Resistance of Isogeny-Based Cryptographic Implementations to a Fault Attack

Élise Tasso[1], Luca De Feo[2], Nadia El Mrabet[3], and Simon Pontié[1]

[1] CEA-Leti, Université Grenoble Alpes, F-38000 Grenoble, France
CEA-Tech, Centre CMP, Équipe Commune CEA Tech - Mines Saint-Étienne, F-13541 Gardanne, France
{elise.tasso2,simon.pontie}@cea.fr
[2] IBM Research, Zürich, Switzerland
cosade21@defeo.lu
[3] Mines Saint-Étienne, CEA-Tech, Centre CMP, F-13541 Gardanne, France
nadia.el-mrabet@emse.fr

Abstract. The threat of quantum computers has sparked the development of a new kind of cryptography to resist their attacks. Isogenies between elliptic curves are one of the tools used for such cryptosystems. They are championed by SIKE (Supersingular isogeny key encapsulation), an "alternate candidate" of the third round of the NIST Post-Quantum Cryptography Standardization Process. While all candidates are believed to be mathematically secure, their implementations may be vulnerable to hardware attacks. In this work we investigate for the first time whether Ti's 2017 theoretical fault injection attack is exploitable in practice. We also examine suitable countermeasures. We manage to recover the secret thanks to electromagnetic fault injection on an ARM Cortex A53 using a correct and an altered public key generation. Moreover we propose a suitable countermeasure to detect faults that has a low overhead as it takes advantage of a redundancy already present in SIKE implementations.

Keywords: Post-quantum Cryptography · SIKE · Elliptic Curve · Isogeny · Fault Injection Attack.

1 Introduction

Starting in 1994 with Shor's factorization algorithm [24], quantum computers have been shown to threaten classic asymmetric cryptography. Thus the National Institute of Standards and Technology launched the Post-Quantum Cryptography Standardization Process in December 2016 [20]. Research teams worldwide had begun to work on algorithms that can be implemented on classical computers but resist quantum computer attacks before and thus continued to study encryption and signature protocols as required by the NIST. These protocols are based on various mathematical tools, including lattices, error correcting codes, multivariate polynomial equations, hash functions and isogenies between elliptic curves. We will focus here on the Supersingular Isogeny Key Encapsulation (SIKE) [14], the only candidate based on isogenies. More precisely, SIKE is a key encapsulation mechanism (KEM) based on the Supersingular Isogeny Diffie-Hellman (SIDH) key exchange proposed by Jao and De Feo in 2011 [15]. It is now an alternate candidate in the third round of the standardization process, meaning that it is deemed promising enough by the NIST to pursue research on it. It has indeed the smallest key size by far among the third round candidates [1], but is comparatively slow. Like the other candidates, SIKE is believed to be mathematically secure, but vulnerabilities may appear in its implementations. A further interesting characteristic is its regularity, which makes hardware attacks more challenging. Hardware attacks assume that the attacker has physical access to the device where the algorithm is being executed. There are two categories of such attacks. In a passive attack, the attacker is only able to observe the execution of the algorithm on the target. They may measure the computation time, the power consumption or the electromagnetic emanation of the circuit and try to deduce information about the keys or exchanged messages. These are called side-channel attacks. In an active attack, they may disrupt the execution of the algorithm by creating power or clock glitches, illuminating the target with a laser beam or injecting an electromagnetic field to get information. These are called fault attacks. Attacks of both kinds have been found to affect SIKE.

Galbraith et al. discussed in 2016 some attacks based on the leakage of partial knowledge of the key [11]. In 2017, Gélin and Wesolowski presented a loop-abort attack by injecting a random fault on in the isogeny computation loop counter during the computation of the shared key [12]. There already exists countermeasures to avoid loop abort attacks, for instance as presented in [22]. The same year, Ti published a paper about another way to do a fault attack on a static key variant of SIDH [27]. Koziel et al. [17] proposed a refined power analysis on the three-point Montgomery differential ladder during the shared secret computation and during the isogeny computation. Countermeasures are proposed for both. In 2018, Koppermann et al. [16] also attacked the shared secret computation, but with a differential power attack on the scalar multiplication during the kernel generator computation. A countermeasure to such an attack is the randomisation of the projective representations of the points [4]. The latest known attack is by Zhang et al. [30]: a differential power attack and differential electromagnetic attack, also on the scalar multiplication during the isogeny kernel generator computation. We classify these attacks in different categories as seen in Table 1 below.

In the implementation of SIKE we can distinguish two phases: a first one that uses only classical elliptic curve cryptography algorithms, where a scalar multiplications on elliptic curve points is performed, and a second one that performs isogeny computations and evaluations. We classify attacks depending on their target, the first or the second phase. To the best of our knowledge, there has not been any experimentally verified attack specific to the isogeny phase, thus we want to investigate whether Ti's 2017 theoretical fault injection

Table 1. Classification of known hardware attacks on SIKE depending on their type, if they are experimentally verified or not, and depending on the part of the algorithm that is attacked.

Gélin et al., 2017 [12]	fault injection	simulated	isogeny
Koziel et al., 2017 [17]	side-channel attack	theoretical	scalar multiplication, isogeny
Ti, 2017 [27]	fault injection	theoretical	isogeny
Koppermann et al., 2018 [16]	side-channel attack	experimentally verified	scalar multiplication
Zhang et al., 2020 [30]	side-channel attack	experimentally verified	scalar multiplication

attack [27] is exploitable in practice. The goal of this attack is to recover the static key, which is a private key used more than once over a long period of time. After reviewing some background information about isogenies and SIDH/SIKE, we will present Ti's attack. Then, we shall describe the experimental setups used in our investigation to finally analyse possible countermeasures.

Contributions We provide the first experimental realization of Ti's 2017 theoretical fault attack by carrying out an attack campaign in a laboratory. We induced faults on the SIKE round 3 implementation optimized for ARM64 on a system on chip (SoC) with four cortex A53 cores by using electromagnetic fault injection. This provides an experimental understanding of the threat on SIKE caused by Ti's attack. At last, we propose two new countermeasures against this attack: one concerns the protocol and the other is a verification at the end of the public key generation.

2 Preliminaries

In this section we are going to present a few mathematical and cryptographic notions, in particular, the SIDH cryptosystem and the key encapsulation in SIKE that will be of use when analysing Yan Bo Ti's attack in Section 3. We shall start with a primer on isogeny-based cryptography.

2.1 Isogenies between Elliptic Curves

For basic definitions concerning elliptic curves and isogenies between elliptic curves, we refer the reader to [25]. An introduction to isogenies as used in cryptography can be found in [8].

The elliptic curves used in SIKE are represented as Montgomery curves [18].

Definition 1. *Let K be a finite field such that $char(K) \neq 2$ and $A, B \in \mathbb{F}_{p^2}$ such that $B(A^2-4) \neq 0$. The Montgomery (elliptic) curve $E_{A,B}$ consists of a point at infinity O and the set of points $(x, y) \in \mathbb{F}_{p^2}$ such that*

$$By^2 = x^3 + Ax^2 + x.$$

In particular, $B = 1$ in SIKE.

One advantage of Montgomery curves is to provide algorithms to compute scalar multiplications more efficiently [18,7]. Indeed, let us consider the multiplication by an integer k on an elliptic curve E:

$$\begin{aligned}[k] : E &\to E \\ P &\mapsto \underbrace{P + P + ... + P}_{\text{k times}}.\end{aligned}$$

The automorphism of E $\ominus : P \mapsto -P$ can be used to quotient E and thus get a map $x : E \to \mathbb{P}^1 \cong E/\langle\ominus\rangle$. We have $x(P) = x(Q)$ if and only if $P = Q$ or $P = -Q$. It is then possible to define an induced multiplication on $\mathbb{P}_1$ for all $k \in \mathbb{Z}$ such that $x(P) \mapsto x([k]P)$. Hence, instead of performing scalar multiplications on points of the curve, one can use the x-coordinates of the points only. Montgomery provides more efficient formulas for point multiplication using the x-coordinates in. For efficiency reasons, this coordinate $x = X/Z$ of Montgomery curves is represented projectively with $(X : Z)$, see [18,7].

As shown in [6], the A coefficient of a Montgomery curve can be recovered using three distinct non-zero x-coordinates of points P, Q and R such that $R = P - Q$ with the following formula (see also algorithm `cfpk` in [14]):

$$A = \frac{(1 - x_P x_Q - x_P x_R - x_Q x_R)^2}{4 x_P x_Q x_R} - x_P - x_Q - x_R. \tag{1}$$

In SIKE, an elliptic curve is encoded by such an x-coordinate triplet.

An invariant can be defined for these elliptic curves [7].

Definition 2. *Let E be a Montgomery curve as above. Then the j-invariant of E is*

$$j(E) = \frac{256(A^2 - 3)^3}{A^2 - 4}.$$

This allows us to create equivalence classes of elliptic curves, see [25, § III.1].

Proposition 1. *Two elliptic curves are isomorphic over the algebraic closure of their definition field if and only if they have the same j-invariant.*

Maps can be defined between these equivalence classes. Let E and F be two elliptic curves defined over a finite field K. An isogeny ϕ between E and F is a non-trivial group morphism between E and F. We will often use the "morphism aspect" of this definition i.e. that for all points P and Q on E,

$$\phi(P + Q) = \phi(P) + \phi(Q).$$

Moreover, we consider in SIKE a special kind of isogenies called separable isogenies that are uniquely determined by their kernel. This kernel C is necessarily finite. Knowing C, there are formulas by Vélu [28] showing how to compute the equation of the target elliptic curve of the isogeny, denoted by E/C. Hence, referring to the kernel of an isogeny amounts to referring to the isogeny itself. As only separable isogenies appear in SIKE, we define the degree $\deg(\phi)$ of ϕ as the size of its kernel. We shall now define the dual of an isogeny.

Definition 3 (Dual isogeny [25, § III.6]). *Let $\phi : E \to F$ be an isogeny. Then there is a unique isogeny $\hat{\phi} : F \to E$ called the dual isogeny of ϕ such that*

$$\hat{\phi} \circ \phi = [\deg(\phi)]_E \text{ and } \phi \circ \hat{\phi} = [\deg(\phi)]_F.$$

The dual isogeny has the following properties.

- For all isogenies $\phi : E \to F$ and $\psi : F \to G$, we have $\widehat{\psi \circ \phi} = \hat{\phi} \circ \hat{\psi}$.
- $\deg(\hat{\phi}) = \deg(\phi)$
- $\hat{\hat{\phi}} = \phi$

Having described the necessary mathematical tools, we will now present the scheme that is at the crux of SIKE.

2.2 The SIDH Key Exchange

The supersingular isogeny Diffie-Hellman (SIDH) key exchange [15] is a Diffie-Hellman-like key exchange that is a building block of SIKE.

Alice and Bob are two parties who would like to share a key. Let e_2 and e_3 be two positive integers that define a prime p such that $p = 2^{e_2}3^{e_3}f \pm 1$, f being a small cofactor (p is of that form in the SIKE specifications [14]). Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$. Let P_2, $Q_2 \in E[2^{e_2}]$ (i.e. $2^{e_2}P_2 = 2^{e_2}Q_2 = O$) and P_3, $Q_3 \in E[3^{e_3}]$ be bases of these respective torsions, R_2 such that $R_2 = P_2 - Q_2$ and $R_3 = P_3 - Q_3$. These parameters are public. Alice and Bob both have a secret key which is a uniformly distributed random integer, respectively $\mathrm{sk}_2 \in [0, 2^{e_2} - 1]$ and $\mathrm{sk}_3 \in [0, 3^{e_3} - 1]$. The associated secret isogenies ϕ_A and ϕ_B are such that

$$\mathrm{Ker}(\phi_A) = \langle P_2 + \mathrm{sk}_2 Q_2 \rangle \text{ and } \mathrm{Ker}(\phi_B) = \langle P_3 + \mathrm{sk}_3 Q_3 \rangle.$$

We denote by E_A (respectively E_B) the target curve of ϕ_A (respectively ϕ_B), E_{AB} the target curve of ϕ'_A with kernel $\langle \phi_B(P_2) + \mathrm{sk}_2\phi_B(Q_2) \rangle$ and E_{BA} the target curve of ϕ'_B with kernel $\langle \phi_A(P_3) + \mathrm{sk}_3\phi_A(Q_3) \rangle$. The following diagram shows how the shared key is constructed.

First, Alice and Bob generate each their private keys, sk_2 and sk_3. Then, they compute their public keys, respectively $(x_{\phi_A(P_3)}, x_{\phi_A(Q_3)}, x_{\phi_A(R_3)})$ and $(x_{\phi_B(P_2)}, x_{\phi_B(Q_2)}, x_{\phi_B(R_2)})$ and exchange them. At last, they both determine the j-invariant of E_{AB} or E_{BA}, depending on the party. It can be shown that E_{AB} and E_{BA} are isomorphic, thus $j(E_{AB}) = j(E_{BA})$, which is used as shared key.

Now we shall see how the public keys are computed in the key generation steps thanks to Algorithm 1 (Algorithm $\texttt{isogen}_l$ in [14, § 1.3.6]).

$$\begin{tikzcd}
E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E_A \arrow[d, "\phi'_B"] \\
E_B \arrow[r, "\phi'_A"'] & E_{BA} \simeq E_{AB}
\end{tikzcd}$$

Fig. 1. The SIDH key exchange [15].

```
   Input  : A private key $\mathrm{sk}_A$.
   Output: A public key $\mathrm{pk}_A$.
1  $x_S \leftarrow x_{(P_2+\mathrm{sk}_2 Q_2)}$ // ladder3pt [14, App. A, Alg. 8]
2  $(x_1, x_2, x_3) \leftarrow (x_{P_3}, x_{Q_3}, x_{R_3})$ // init_basis[4]
3  for $i$ from 0 to $e_2 - 1$ // Tree traversal loop
4   do
5      (a) Compute an 2-isogeny
6
           $\phi_i : E_i \to E'$
           $(x, ...) \mapsto (f_i(x), ...)$
7      such that $\mathrm{Ker}(\phi_i) = \langle 2^{e_2-i-1} S \rangle$.
8      (b) $E_{i+1} = E'$
9      (c) $x_s = f_i(x_S)$
10     (d) $(x_1, x_2, x_3) \leftarrow (f_i(x_{x_1}), f_i(x_{x_2}), f_i(x_{x_3}))$
11 end
12 Return $(x_1, x_2, x_3)$.
```

Algorithm 1: SIKE public key computation with 2-isogenies [14, § 1.3.6].

Remark 1. Optimized implementations of SIDH use a slightly more involved algorithm, performing scalar multiplications and isogeny evaluations according to a predetermined binary tree topology in the tree traversal loop starting at Line 3 of Algorithm 1, as described in [14, § 1.3.8]. The choice of algorithm does not impact the feasibility of Ti's attack.

In the following section, we present key encapsulation in SIKE using the concepts of SIDH.

2.3 SIKE

SIKE is a key encapsulation mechanism (KEM). A KEM is used to securely exchange a symmetric key for data encryption using asymmetric cryptography. Figure 2 shows its different elements. It is composed of three algorithms:

[4] https://github.com/microsoft/PQCrypto-SIDH/blob/97c1/src/sidh.c#L10

1. Keygen generates a pair of long term secret and public keys.
2. Encaps takes as input the public key and outputs a random symmetric key K and an encapsulation c of said key.
3. Decaps takes as input the secret key, the public key and c and outputs the symmetric key K.

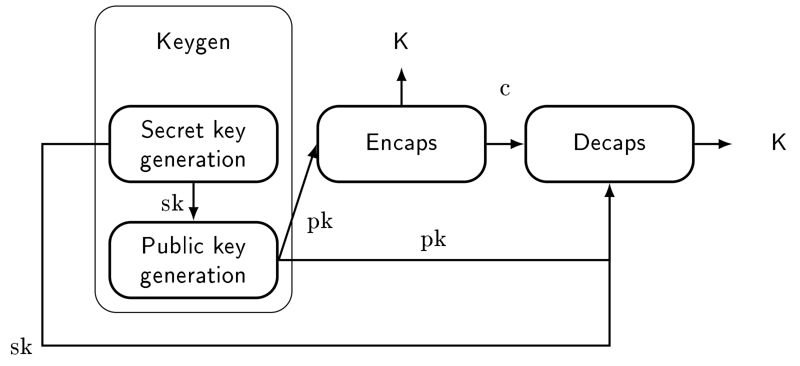

Fig. 2. Description of a Key Encapsulation Mechanism (KEM).

We use the notions of Section 2.2 to explain how the concepts of SIDH are used in SIKE. In SIKE, an IND-CPA PKE scheme is built using the same operations as in SIDH, similar to how the "hashed ElGamal" PKE is obtained from the Diffie–Hellman key exchange. The SIKE IND-CCA KEM is built from the PKE scheme using a transformation of Hofheinz, Hövelmanns and Kiltz [13]. The secret key and the public key generation in Keygen are performed exactly as in SIDH. Encaps and Decaps are based on operations from SIDH but also use hash functions and XOR operations. An ephemeral scalar and the associated isogeny are generated in Encaps. This scalar cannot be called "secret" key because it will be recovered by the other party in Decaps. Each time the parties want to compute a shared key K, this scalar is generated anew. However, this is not necessarily the case for the key material in Keygen. Ti's attack takes advantage of this static key, as we will see in the next section. Using this ephemeral scalar and the public key pk, a shared secret j (a j-invariant) is computed. The "secret" scalar and a hash of j are then used to compute the symmetric key K and an encapsulated key c as a ciphertext. In Decaps, the ciphertext is then decrypted with the secret key to recover K and K is validated by recomputing the "secret" scalar and the associated isogeny.

Remark 2. Since Round 2 [14], SIKE specifies two variants, called *uncompressed* and *compressed*. For efficiency reasons, the roles of 2 and 3 are swapped between the two: in the uncompressed version, the public key is computed using 3-isogenies, while the ciphertext is computed using 2-isogenies. In the compressed version, key generation is done by computing 2-isogenies and encapsulation by computing 3-isogenies. The key material used inside the encapsulation is generated anew at each call of Encaps, while the public key produced by Keygen is generated once and can be reused for multiple encapsulations.

3 Ti's Theoretical Fault Attack

As seen in Table 1, there are no experimental validations of attacks on SIKE specific to isogenies. Ti's attack imposes few constraints on the faults to inject, thus it is a good candidate for practical exploitation, even on systems where controlling the produced faults is difficult. This is why we decided to tackle Ti's attack and put it in practice on a modern SoC.

First, we will explain present Ti's attack scenario. The overview of the attack is described in Figure 3.

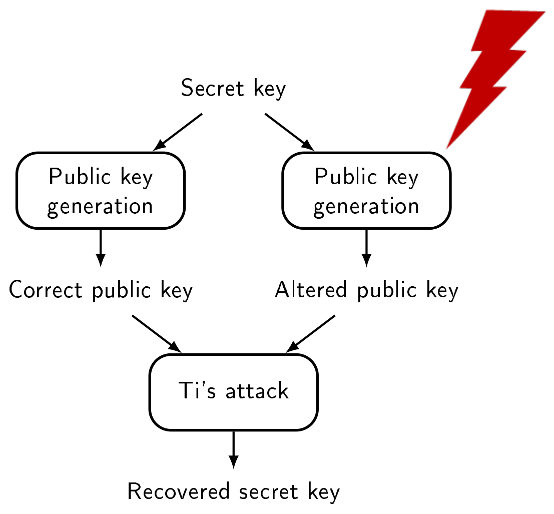

Fig. 3. Schematic representation of Ti's attack scenario.

Attack Scenario We consider an attack scenario where the victim holds a public-private SIKE key pair, and where the attacker is permitted to retrigger public key generation **from the same secret**, injecting a fault to produce an altered public key (with some probability of success that we shall determine later). To mount the attack, a single altered public key is sufficient.

In a KEM, Keygen generates a fresh secret key and outputs the public key at the same time, hence it is a design mistake to enable a regeneration of the public key. It is however difficult to ensure that all developers will respect the KEM

API and avoid generating more than once a public key from the same secret. Hence a countermeasure intrinsic to the public key generation would be useful.

Moreover, avoiding secret reuse is simply not possible in a multipartite exchange [2] where Bob has to use his secret key to generate multiple triplets of points to send to Alice and Charlie, for instance. Hence if the attacker injects a fault during the computation of the triplet Bob wants to send to Alice, they can still recover a correct triplet of Bob's by intercepting the communication from Bob to Charlie. Thus a countermeasure intrinsic to public key generation is strictly necessary for multipartite key exchange (Figure 4).

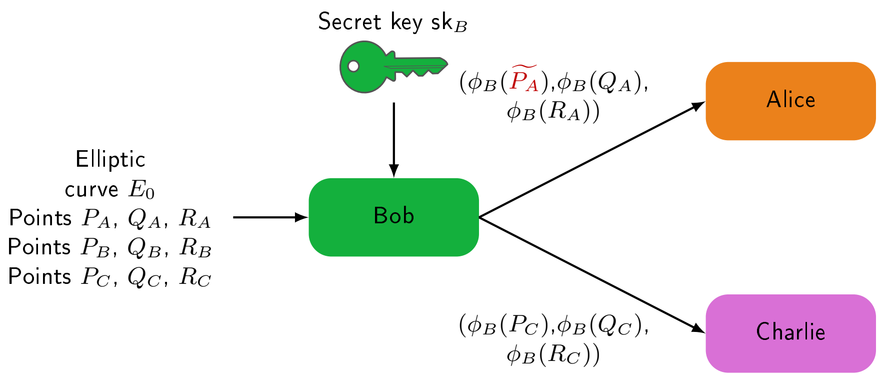
Secret key sk_B
$(\phi_B(\widetilde{P_A}),\phi_B(Q_A),$
$\phi_B(R_A))$
Alice
Elliptic
curve E_0
Points P_A, Q_A, R_A
Points P_B, Q_B, R_B
Points P_C, Q_C, R_C
Bob
$(\phi_B(P_C),\phi_B(Q_C),$
$\phi_B(R_C))$
Charlie

Fig. 4. Ti's attack in a multipartite key exchange setting.

In Ti's article, the starting curve E_0 is defined on $\mathbb{F}_{p^2}$ with $p = 2^{e_2}3^{e_3}f \pm 1$ where f is a small cofactor as in Section 2.2. Normally, when the key generation is not under attack, the images of three fixed public points P_3, Q_3 and R_3 by the secret isogeny ϕ are computed to get the public key (line 10 of Algorithm 1). The attacker will force the computation of the image of a random point by ϕ. The result is an altered point $\phi(\widetilde{P_3})$, $\phi(\widetilde{Q_3})$ or $\phi(\widetilde{R_3})$.

Ti uses the following lemma to show that it is possible to recover the secret key via this point.

Lemma 1. *Let p be a prime number such that $p = 2^{e_2}3^{e_3}f \pm 1$, where f is a small positive integer and e_2 and e_3 are positive integers such that $2^{e_2} \approx 3^{e_3}$, the same form as in the SIKE specifications [14]. Let E_1, E_2 and E' be supersingular elliptic curves defined over $\mathbb{F}_{p^2}$. Suppose that $\phi : E_1 \to E_2$ is an isogeny of degree 2^{e_2} with a cylic kernel and let P and Q be generators of $E_1[2^{e_2}]$. For any $X \in E_1[2^{e_2}]$, let $\psi : E_2 \to E'$ be an isogeny with kernel generated by $\phi(X)$. Then there exists an isogeny $\theta : E' \to E_1$ of degree 2^{ϵ} where ϵ is a positive integer such that $\epsilon \leqslant e_2$ and $\hat{\phi} = \theta \circ \psi$.*

Proof. See [27].

```
Input  : $\phi(P_3)$, $\phi(Q_3)$ or $\phi(R_3)$ and
         $M$, an altered point that can be $\phi(\widetilde{P_3})$, $\phi(\widetilde{Q_3})$ or $\phi(\widetilde{R_3})$.
Output: $\hat{\phi}$
1 $\lambda = 3^{e_3} f$
2 Compute $A_{e_2}$, the parameter of the final Montgomery curve $E_{A_{\mathrm{acc}}}$, using
    algorithm cfpk of section 1.2.1 in [14].
3 $T = \lambda M$ on $E_{e_2}$
4 if $\mathrm{ord}(T) = 2^{e_2}$ then
5     $\langle T \rangle = \ker(\hat{\phi})$
6 else
7     Brute force for $\theta$ such that $\hat{\phi} = \theta \circ \psi$
```

Algorithm 2: Ti's key recovery algorithm.

This lemma is translated to Algorithm 2. In it, we highlight the case where the candidate dual isogeny ψ has maximum order, so that θ is the identity map (Figure 5). The general case is represented in Figure 6.

$$\begin{tikzcd} E_0 \arrow[r, bend left, "\phi"] & E_A \arrow[l, bend left, "\psi"] \end{tikzcd}$$

Fig. 5. T has maximal order.

$$\begin{tikzcd} E_0 \arrow[rr, "\phi"] & & E_A \arrow[dl, "\psi"] \\ & E' \arrow[ul, "\theta"] & \end{tikzcd}$$

Fig. 6. T does not have maximal order.

If the kernel generator T has maximal order, i.e. 2^{e_2}, then the isogeny with kernel $\langle T \rangle$ has degree 2^{e_2} and is the dual of ϕ since an isogeny and its dual have the same degree. If, however, T does not have maximal order, an additional isogeny θ of degree $2^{e_2 - \mathrm{ord}(T)}$ will be needed so that $\theta \circ \psi$ is the dual of ϕ. Recovering ϕ knowing the dual is then possible using its definition.

Remark 3. What is the size of the search space for θ's kernel? Do note that we study here a public key generation with a secret isogeny of degree 2^{e_2}. First, to be able to carry out the attack, we need the altered point to be on E_0. Assuming the altered x-coordinate (recall we are using Montgomery curve arithmetic, see Section 2) behaves like a random element of $\mathbb{F}_{p^2}$, the probability that it corresponds to a point X on E_0 is approximately $\frac{1}{2}$. Moreover, as E_0 is supersingular, $E_0(\mathbb{F}_{p^2}) \simeq (\mathbb{Z}/(p+1)\mathbb{Z})^2$. Hence there is a basis (A, B) of $E_0(\mathbb{F}_{p^2})$ such that A and B are of order $p+1$ and $3^{e_3} B = S$, where $S \in E_0$ is the secret kernel generator of ϕ and $\mathrm{ord}(S) = 2^{e_2}$. Let $X \in E_0(\mathbb{F}_{p^2})$ such that $X = aA + bB$ where $a, b \in \mathbb{Z}/(p+1)\mathbb{Z}$. Then $T = 3^{e_3}\phi(X) = a3^{e_3}\phi(A) + b\phi(3^{e_3}B)$ and thus $T = a\phi(3^{e_3}A)$ using the definition of B. The order of $3^{e_3}A$ is 2^{e_2}, and so the order of $\phi(3^{e_3}A)$ is also 2^{e_2}. The order of T is then $2^{e_2 - \mathrm{val}_2(a)}$, where $2^{\mathrm{val}_2(a)}$ is the maximum power of 2 dividing a. The order of T is maximal if $\mathrm{val}_2(a) = 0$ i.e.

if a is odd. The probability of a being odd is $\frac{1}{2}$. T is of order 2^{e_2-1} if a is even but not divisible by 4. The probability for this is $\frac{1}{4}$. In the worst case, $\mathrm{val}_2(a) = e_2$, and the brute force to compute θ is the longest. But the probability of such an event is low: only $\frac{1}{2^{e_2+1}}$. Hence the probability to have no θ to compute is $\frac{1}{4}$, to have a θ of degree 2 to compute is $\frac{1}{8}$ and the worst case, which corresponds to a brute force of the dual isogeny, has probability $\frac{1}{2^{e_2+2}}$ to occur. The expected value of the search space for θ is then given by $\sum_{i=0}^{e_2} \frac{1}{2^{i+1}} \cdot 2^i = \frac{e_2+1}{2}$, which is quite low. All in all, there is a 50% chance that the attack will fail, a 25% chance that the attacker will not need to search for θ, a 12.5% chance that he will have to find θ among the isogenies of degree 2^1 and a 6.25% that he will have to find θ among the isogenies of degree 2^2, etc... Thus it is clear that the probability for the attacker to succeed within a reasonable time is close to 50%. Do note that the reasoning is similar if the attack is performed to find a secret 3^{e_3}-isogeny, and that there is again a 50% chance for the attack to fail and a nearly 50% chance for it to succeed in a reasonable amount of time. These results are valid for a fault at line 2 of Algorithm 1. A fault on x_1, x_2 or x_3 after executing line 10 of Algorithm 1 for the i-th time will modify the probability distribution of $\mathrm{val}_2(\mathrm{ord}(T))$. Indeed, T will have at most a 25% chance to have order 2^{e_2-i} (or 3^{e_3-i}, depending on the case): the distribution is the same, but shifted by i on the $\mathrm{val}_2(\mathrm{ord}(T))$ axis.

Remark 4. Some implementations of SIKE use compressed public keys (for instance [14]). There are different variants of the compression available, for instance in [3,5,19,21,29]. We focus on the one presented in [19], which is the one used in the round 3 submission of SIKE [14]. There are three main steps to compress a public key:

1. Compute the basis of the 3^{e_3}-torsion of E_A.
2. Pull it back to E_0 with the dual of the secret isogeny.
3. Compute the coordinates of P_3, Q_3 and R_3 in that basis on E_0.

These coordinates are the same as the coordinates of $\phi(P_3)$, $\phi(Q_3)$ and $\phi(R_3)$ in the 3^{e_3}-torsion of E_A, and they are the compression of the public key. In this case, we would have to adapt Ti's attack to compute the image of incorrect points by the dual instead of images of the basis. Thus we would have to create a fault during the computation of the image of the basis by the dual. The impact of the different compression methods on the feasability of the attack should thus be studied in the future. In the rest of this work, we will focus on the non-compressed version of SIKE.

Remark 5. Ti's attack is possible on both 2-isogenies and 3-isogenies. In practice, the attacker will attack 2-isogenies or 3-isogenies depending on what is used in Keygen. We chose to focus our experiments on the attack of only one type of isogenies. When altering point initializations with fault injections, an effect can be to uninitialize some words. In the studied SIKE implementation, the allocated memory space for a point is first filled with zero words. All these zero words are then overwritten by data from constant parameter set coordinates (`init_basis` at Line 2 of Algorithm 1). The expected effect of fault injection is to skip an instruction during this overwriting to obtain altered point coordinates. An overwriting by zero of a zero word has no effect thus it is easier to alter points initialized with few zero words. When looking at the non-compressed SIKE p434 parameter set, one notices that the points P_3 and Q_3 in the ternary torsion have more zero words in their coordinates so it is more difficult to alter these points. Hence we decided to attack a public key generation with computations of the images of P_3 and Q_3 by 2-isogenies, starting by the non-compressed version, with the goal of attacking the compressed version later.

4 Experimental Setups

Before performing real-life electromagnetic fault attacks, we decided to simulate these attacks using software only. Indeed, fault injection attacks are long and complex to carry out [10], thus we chose to validate the attack with a simulation before the laboratory experiments. There were two steps:

– first, we used Sagemath [26] to simulate fault injection **and** to recover the secret isogeny with an implementation of Algorithm 2 and
– then we emulated the target in C and injected the fault by debugging, while recovering the secret with the same Sage implementation of Algorithm 2.

We are going to subsequently describe the second step, as it is the most realistic simulation of the two, and then the experimental setup for the real-life fault injection.

4.1 Fault Injection Simulation with C

For that second step, we simulate the fault injection by debugging with GDB the optimized round 3 ARM64 implementation of SIKE [14] with curve p434 (non compressed version) using QEMU as an emulator of our target. The emulator is a tool to study the execution of the ARM64 application on an Intel processor. We compiled the SIKE sources from the path Additional_Implementations/ARM64/SIKEp434 in the round 3 SIKE archive with gcc 7.2.1 and the optimization level 3 [14][5].

First, we programmed a tool to generate the key material, i.e. the private key and the public key. Our public key generation tool is a simple encapsulation of the function `EphemeralKeyGeneration_A` of said ARM64 SIKE implementation. Then we could perform the fault injection simulation, which consists in debugging the public key generation program executed by the emulator with debugger GDB. The input is Alice's secret key. 300 fault attacks are launched by executing `EphemeralKeyGeneration_A` and skipping a different instruction

[5] The original ARMv8-A software implementation is also available in the SIDH Microsoft library at `https://github.com/microsoft/PQCrypto-SIDH/tree/f43c9f74`.

for each experiment. We only observed the program's behaviour when skipping one of the first 300 instructions. Indeed, the "instruction skip" fault model is easy to implement in GDB with the command "`set $pc=$pc+4`" and is a very simplified but satisfying model before the implementation of a real attack, as shown in [9] and [23].

This fault injection simulation was done for two different random secret keys. Among the 300 instructions that were skipped, 85 are particularly interesting because they are load/store pairs of instructions that copy the coordinates x_{P_3}, x_{Q_3} and x_{R_3} in the accumulator of line 2 of Algorithm 1. They correspond to the second `init_basis` function call at the beginning of function `EphemeralKeyGeneration_A` in `sidh.c`[6]. 28 of these instructions have no impact on the public key when skipped. The 57 remaining instructions modify the public key when skipped. Out of these 57, 8 instructions yield an x that does not correspond to a point on the target curve because it is impossible to compute y. For the 49 other instructions, it is possible to compute y. Table 2 shows the order of T and the attack successes for these 49 instructions. Recovering the secret was limited to points T with $\log_2(\text{ord}(T)) >= 210$ to limit brute force computations and, as stated in theory in Remark 3, we notice that the obtained $\log_2(\text{ord}(T))$ are close to e_2. Thus 45 to 48 instructions yield the secret key, and this shows that different faults enable secret key recovery, and is encouraging for the set up of a laboratory fault attack campaign.

Table 2. Number of altered instructions during the second call to `init_basis` yielding the secret key.

$\log_2(\text{ord}(T))$	<208	209	210	211	212	213	214	215	216
Instructions yielding T for secret key #1	0	1	0	1	3	4	10	21	9
Instructions yielding secret key #1	/	/	0	1	3	4	10	21	9
Instructions yielding T for secret key #2	0	4	0	3	3	2	14	16	7
Instructions yielding secret key #2	/	/	0	3	3	2	14	16	7

Sensitive instructions yielding the secret key were also identified outside of the second call to `init_basis`. We focused the study on the first 300 instructions of the function `EphemeralKeyGeneration_A`. 11 additional instructions were identified after calls to the `init_basis` functions. A lot of instruction skips generated altered points that could not be exploited to recover the secret. For example, altering the points used to compute the secret kernel generates three altered points in the public key but this alteration of the secret isogeny cannot be exploited by the attack. Table 3 shows the order of T computed from altered points. Each instruction skip that yields the secret key generates a unique altered point. Again, as expected in Remark 3, the obtained $\log_2(\text{ord}(T))$ are close to e_2.

[6] `https://github.com/microsoft/PQCrypto-SIDH/blob/f43c9f74/src/sidh.c#L51`

Table 3. Number of altered instructions in the 300 first instructions of `EphemeralKeyGeneration_A` yielding the secret key.

$\log_2(\text{ord}(T))$	<208	209	210	211	212	213	214	215	216
Altered point yielding T for secret key #1	1	2	0	1	4	6	19	53	136
Instructions yielding secret key #1	/	/	0	1	3	5	12	27	11
Altered point yielding T for secret key #2	1	4	0	4	3	4	31	46	128
Instructions yielding secret key #2	/	/	0	3	3	3	19	19	9

4.2 Carrying Out the Fault Injection in a Laboratory

After having shown that it is possible to simulate Ti's attack, we now present an experimental version. The target is a system on chip (SoC) equipped with four ARM Cortex-A53 cores and including a Yocto Linux operating system. While it is difficult to fault a chosen instruction because of a poorly predictable latency of execution in SoCs [10], we have seen in Section 3 and with the simulation in Section 4.1 that we only need to fault the beginning of the key generation and do not need to choose precisely the time for the fault injection. Thus we can expect a successful attack on such a target. As seen in Section 3, we specifically target key generation, thus we only take this part of the code from the optimized version for ARMv8-A of the SIKE round 3 submission, ignoring decapsulation. We force computations to run on a single CPU of the quad-core, and we fix the CPU clock to the maximum frequency, i.e. 1.2 GHz. We choose electromagnetic injection to create faults, because it is relatively cheap, and because it does not require a complex sample preparation, like removing the circuit packaging. Our setup for the campaign can be seen in Figure 7.

To launch attacks, the control computer executes a campaign script that manages the communication with the target and the other devices (target power supply, oscilloscope, pulse generator and motorized stage). The key generation implemented on the target has been modified so that the state of a GPIO (general purpose input/output, here output pin) of the target changes just before the public key computation. The fault injection is then triggered by the target when this rising edge appears on the GPIO. The fault is induced by an electromagnetic disturbance generated with a tension pulse generator. The width in nanoseconds, the amplitude in Volts and the delay of the pulse, i.e. the time between the rising edge corresponding to the public key computation and the injection of said pulse can all be controlled. The tension pulse is then transmitted as an electromagnetic disturbance to the target through an electromagnetic probe. The probe can be moved with the motorized stage to find the optimal position for fault injection, that is to say the position where it is indeed possible to modify the execution of the algorithm and perform our attack. This induces unwanted currents inside the target. The results of the algorithm computations after injection may be affected and are retrieved and analysed by the computer. In case of application or kernel crash, the power supply is used to reboot the target.

During our campaign, the probe does not move and the pulse width is set at 6 ns. This position and the width are propitious for fault injection according

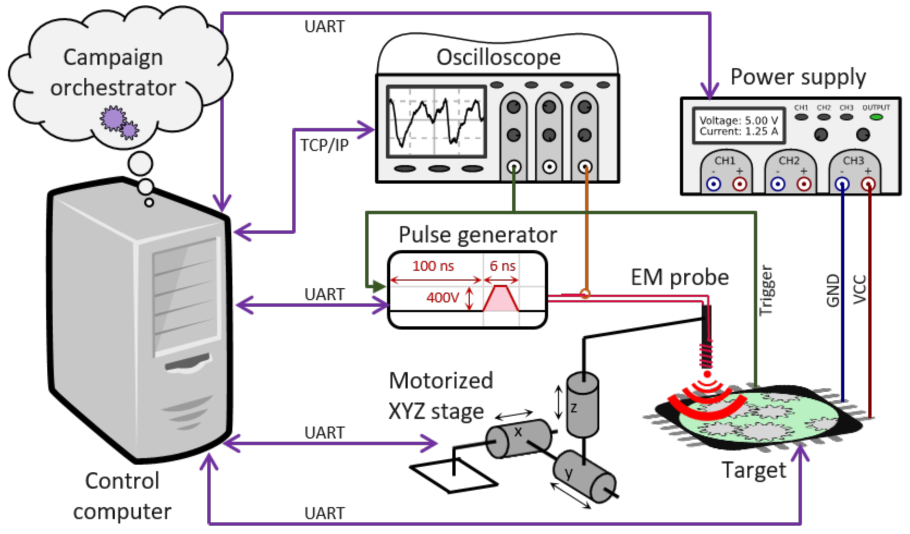

Fig. 7. Campaign setup.

to [10]. The amplitude varies from 300 to 360 V with a 20 V step and the delay between the start of the public key generation and the fault injection varies from 100 to 600 ns with a 20 ns step. Per such configuration, 10,000 attempts are made. Hence 1,040,000 attemps were made in total. The campaign lasted for 4.5 days.

4.3 Analysis

To analyse the results produced by either the simulation or the real fault attack, we use a Sagemath tool and follow the steps described in Section 3. In the proof of Ti's attack the order of the candidate dual generator can be maximal or not, and in the latter case, a brute force is necessary to get a θ isogeny to "go back" to the starting elliptic curve. To ensure that the computations do not take "too much time", i.e. that the number of candidates for θ when performing the brute force is small, the order of generator T should be as close to the maximum order, 2^{e_2}, as possible. In that case, the degree of θ (the size of its kernel) will be small. When computing T and checking its order, we will thus only keep points that yield generators T with a "nearly" maximum order, higher or equal to 2^{e_2-6}. Assuming that the altered point is on E_0 (Remark 3), meaning that we can compute T, the probability to get a point T with such order is more than 99%. We then compute a candidate ψ for the dual of the secret isogeny. Depending on the order of T, we will then determine θ or not, and then we will compute the images of P_3, Q_3 and R_3 by the reconstructed secret isogeny to get a reconstructed public key and check that the attack went well. Recovering the secret scalar associated to this secret isogeny is also possible because we know its kernel point and solving discrete logarithms is easy in the smooth order groups used by SIDH/SIKE.

4.4 Experimental Results

During the 4.5 days campaign, we obtained among 1,040,000 attempts:

- 8706 attempts producing at least one altered output point (0.84%) i.e. at least one point whose x-coordinate is different from the corresponding correct public key coordinate. At this point, we have not yet tested if these points are on the correct public key curve. Even if our goal is an alteration of the isogeny input point, other faults might arise and prevent us from recovering the secret, for example a corrupted secret kernel.
- 1780 attempts yielding the secret key (0.17% of all attempts). This represents 20.45% of attempts with faulted output point, which is nearly half of the estimated probability in Section 3. The main explanation for this lower probability must be that our probability estimation is based on one assumption: the x-coordinate of the input point of the isogeny is faulted. In practice, we also induced other faults. Table 4 shows $\log_2(\mathrm{ord}(T))$ with T from the faulted output point for the 1780 attempts (see line 3 of Algorithm 2). As anticipated, the order of most of these points is high (Table 6 shows the number of attempts we ignored because they yielded a point T with an order too small for us to want to continue the attack) and thus, brute force for θ is fast.

Table 4. Attacks yielding the secret by altering P_3, Q_3 or R_3.

$\log_2(\mathrm{ord}(T))$	210	211	212	213	214	215	216
Number of successful attempts with altered $\phi(P_3)$	5	6	64	80	115	273	82
Number of successful attempts with altered $\phi(Q_3)$	11	14	60	52	188	371	93
Number of successful attempts with altered $\phi(R_3)$	4	19	12	10	58	75	219

Table 5. Attacks altering P_3, Q_3 or R_3 and getting $\mathrm{ord}(T) >= 210$ but not yielding the secret.

$\log_2(\mathrm{ord}(T))$	210	211	212	213	214	215	216
Number of unsucessful attempts with altered $\phi(P_3)$	0	2	5	30	176	512	2169
Number of unsucessful attempts with altered $\phi(Q_3)$	0	2	4	28	83	572	2174
Number of unsucessful attempts with altered $\phi(R_3)$	0	2	5	33	82	621	2124

Remark 6. Do note that for each (amplitude, delay) configuration, we check if either P, Q or R has been altered. Sometimes, for a given configuration, more

Table 6. Attacks yielding T but its order is too small for us to want to continue the attack.

$\log_2(\mathrm{ord}(T))$	1	206	207	208	209
Number of attempts with altered $\phi(P_3)$	155	1	4	0	3
Number of attempts with altered $\phi(Q_3)$	350	0	0	5	69
Number of attempts with altered $\phi(R_3)$	225	0	0	3	1

than one of these points is altered and matches our condition (yielding the secret, yielding a T of order greater than 210 but not the secret, yielding a T with an order strictly smaller than 210). Thus for instance the line for attempts with an altered $\phi(P_3)$ matching the chosen condition also includes attempts where two points including $\phi(P_3)$ are altered and match it, and attempts where the three points are altered and match it.

Figure 8 is a heat map representing the percentage of successful attempts i.e. those that yield Alice's secret key depending on the (delay, amplitude) configuration. After one campaign, injections of pulses with a 360 V amplitude and a delay of 440 ns seem to be the best choice to recover the secret key: there is a 0.62% chance to recover the secret key in this configuration. Do note that the maximum amplitude delivered by our pulser is 400 V, and that the number of reboots increases when approaching that limit, thus slowing the campaign. There are few configurations where there is no chance to recover the secret. This confirms than the required accuracy on the induced fault is low and compatible with practical fault injection.

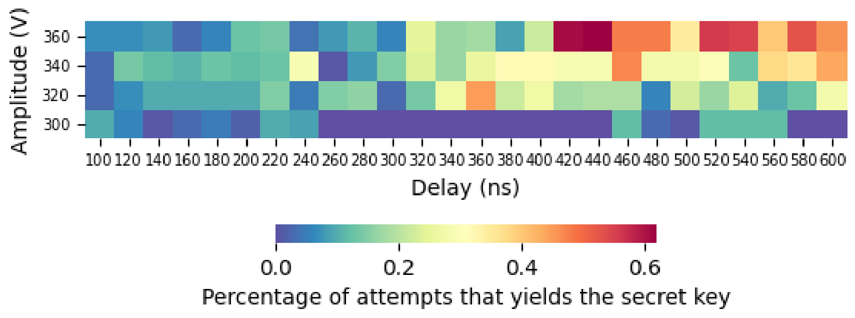

Fig. 8. Percentage of successful attempts depending on the amplitude and delay.

Remark 7. The reader might wonder why the best success rate is only 0.62% experimentally while the theoretical success rate of the attack is around 50% as seen in Remark 3. This is due to the low repeatability of electromagnetic fault injection [10]: a lot of attempts at altering the algorithm's execution does not induce faults, or at least not in a way that enables us to perform the attack (e.g. a reboot).

5 Countermeasures

As discussed in Section 3, it is difficult to make sure that people implementing the SIKE protocol will adhere to the specified API and avoid computing twice the public key. Moreover, it is not possible to avoid computing more than one public key using the same secret in a multipartite setting. Thus we propose a countermeasure that shall work in this setting too.

Let us consider the round 3 optimized implementation of SIKE. The starting curve E_0 is pushed successively through the isogenies of small degrees (update of the curve line 8 of Algorithm 1 and Algorithm 3) to be used for the computation of the kernel generator of the next isogeny (line 7 of Algorithm 1 and Algorithm 3). Let us compute the coefficient A of this last target elliptic curve. We call it A_{e_2}. But we can also compute the coefficient by using the x-coordinates of the public key. Let us call the result of this computation A_{x_1,x_2,x_3}. If at least one image of a point different of P_3, Q_3 and R_3 is computed, then the probability to recover the correct elliptic curve coefficient using the x-coordinates at the end will be very low: most of the time, we will have $A_{x_1,x_2,x_3} \neq A_{e_A}$. Algorithm 3 is a modified version of the public key generation Algorithm 1 with the added countermeasure.

We would like to know the probability of not detecting a faulted point. Suppose that there is only one faulted point. Then according to Equation (1) in Section 2.1, A is a polynomial of degree 2 in x_P for instance. Hence it has two roots in $\mathbb{F}_{p^2}$. One is the correct x_P. The probability to get the wrong one is then $\frac{1}{p^2}$, as there is only one value that is a root and that is not the correct abscissa. Looking at the size of p, it is a very low probability.

We implement this countermeasure during the public key generation. While we have chosen to attack a public key generation with 2-isogenies as explained in Remark 5 of Section 3, we also propose the variant for 3-isogenies. We implement the test of line 12 of Algorithm 3 as follows.

- **2-isogenies:** we use the computation of the coefficient A_{e_2} such that $A_{e_2} = \frac{A}{C}$ of the public key curve at the line 5 of algorithm 23 in [14]: $(A : C) = (4A_{24}^{+} - 2C_{24} : C_{24})$ in projective coordinates. Even if this coefficient is not needed in the public key, its computation is present in SIKE. We take advantage of this redundancy. Algorithm 10 is used to compute coefficient A_{x_1,x_2,x_3} using the triplet of x-coordinates of the public key. We want to check that

$$(CA_{x_1,x_2,x_3} : C) = (4A_{24}^{+} - 2C_{24} : C_{24}).$$

Thus we check that

```
Input  : A private key $\mathrm{sk}_A$.
Output: A public key $\mathrm{pk}_A$.
1  $x_S \leftarrow x_{(P_A+\mathrm{sk}_A Q_A)}$ ; // ladder3pt [14, App. A, Alg. 8]
2  $(x_1, x_2, x_3) \leftarrow (x_{P_B}, x_{Q_B}, x_{R_B})$ ; // init_basis[7]
3  for $i$ from 0 to $e_A - 1$ ; // Tree traversal loop
4   do
5      (a) Compute an $l_A$-isogeny
6
           $\phi_i : E_i \rightarrow E'$
           $(x, ...) \mapsto (f_i(x), ...)$
7      such that $\mathrm{Ker}(\phi_i) = \langle l_A^{e_A-i-1} S\rangle$.
8      b) $E_{i+1} = E'$
9      c) $x_S = f_i(x_S)$
10     d) $(x_1, x_2, x_3) \leftarrow (f_i(x_1), f_i(x_2), f_i(x_3))$
11 end
   // $A_{e_2}$ can be retrieved after the computation of $E_{e_2}$.
12 $A_{x_1,x_2,x_3} = \mathrm{get\_A}(x_1, x_2, x_3)$
13 if $A_{x_1,x_2,x_3} \neq A_{e_2}$ then
14     return 0 // fault detected, do not return the altered public key
15 else
16     return $\mathrm{pk}_A = (x_1, x_2, x_3)$.// return the public key
17 end
```
Algorithm 3: SIKE public key computation with countermeasure.

$$4A_{24}^{+} = A_{x_1,x_2,x_3}C_{24} + 2C_{24}.$$

If not, then we detect a problem during the public key generation.
This costs four additions, one multiplication and one call to `get_A`.

- **3-isogenies:** we use the computation of the coefficient A_{e_2} such that $A_{e_2} = \frac{A}{C}$ of the public key curve at the line 5 of algorithm 24 in [14]: $(A : C) = (2(A_{24}^{+} + A_{24}^{-}) : (A_{24}^{+} - A_{24}^{-}))$ in projective coordinates. Even if this coefficient is not needed in the public key, its computation is present in SIKE. Algorithm 10 is used to compute coefficient A_{x_1,x_2,x_3} using the triplet of x-coordinates of the public key. We want to check that

$$(CA_{x_1,x_2,x_3} : C) = (2(A_{24}^{+} + A_{24}^{-}) : (A_{24}^{+} - A_{24}^{-})).$$

Thus we check that

$$(A_{24}^{+} - A_{24}^{-})A_{x_1,x_2,x_3} = 2(A_{24}^{+} + A_{24}^{-}).$$

If not, then we detect a problem during the public key generation.
This costs three additions and substractions, one multiplication and one call to `get_A`.

[7] `https://github.com/microsoft/PQCrypto-SIDH/blob/97c1/src/sidh.c#L10`

A call to get_A costs seven additions and subtractions, one squaring, four multiplications and one inversion. It is possible to get rid of the inversion and obtain a faster verification by manipulating the equality we check using the formula of A in Equation (1) as computed by get_A.

The number of operations to add to implement the countermeasure is very small compared to the number of operations necessary to generate the public key, thus the overhead is low. We added the countermeasure in function EphemeralKeyGeneration_B[8] of the implementation described in Section 4 with ARMv8-A assembly optimizations and then mesured on a Cortex-A53 a 1.5% overhead during the public key generation with 3-isogenies. In our naive implementation of the countermeasure, we use the existing get_A function which includes a division to obtain the affine representation of the A coefficient. This normalization is not necessary to compare the two coefficients. The overhead could thus be further reduced by avoiding the division. The verification can also be done during and after the tree traversal step of Algorithm 3. But considering the probability to detect a fault at the end of the public key computation, it does not seem necessary.

6 Conclusion

We have shown that Ti's 2017 fault injection attack on the key generation step of SIKE is exploitable in practice though electromagnetic injection on a SoC. While it is complex to generate faults on a SoC, Ti's attack does not require a high precision when performing it, which simplifies the experimental verification in a laboratory. In a 4.5 days campaign, 0.17% of the attack configurations yielded the secret key for at least one of the altered public key points, which corresponds to around one configuration that enables us to recover the secret key every 3 minutes and 18 seconds. This attack requires both the real public key of Alice and an altered version. While the attack scenario is unlikely to apply to implementations of SIKE that respect the KEM API, it occurs in a multipartite setting. We thus propose a countermeasure which consists in computing the public key curve coefficient by using two different methods. This countermeasure has both a small overhead and a high probability to detect a fault. It remains to be seen if the attack is still feasible when the public keys are compressed.

References

1. Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Kelsey, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., et al.: Status report on the second round of the NIST post-quantum cryptography standardization process. US Department of Commerce, NIST (2020)
2. Azarderakhsh, R., Jalali, A., Jao, D., Soukharev, V.: Practical supersingular isogeny group key agreement. IACR Cryptol. ePrint Arch. **2019**, 330 (2019)

[8] https://github.com/microsoft/PQCrypto-SIDH/blob/97c1/src/sidh.c/#L123

3. Azarderakhsh, R., Jao, D., Kalach, K., Koziel, B., Leonardi, C.: Key compression for isogeny-based cryptosystems. In: Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography. pp. 1–10 (2016)
4. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: International workshop on cryptographic hardware and embedded systems. pp. 292–302. Springer (1999)
5. Costello, C., Jao, D., Longa, P., Naehrig, M., Renes, J., Urbanik, D.: Efficient compression of SIDH public keys. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 679–706. Springer (2017)
6. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for Supersingular Isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016: 36th Annual International Cryptology Conference. pp. 572–601. Springer Berlin Heidelberg (2016). https://doi.org/doi:10.1007/978-3-662-53018-4_21
7. Costello, C., Smith, B.: Montgomery curves and their arithmetic. Journal of Cryptographic Engineering **8**(3), 227–240 (2018)
8. De Feo, L.: Mathematics of isogeny based cryptography. CoRR **abs/1711.04062** (2017), `http://arxiv.org/abs/1711.04062`
9. Dehbaoui, A., Dutertre, J.M., Robisson, B., Tria, A.: Electromagnetic transient faults injection on a hardware and a software implementations of AES. In: 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 7–15. IEEE (2012)
10. Gaine, C., Aboulkassimi, D., Pontié, S., Nikolovski, J.P., Dutertre, J.M.: Electromagnetic fault injection as a new forensic approach for SoCs. In: 2020 IEEE International Workshop on Information Forensics and Security (WIFS). pp. 1–6. IEEE (2020)
11. Galbraith, S.D., Petit, C., Shani, B., Ti, Y.B.: On the security of supersingular isogeny cryptosystems. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 63–91. Springer (2016)
12. Gélin, A., Wesolowski, B.: Loop-abort faults on supersingular isogeny cryptosystems. In: International Workshop on Post-Quantum Cryptography. pp. 93–106. Springer (2017)
13. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Theory of Cryptography Conference. pp. 341–371. Springer (2017)
14. Jao, D., Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hess, B., Hutchinson, A., Jalali, A., Karabina, K., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Pereira, G., Renes, J., Soukharev, V., Urbanik, D.: SIKE: supersingular isogeny key encapsulation (2020), `https://sike.org/files/SIDH-spec.pdf`
15. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: International Workshop on Post-Quantum Cryptography. pp. 19–34. Springer (2011)
16. Koppermann, P., Pop, E., Heyszl, J., Sigl, G.: 18 seconds to key exchange: Limitations of supersingular isogeny Diffie-Hellman on embedded devices. IACR Cryptology ePrint Archive **2018**, 932 (2018)
17. Koziel, B., Azarderakhsh, R., Jao, D.: Side-channel attacks on quantum-resistant supersingular isogeny Diffie-Hellman. In: International Conference on Selected Areas in Cryptography. pp. 64–81. Springer (2017)
18. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Mathematics of computation **48**(177), 243–264 (1987)

19. Naehrig, M., Renes, J.: Dual isogenies and their application to public-key compression for isogeny-based cryptography. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 243–272. Springer (2019)
20. NIST: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (Dec 2016), https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf
21. Pereira, G.C., Doliskani, J., Jao, D.: x-only point addition formula and faster torsion basis generation in compressed SIKE. IACR Cryptol. ePrint Arch. **2020**, 431 (2020)
22. Proy, J., Heydemann, K., Berzati, A., Cohen, A.: Compiler-assisted loop hardening against fault attacks. ACM Transactions on Architecture and Code Optimization (TACO) **14**(4), 1–25 (2017)
23. Proy, J., Heydemann, K., Majeric, F., Cohen, A., Berzati, A.: Studying EM pulse effects on superscalar microarchitectures at isa level. arXiv preprint arXiv:1903.02623 (2019)
24. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science. pp. 124–134. IEEE (1994)
25. Silverman, J.H.: The arithmetic of elliptic curves, vol. 106. Springer Science & Business Media (2009)
26. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 8.1) (2017), https://www.sagemath.org
27. Ti, Y.B.: Fault attack on supersingular isogeny cryptosystems. In: International Workshop on Post-Quantum Cryptography. pp. 107–122. Springer (2017)
28. Vélu, J.: Isogénies entre courbes elliptiques. CR Acad. Sci. Paris, Séries A **273**, 305–347 (1971)
29. Zanon, G.H., Simplicio, M.A., Pereira, G.C., Doliskani, J., Barreto, P.S.: Faster isogeny-based compressed key agreement. In: International Conference on Post-Quantum Cryptography. pp. 248–268. Springer (2018)
30. Zhang, F., Yang, B., Dong, X., Guilley, S., Liu, Z., He, W., Zhang, F., Ren, K.: Side-channel analysis and countermeasure design on ARM-based quantum-resistant SIKE. IEEE Transactions on Computers **69**(11), 1681–1693 (2020)