\mathbb{E}_0/\langle R_A\rangle$. Finally, Alice evaluates the points P_B and Q_B using the isogeny ϕ_A, and saves the values $\phi_A(P_B)$ and $\phi_A(Q_B)$. Bob proceeds *mutatis mutandis*. The triple $(\mathbb{E}_A, \phi_A(P_B), \phi_A(Q_B))$ is Alice's public key and the pair (m_A, n_A) is her private key. Furthermore, let $(\mathbb{E}_B, \phi_B(P_A), \phi_B(Q_A))$ and (m_B, n_B) be the similarly computed key pair belonging to Bob.

3.3 Shared secret computation

Alice now has access to Bob's public key $(\mathbb{E}_B, \phi_B(P_A), \phi_B(Q_A))$. The goal is to reach some new elliptic curve $\mathbb{E}_{BA}$ by computing a new isogeny $\phi'_A : E_B \mapsto \mathbb{E}_{BA}$. For this purpose, Alice uses her secret integers (m_A, n_A) and computes the point $S_A \coloneqq [m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A)$. As in the previous phase of the protocol, an isogeny ϕ'_A with $\ker(\phi'_A) = \langle S_A\rangle$ and thus $\deg(\phi'_A) = \ell_A^{e_A}$ can be efficiently computed taking $\mathbb{E}_B$ to the final elliptic curve $\mathbb{E}_{BA}$. Bob proceeds *mutatis mutandis* and computes the isogeny ϕ'_B and the elliptic curve $\mathbb{E}_{AB}$. It holds that $\mathbb{E}_{BA} \cong \mathbb{E}_{AB}$, which implies that their j-invariants $j(\mathbb{E}_{BA}) = j(\mathbb{E}_{AB})$. Alice and Bob can thus use this common value as a shared secret key. For further details regarding the j-invariants of elliptic curves, we refer the reader to [14].

3.4 Large degree isogeny computation

Given an elliptic curve $\mathbb{E}/\mathbb{F}_q$ and a subgroup $H \subseteq \mathbb{E}$ with $H \coloneqq \langle R\rangle, R \in \mathbb{E}, \operatorname{ord}(R) = \ell^e$, where ℓ is a small prime, one can compute an isogeny ϕ with $\ker(\phi) = H = \langle R\rangle$ and $\deg(\phi) = \#H = \ell^e$. For example, Alice was required to compute ϕ_A with $\ker(\phi_A) = \langle R_A\rangle = \langle [m_A]P_A + [n_A]Q_A\rangle$. However, to compute the isogeny ϕ_A, the problem needs to be divided into smaller operations comparable to decomposing the ECC scalar multiplication into single point additions. The isogeny ϕ can be written as a composition of e isogenies ϕ_i of degree ℓ. Jao and De Feo [16] accomplish that by making use of Vélu's formulas [31]. In short, they iteratively compute the isogeny ϕ taking the curve $\mathbb{E}$ to a curve isomorphic to the quotient of $\langle R\rangle$ in $\mathbb{E}$, i.e. $\phi : \mathbb{E} \mapsto \mathbb{E}/\langle R\rangle$. First set $R_0 \coloneqq R$ and $\mathbb{E}_0 \coloneqq \mathbb{E}$. Then for $0 \le i < e$, the simplest algorithm for the large-degree isogeny computation is the *multiplication oriented* approach and is given by:

$$\mathbb{E}_{i+1} = \mathbb{E}_i/\langle \ell^{e-i-1} R_i\rangle, \quad \phi_i : \mathbb{E}_i \mapsto \mathbb{E}_{i+1}, \quad R_{i+1} = \phi_i(R_i),$$

with $\mathbb{E}_e \cong \mathbb{E}/\langle R \rangle$ and $\phi_{e-1} \circ \phi_{e-2} \circ \cdots \circ \phi_0 = \phi$.

Strategies. Aside from the *multiplication oriented* algorithm, Jao and De Feo [11] also introduced and formally defined the *isogeny oriented* algorithm. In short, instead of relying on point multiplications as the main operation, the *isogeny oriented* approach computes mainly ℓ-isogeny evaluations. As the authors of [11] show, both of these approaches are non-optimal, i.e. they carry out more operations than necessary. Instead, they define the concept of an *optimal strategy* as the combination of the two approaches which results in the fewest number of base operations required. Optimal strategies can be computed in advance and stored as constants, as described by [8]. This technique has been used in a number of SIDH implementations, including ours.

4 Implementation

In this section, we provide the reader with a detailed description of our speed optimized implementation. We begin by describing the platform independent design decisions in Sect. 4.1. This is followed by a summary of the features of the two microcontrollers in Sect. 4.2 and a detailed description on the implementation of the prime field arithmetic for the corresponding architectures in Sect. 4.3.

4.1 Platform independent design decisions

In the following, we summarize a selection of design decisions that we made for our implementation:

Projective coordinates. As with traditional ECC, projective coordinates speed up each scalar point multiplication (performed twice for an ephemeral key exchange) as it reduces costly field inversions. Costello et al. [8] showed that a more compact representation is derived when operating on variable curve parameters represented in the projective space. Additionally, we represent curve points in projective coordinates and randomize them during scalar multiplication as a computationally efficient countermeasure to thwart DPA.

Structure of public keys. To limit the communication overhead and save resources such as energy, the size of the public key should be small. Compared to the initial proposal by Jao and De Feo [16], we follow Costello et. al [8] where the size of the public key is reduced from 768 bytes to 564 bytes. More precisely, the public key is a triple of the field elements in $\mathbb{F}_{p^2}$, representing the x-coordinates of $\phi_A(P_B), \phi_A(Q_B), \phi_A(Q_B - P_B)$ as an example for Alice. The normalized Montgomery curve parameter A of the public curve is recovered from those three points on the curve, and does not need to be included. Note that in [7] it was shown that the public key can be further reduced to only 330 bytes.

However, we discarded this technique because it reduces the speed by more than a factor of 3, which collides with our optimization preference for speed.

Chosen parameters. The characteristic of the field $\mathbb{F}_{p^2}$ is $p = 2^{372} \cdot 3^{239} - 1$ with $\lceil \log_2(p) \rceil = 751$. This prime precisely provides a 124-bit quantum security level, however, it is usually associated to a 128-bit quantum security level. Other primes are proposed in [20] such as $2^{250} \cdot 3^{159} - 1$ and $2^{493} \cdot 3^{307} - 1$ that provide a 83-bit and 162-bit quantum security level, respectively. We decided to target the 128-bit quantum security level, as it is considered to be reasonable secure for the next few decades, while being small enough for sufficient speed. The bases $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ are set by the following points: $P_A = [3^{239}](11, \sqrt{11^3 + 11})$, $Q_A = \tau(P_A)$, $P_B = [2^{372}](6, \sqrt{6^3 + 6})$, $Q_B = \tau(P_B)$, where $\tau : \mathbb{E}_0 \mapsto \mathbb{E}_0$ and $\tau(x, y) = (-x, iy)$. The base supersingular elliptic curve has the short Weierstrass form:

$$\mathbb{E}_0/\mathbb{F}_{p^2} : y^2 = x^3 + x. \tag{1}$$

One scalar as private key. Instead of choosing two randomly distributed integers m_A and n_A and computing the secret isogeny whose kernel is $\langle [m_A]P_A + [n_A]Q_A \rangle$, Alice chooses one single integer $m_A \in [1, 2^{371} - 1]$ and the isogeny with the kernel $\langle P_A + [2 \cdot m_A]Q_A \rangle$. Similarly, the kernel of Bob's secret isogeny will by $\langle P_B + [3 \cdot m_B]P_B \rangle$, where $m_B \in [1, 3^{238} - 1]$. This is done in order to facilitate the use of pre-computed strategies for isogeny computations. As Costello et al. [8] point out, this reduces the total number of possibilities for the public key by a factor of 3, for Alice, and a factor of 4, for Bob. However, the authors claim there is currently no reason to believe the security of the system is affected by this implementation choice.

4.2 Microcontrollers

For development and testing purposes, we used the MSP430FR5994 launchapd and FRDM-K64F development board, that feature two popular 16-bit and 32-bit microcontroller architectures, respectively:

TI MSP430FR5994. Is based on the 16-bit MSP430X architecture running at a maximum clock frequency of 16 MHz with 8 kB of RAM and 256 kB of non-volatile FRAM (Ferromagnetic Random Access Memory). The FRAM can be accessed at a frequency of 8 MHz and can be used for long-term storage, as well as machine code and data storage. When the core is clocked with 16 MHz, additional wait cycles are introduced if FRAM access is required due to the difference in the two operating clock frequencies. This can effect the overall performance and is described in the results section. We used Code Composer Studio for code development and compilation with optimization level set to speed.

Kinetis K64. Is based on the 32-bit ARM Cortex-M4 core running at 120 MHz with 1 MB of flash memory and 512 kB of RAM. The compilation was done using the GNU ARM Embedded toolchain with optimization set to $-$O3.

4.3 Finite field operations

Finite field arithmetic is the foundation for various public-key cryptosystems. As discussed in Sect. 3.1, SIDH defines elliptic curves over the extension field $\mathbb{F}_{p^2}$. Yet, operations in the extension field $\mathbb{F}_{p^2}$ are composed of operations in the finite field $\mathbb{F}_p$. Since the performance of operations in $\mathbb{F}_p$ has strong impact on the overall performance, it is crucial to optimize them for best speed results. The relevant operations are addition, subtraction, multiplication, and modular reduction. All operations run in constant-time, are written in assembly with fully unrolled loops and no calls to subroutines.

Addition and subtraction. The modular addition and subtraction correspond to standard 24-limb and 48-limb operations for the 32-bit Cortex-M4 and the 16-bit MSP430X, respectively. Note that both the operands and the result will be elements in $[0, 2p-1]$, instead of $[0, p-1]$. As [30] points out, this circumvents the necessity of a subtraction at the end of the modular operation. After an addition or subtraction has taken place, the result has to be reduced to $[0, 2p-1]$. For the addition, since $a, b < 2p$, it holds that $c \coloneqq a + b < 4p$, i.e. the bitlength of $a+b$ is higher by at most 1 when compared to the bitlength of a and b. If $c > 2p$, then $c - 2p \in [0, 2p-1]$ will be the correct result. In order to avoid conditional branching, instead of comparing c to $2p$, the use of the following well-known strategy is employed:

1. Set $c \leftarrow c - 2p$, and remember the borrow bit b.
2. Compute the bitmask $m \coloneqq (b \ \& \ 2p)$, and set $c \leftarrow c + m$.

If $b = 1$, it holds that the subtraction $c - 2p$ produces a borrow out, so initially $c < 2p$, which means c was in the correct interval before the first assignment. The second step corrects this issue by adding $m = 2p$ to c. If, however, $b = 0$, then $c > 0$ is in the correct interval after the first step. The second assignment, i.e. the addition of the bitmask $m = 0$ does not alter the result.

The same process is similar for the subtraction. The difference $a - b$ is either negative, i.e. there is a borrow at the end of the subtraction, or it is already an element in $[0, 2p-1]$. In any case, the second step of this strategy is carried out, so the bitmask m described above is computed and added onto $a - b$, achieving the same result.

Multiplication. We decided to use Karatsuba multiplication [17] because it has a time complexity of only $\mathcal{O}(n^{\log_2 3})$; for comparison, the standard schoolbook multiplication has a time complexity of $\mathcal{O}(n^2)$. More precisely, we implemented a 1-level additive Karatsuba multiplication with Comba optimizations [28]. The purpose of the latter is to decrease expensive memory accesses and storage requirements for intermediate results. With these optimizations, the memory space dedicated to the result is only accessed when the final value for a specific limb has been computed.

In Karatsuba multiplication, two n-digit operands x, y represented in some base R are split into two parts each: the top (most significant) halves x_H, y_H, and the bottom (least significant) halves x_L, y_L. Define:

$$\begin{aligned} H &\coloneqq x_H \cdot y_H \\ L &\coloneqq x_L \cdot y_L \\ M &\coloneqq (x_H + x_L) \cdot (y_H + y_L) - L - H\,. \end{aligned}$$

Then the following holds:

$$x \cdot y = H \cdot R^n + M \cdot R^{n/2} + L\,. \qquad (2)$$

In our case, the operands x, y are 768 bits (96 bytes) long, in either 48-limb representation on the MSP430X, i.e. $n = 48, R = 2^{16}$, or 24-limb representation on the Cortex-M4, i.e. $n = 24, R = 2^{32}$. The result is stored in z, which is a $768 \cdot 2 = 1536$ bits (192 bytes) memory location. The most significant words are stored first. In order to store intermediate results, 96 bytes of stack space are allocated at the beginning of the routine. We start by computing the three partial multiplications as follows:

1. $(x_H + x_L) \cdot (y_H + y_L)$: To obtain both operands, two additions must be computed where each could produce a carry-out. Store $\tilde{x} \coloneqq x_H + x_L$ and $\tilde{y} \coloneqq y_H + y_L$ in z, and their carry-out bits in c_1 and c_2, respectively. If one of the carry bits is set, then the corresponding sum is $(n/2)+1$ limbs long, since x_H, x_L and y_H, y_L are half the size of x and y, respectively. Hence the extra limb might have a bit set, however, a technique can be used to disregard it and adjust the result afterwards instead. For this purpose, multiply the least significant $n/2$ limbs of $\tilde{x}$ and $\tilde{y}$ (ignoring the aforementioned extra limb, should it be present) via the Comba method, storing the n-limb long result i.e. 96-byte in the stack. If $c_1 = 1$, $\tilde{x}$ has an extra limb equalling 1 at the most significant part, so when carrying out the multiplication $\tilde{x} \cdot \tilde{y}$, the value $\tilde{y} \cdot 1 = \tilde{y}$ would be added to the most significant limb of the result. To avoid conditional branching, compute bitwise $(c_1 \mathbin{\&} \tilde{y})$ and add it to the most significant limb of the result, which accomplishes the same purpose, either adding $\tilde{y}$ or 0, depending on the value of c_1. Analogously, add $(c_2 \mathbin{\&} \tilde{x})$ to the most significant limb of the result.
2. $H \coloneqq x_H \cdot y_H$: use Comba multiplication and store the 96-byte-long result in the most significant part of z.
3. $L \coloneqq x_L \cdot y_H$: same as above, but store the result in the least significant 96 bytes of z.

After determining all partial multiplications, M can now be computed by subtracting H and L from the result of the first multiplication. The first product is stored in the first 96 bytes of the allocated stack space, so when subtracting H and L, save the results in the remaining 96 bytes. Afterwards, M can be added to the middle part of z as per Eq. (2). This spans the bytes 49-144 of z. Lastly, add the overflow resulting from the last digit addition, as well as any further overflows this operation might produce to the remaining bytes of z, in sequence.

We note that optimizing the multiplication by exploring further algorithms could potentially result in a performance improvement. For example, one could implement multi-level Karatsuba as well as exotic, microcontroller-optimized multiplication algorithms [22]. However, speeding up the multiplication by a factor-2, which is rather optimistic, can at best result in an overall speed-up of the key exchange by the same factor. Yet, making SIDH suitable for real-life applications, a speed-up by more than an order of magnitude is required. We emphasize that the performance limitations of SIDH reside on an protocol, rather than on an implementation level.

Reduction. The modular reduction is an adaptation of the well-known Montgomery reduction [25]: let $\mathbb{F}_p$ be the base field with $p = 2^{372} \cdot 3^{239} - 1$, $\lceil \log_2(p) \rceil = 751$, and define $R \coloneqq 2^{768}$ and $p' = -p^{-1} \mod R$. For any input $a < pR$, compute the Montgomery residue $c = aR^{-1} \mod p$:

$$c \coloneqq (a + (ap' \bmod R) \cdot p)/R\,. \tag{3}$$

This operation is generally computed iteratively: first define r as the bitsize of an integer, and set s such that $R = 2^{r \cdot s}$. In this case, $R = 2^{768}$, so for the Cortex-M4, $r = 32, s = 24$, and for the MSP430X, $r = 16, s = 48$. Set $c \leftarrow a$, then repeat s times: $c \leftarrow (c + (c \cdot p'' \mod 2^r) \cdot p)/2^r$, where $p'' = -p^{-1} \mod 2^r$.

As Costello et al. [8] showed, Eq. (3) can be converted for the chosen prime $p = 2^{372} \cdot 3^{239} - 1$ to:

$$c = a/2^{768} + ((ap' \bmod 2^{768}) \cdot 3^{239})/2^{396}\,,$$

which decreases the number of required multiplications for a modular reduction. Furthermore, they show that in the iterative process, it holds that $p'' = 1$, which allows the transformation: $c \leftarrow (c + (c \mod 2^r) \cdot (p + 1))/2^r$. More details regarding these transformations are available at [8]. This is advantageous, because in this case, $p + 1$ has a number of its least-significant limbs equal to 0 (11 limbs in 32-bit representation and 23 limbs in 16-bit representation), and they can thus be excluded from the multiplication. These individual multiplications were carried out using Comba optimizations.

4.4 Results for the assembly optimized field operations

In Table 1, we present the number of clock cycles for each field operation for future reference. We implemented the described algorithms in assembly and com-

Table 1. Cycle count including improvement factor for the prime field operations of the generic and assembly implementation on both architectures.

Operation	**Cortex-M4**			**MSP430X**		
	C	**ASM**	↑	**C**	**ASM**	↑
Addition	10,779	559	19.3	18,500	1,192	15.5
Subtraction	7,109	419	17.0	12,568	831	15.1
Multiplication	244,209	4,319	56.5	945,252	32,517	29.1
Reduction	167,619	3,254	51.5	586,596	20,094	29.2

pare the performance with the generic C implementation by Costello et al. [8], which we ported to our microcontrollers without further modification. It can be noted that our optimized operations require between 15 and 56 times fewer cycles than their generic counterparts. The speed-up of the assembly implementations is comparable for both architectures, while the difference in performance is linked to the architecture dependent word size. The improvement factor is higher for the Cortex-M4, which is likely a result of its lower cycle requirement when accessing consecutive memory locations. Both the generic and the optimized operations run in constant-time.

5 Results and analysis of constant-time implementations

In this section, we first report and compare our results for an ephemeral key exchange to other SIDH implementations. This comparison should aid the reader to classify our results and verify their soundness. Subsequently, we compare our implementation to other quantum-secure key exchange algorithms on embedded devices in order to evaluate our work in a broader context.

Table 2 compares the clock cycle count for the key pair generation and the shared secret key computation on the Cortex-M4 and the MSP430X to other published SIDH implementations. Note that the clock cycle count differs for Alice and Bob because the computational complexity depends on the selected prime $\ell_A^{e_A}, \ell_B^{e_B}$. For the 32-bit Cortex-M4, the code is compiled to a size of 71.53 kB, and key pairs are generated in 1025 and 1149 million clock cycles for Alice and Bob, respectively. Similar numbers are obtained for the shared secret key computation. For the 16-bit MSP430X microcontroller, we obtained a code size of 110.33 kB. The clock cycle count is reported for two different clock frequencies to show the effect of the introduced wait cycles linked to the lower clock frequency of the FRAM. In case of 8 MHz clock frequency, a key pair key is computed in about 4559 million cycles and a shared secret in about 4339 million cycles. The number of clock cycles increases to about 5480 and 5216 million clock cycles for key pair generation and shared secret key computation, respectively, when being clocked with 16 MHz.

Compared to the performance of the Cortex-M4, the MSP430X requires about 4-times more clock cycles which is linked to the reduced word size of

Table 2. Clock cycle count [$\times 10^6$] for SIDH on different processors supporting a 128-bit quantum security level.

Work	Platform	Word size	Key gen.		Secr. gen.	
			Alice	Bob	Alice	Bob
[10]	Intel Skylake	64-bit	27	31	25	29
[10]	Intel Haswell	64-bit	38	43	34	40
[8]	Intel Haswell	64-bit	51	59	47	57
[15]	ARM Cortex-A57	64-bit	103	118	97	113
[20]	Cortex-A15	32-bit	437	474	346	375
This work	Cortex-M4	32-bit	1025	1148	967	1112
This work	MSP430X (8 MHz)	16-bit	4260	4855	4020	4658
	MSP430X (16 MHz)	16-bit	5136	5824	4832	5600

16-bit. A similar relation is observed when we compare the 64-bit Cortex-A57 [15] and the 32-bit Cortex-A15 [20] implementation, indicating the plausibility of our results. Comparing the 32-bit Cortex-A15 implementation to our implementation on the Cortex-M4, the key generation and shared secret computation requires about 2.38-times and 2.79-times less cycles, respectively. Note that the Cortex-A15 core is based on the ARMv7 architecture and is equipped with features such as caches and the NEON SIMD architecture extension. The lack of such accelerator features explains the increase in clock cycles for our Cortex-M4 implementation. Most works optimized SIDH for 64-bit processors [8, 10, 15] making a comparison with smaller devices, such as the 16-bit MSP430X, unfair. On 64-bit processors, the current speed record for constant-time implementations is set by Jalali et al. [15], which represents an optimized version of the work by Costello et al. [15].

Comparison with other quantum-secure algorithms. In Table 3 we compare other quantum-secure on embedded devices with our implementation. Relevant parameters are performance in terms of required time measured in seconds and communication overhead measured in transmitted bytes. All listed implementations feature a similar security level of around 128-bit. NewHope [1] was implemented on the ARM Cortex-M4 and Cortex-M0 and executed in only 0.01 and 0.035 seconds, respectively. Even when comparing our Cortex-M4 implementation to NewHope on the less powerful Cortex-M0 (clocked with only 48 MHz), NewHope is more than 500-times faster with only 4-times higher communication overhead. Frodo [2] is a LWE-based quantum-secure key encapsulation with promising performance results as well. For smaller processors, there is only one implementation available for the Cortex-A8, however, its communication overhead implies that implementing it on constrained devices might be impractical. The SIDH implementation on the Cortex-A8 by Koziel et al. [20] shows tolerable execution time and indicates the general applicability of SIDH on such processors. However, compared to NewHope [1] or Frodo [2] the tremendous difference in speed becomes apparent. We conclude that SIDH has small key sizes but clearly suffers in speed, which leads to extensive computation time on small microcontrollers.

6 Implementation security

Contrary to other PQC algorithms (e.g. NTRU [21]), SIDH supports perfect forward-secrecy; however, this also requires the use of ephemeral keys. While forward-secrecy is a desirable property, the secure use of static keys is important for embedded devices due to limited computational power and energy budget. It is well known that elliptic curve based cryptosystems can be attacked by invalid point attacks [9], where a maliciously generated public key is used to gain access to the secret private key. In case of SIDH, the attacker may choose his public key in a way that forces the party using a static private key to reach a pre-computed shared secret, or to enable the attacker to gain knowledge about their static private key, as described in [8, 12]. To thwart this type of attack, the received public keys must be validated, i.e. the correctness of their generation must be verified. As it turns out, validation techniques in the context of SIDH are not trivial: they are either computationally efficient and insecure [8, 12], or secure and computationally inefficient [18]. For example, Kirkwood et al. [18] proposed a working validation technique, which requires as much time as an ephemeral key generation. Therefore, we decided to neglect the implementation of point validation techniques. However, with on-going research we expect computational efficient and secure point validation techniques to be found.

While a point validation technique is the first mandatory step towards the secure use of static keys, a software designer should be aware that static keys can facilitate some attacks. As attackers can typically get physical access to embedded devices, we consider side-channel analysis as an additional attack vector. When static keys are used, an attacker can acquire multiple traces using the

Table 3. Performance evaluation of different quantum-secure key exchange/encapsulation protocols on mid- and low-end processors.

Protocol	**Platform**	**Freq.**	**Latency [sec.]**		**Comm. [bytes]**	
		[MHz]	**Alice**	**Bob**	**A→B**	**B→A**
NewHope [1]	Cortex-M0	48	0.03	0.04	1824	2048
NewHope [1]	Cortex-M4	164	0.01	0.01	1824	2048
Frodo [2]	Cortex-A8	1000	0.08	0.08	11296	11288
SIDH [20]	Cortex-A8	1000	1.41	1.53	564	564
SIDH (this work)	Cortex-M4	120	16.59	18.83	564	564
	MSP430X	8	1035.00	1188.00	564	564
	MSP430X	16	623.00	714.00	564	564

same key. Therefore, we evaluate randomized projective coordinates in greater detail in Sect. 6.1 as a countermeasure for preventing DPA [19].

6.1 Randomized projective coordinates to thwart DPA

The shared secret computation phase poses a natural target for an attacker because he can control data which is directly processed with the secret private key i.e. the input point that is multiplied with the secret integer during elliptic curve scalar multiplication. DPA on this standard elliptic curve scalar multiplication is well understood. As explained in Sect. 4.1, we only use one integer as our secret scalar for the point multiplication. Here, we target Alice's secret integer n_a and assume that Bob is the malicious entity and can modify $\phi_B(P_A),\ \phi_B(Q_A)$.

$$S_A = \phi_B(P_A) + [n_A]\phi_B(Q_A)\,.$$

The scalar multiplication and the additional point addition is carried out using the three-point ladder as described by Jao and De Feo [16] and shown in Algorithm 1 (ladder_3pt). Note that the if-clause is only used here for readability

Algorithm 1 ladder_3pt: Three-point ladder [16] that computes $P + [n]Q$.

```
Input: $n, P, Q$
Output: $P + [n]Q$
1: $A = 0, B = Q, C = P$
2: for $i$ decreasing from $|n|$ to 1 do
3:     if $n_i = 0$ then
4:         $A \leftarrow 2A, \quad B \leftarrow A + B, \quad C \leftarrow A + C$
5:     else
6:         $A \leftarrow A + B, \quad B \leftarrow 2B, \quad C \leftarrow B + C$
7:     end if
8: end for
```

purposes; in our and most other implementations it is replaced by constant-time point swap to prevent SPA and timing attacks.

Coron [5] described *randomized projective coordinates* as an appropriate countermeasure to thwart DPA. This countermeasure is characterized by relatively low computational overhead. Implementing randomized projective coordinates implies a randomly generated λ being multiplied with the input points P, Q in their projective representation during the ladder initialization. Using Montgomery formulas [26], differential point addition for the x-coordinate is given by:

$$(P + Q)_x = (P_z - Q_z)[(P_x - P_z)(Q_x + Q_z) + (P_x + P_z)(Q_x - Q_z)]^2\,,$$

with the two input points $P = \{P_x, P_z\}$ and $Q = \{Q_x, Q_z\}$. The difference point $(P_z - Q_z)$ can be neglected in the usual case, but equals λ for randomized projective coordinates, which translates to one additional multiplication for each point addition. As shown in Alg. 1, two point additions are performed in each ladder step; thus, enabling randomized projective coordinates results in $744 = 2 \cdot 372$ and $758 = 2 \cdot 379$ additional multiplications in $\mathbb{F}_{p^2}$ for Alice and Bob, respectively. Compared to an unprotected implementation, we require only about 3% more cycles with randomized projective coordinates. This renders randomized projective coordinates a computationally efficient countermeasure.

Case study: leakage assessment on the FRDM-K64F. The Montgomery ladder combined with randomized projective coordinates is considered to be an effective countermeasure to thwart DPA. Even though we expect similar protection for the three-point ladder, a case study is useful for supporting this claim. We acquire EM traces with a Langer RF-B 3-2 near H-field probe (horizontal) placed above the packaged chip. For each implementation, we collect 5000 synchronized traces at at sampling rate of 5 GS/s using a LeCroy WavePro 725 Zi oscilloscope. We evaluate randomized projective coordinates on the FRDM-K64F (featuring the Cortex-M4) using the non-specific t-test as the leakage detection test [6, 27]. Fig. 1 shows on the left two ladder steps of the Montgomery ladder,

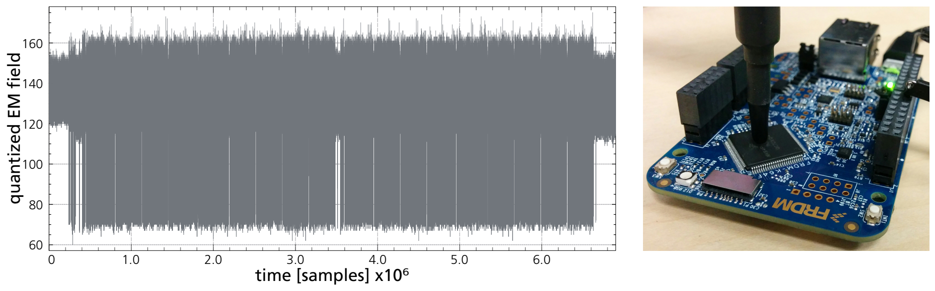

Fig. 1. Left: Exemplary EM trace for two ladder steps. Right: Langer probe placed above FRDM-K64F.

and on the right the probe placed above the FRDM-K64F.

The t-test can be used to detect whether the device's implementation has exploitable leakage. We first test and show that the device leaks secret information with no DPA countermeasure enabled. With the same measurement setup, we then evaluate the leakage with randomized projective coordinates. We apply a *fixed-vs-random* methodology on the input point, i.e. we acquire 2500 traces with a fixed input point and 2500 with a random input point; subsequently, the t-test determines whether the two data sets are significantly different to each other. The input point and the random number λ are sent to the development board via UART while the secret remains fixed. In case of the unprotected implementation, we fix λ to a constant value. Fig. 2 shows on the left the t-test with no DPA countermeasures and on the right with randomized projective coordinates. With no countermeasures enabled, the device fails the t-test as it exceeds the

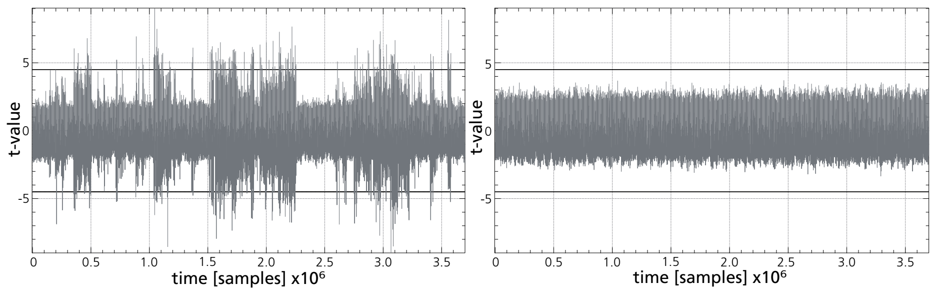

Fig. 2. Fixed-vs-random leakage detection test on the input point using 5000 traces. Left: no DPA countermeasure. Right: randomized projective coordinates enabled.

threshold $\pm C = 4.5$, which clearly indicates leakage. On the contrary, the test results after the introduction of randomized projective coordinates indicate the effectiveness of the countermeasure as expected.

7 Conclusions

We presented two implementations of SIDH targeting a 128-bit quantum security level for the 32-bit ARM Cortex-M4 and 16-bit TI MSP430X architectures that perform the shared secret key computation including key pair generation in about 18 seconds and 11 minutes, respectively. Although our results only set a first benchmark, we conclude that even the inferior performance results of the unprotected implementations indicate that SIDH is impractical for securing resource-constrained devices. It is likely that our implementations can be optimized by a small factor, but it seems to be unrealistic that the performance can be improved by a factor-100. We use randomized projective coordinates to thwart multi-trace DPA as it only reduces the speed by approximately 3%. However, we note that current public key validation techniques imply tremendous performance loss emphasizing the need for further research. Moreover, other quantum secure key encapsulation protocols (such as NewHope [1]) seem more suitable for embedded devices. Yet, we can imagine that SIDH is suitable for securing the Internet communication where typically more powerful processors are used.

References

1. Alkim, E., Jakubeit, P., Schwabe, P.: Newhope on ARM cortex-m. In: Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings. pp. 332–349 (2016), https://doi.org/10.1007/978-3-319-49445-6_19
2. Bos, J.W., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In: Proceedings of the 2016 ACM SIGSAC Conference on

Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 1006–1018 (2016), http://doi.acm.org/10.1145/2976749.2978425
3. Chen, L., Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D.: Report on post-quantum cryptography. US Department of Commerce, National Institute of Standards and Technology (2016)
4. Childs, A.M., Jao, D., Soukharev, V.: Constructing elliptic curve isogenies in quantum subexponential time. J. Mathematical Cryptology 8(1), 1–29 (2014), https://doi.org/10.1515/jmc-2012-0016
5. Coron, J.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings. pp. 292–302 (1999), https://doi.org/10.1007/3-540-48059-5_25
6. Coron, J., Naccache, D., Kocher, P.C.: Statistics and secret leakage. ACM Trans. Embedded Comput. Syst. 3(3), 492–508 (2004), http://doi.acm.org/10.1145/1015047.1015050
7. Costello, C., Jao, D., Longa, P., Naehrig, M., Renes, J., Urbanik, D.: Efficient compression of SIDH public keys. In: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I. pp. 679–706 (2017), https://doi.org/10.1007/978-3-319-56620-7_24
8. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny diffie-hellman. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. pp. 572–601 (2016), https://doi.org/10.1007/978-3-662-53018-4_21
9. Fan, J., Verbauwhede, I.: An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In: Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday. pp. 265–282 (2012), https://doi.org/10.1007/978-3-642-28368-0_18
10. Faz-Hernández, A., López, J., Ochoa-Jiménez, E., Rodríguez-Henríquez, F.: A Faster Software Implementation of the Supersingular Isogeny Diffie-Hellman Key Exchange Protocol. Cryptology ePrint Archive, Report 2017/1015 (2017), https://eprint.iacr.org/2017/1015
11. Feo, L.D., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. J. Mathematical Cryptology 8(3), 209–247 (2014), https://doi.org/10.1515/jmc-2012-0015
12. Galbraith, S.D., Petit, C., Shani, B., Ti, Y.B.: On the security of supersingular isogeny cryptosystems. In: Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. pp. 63–91 (2016), https://doi.org/10.1007/978-3-662-53887-6_3
13. Galbraith, S.D., Vercauteren, F.: Computational problems in supersingular elliptic curve isogenies. IACR Cryptology ePrint Archive 2017, 774 (2017), http://eprint.iacr.org/2017/774
14. H.Silverman, J.: The arithmetic of elliptic curves, vol. 106. Springer Science & Business Media, 2 edn. (2009)
15. Jalali, A., Azarderakhsh, R., Kermani, M.M., Jao, D.: Supersingular isogeny diffie-hellman key exchange on 64-bit ARM. IEEE Transactions on Dependable and Secure Computing PP(99), 1–1 (2017)

16. Jao, D., Feo, L.D.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings. pp. 19–34 (2011), https://doi.org/10.1007/978-3-642-25405-5_2
17. Karatsuba, A., Ofman, Y.: Multiplication of many-digital numbers by automatic computers. Proc. of the USSR Academy of Sciences 145, 293–294 (1962)
18. Kirkwood, D., Lackey, B.C., McVey, J., Motley, M., Solinas, J.A., Tuller, D.: Failure is not an option: Standardization issues for post-quantum key agreement. In: Talk at NIST workshop on Cybersecurity in a Post-Quantum World. vol. 2 (2015), https://www.nist.gov/itl/csd/ct/post-quantum-crypto-workshop-2015.cfm
19. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. pp. 388–397 (1999), https://doi.org/10.1007/3-540-48405-1_25
20. 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 2016, Milan, Italy, November 14-16, 2016, Proceedings. pp. 88–103 (2016), https://doi.org/10.1007/978-3-319-48965-0_6
21. Lei, X., Liao, X.: NTRU-KE: A lattice-based public key exchange protocol. IACR Cryptology ePrint Archive 2013, 718 (2013), http://eprint.iacr.org/2013/718
22. Liu, Z., Großschädl, J.: New speed records for montgomery modular multiplication on 8-bit AVR microcontrollers. In: Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings. pp. 215–234 (2014), https://doi.org/10.1007/978-3-319-06734-6_14
23. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. Deep Space Network Progress Report 44, 114–116 (Jan 1978)
24. Menezes, A., Okamoto, T., Vanstone, S.A.: Reducing elliptic curve logarithms to logarithms in a finite field. IEEE Trans. Information Theory 39(5), 1639–1646 (1993), https://doi.org/10.1109/18.259647
25. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of computation 44(170), 519–521 (1985)
26. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Mathematics of computation 48(177), 243–264 (1987)
27. Schneider, T., Moradi, A.: Leakage assessment methodology - A clear roadmap for side-channel evaluations. In: Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings. pp. 495–513 (2015), https://doi.org/10.1007/978-3-662-48324-4_25
28. Scott, M.: Fast machine code for modular multiplication (1995), http://www.compapp.dcu.ie/research/CA_Working_Papers/wp95.html
29. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Review 41(2), 303–332 (1999), https://doi.org/10.1137/S0036144598347011
30. Teske, E.: An elliptic curve trapdoor system. J. Cryptology 19(1), 115–133 (2006), https://doi.org/10.1007/s00145-004-0328-3
31. Vélu, J.: Isogénies entre courbes elliptiques. CR Acad. Sci. Paris Sér. AB 273, A238–A241 (1971)
32. Washington, L.C.: Elliptic curves: number theory and cryptography. CRC press, 2 edn. (2008)