Loop-Abort Faults on Supersingular Isogeny Cryptosystems

Alexandre Gélin, Benjamin Wesolowski

Loop-abort faults on supersingular isogeny cryptosystems

Alexandre Gélin[1] and Benjamin Wesolowski[2]

[1] Sorbonne Universités, UPMC Paris 6, UMR 7606, LIP6, Paris, France
alexandre.gelin@lip6.fr
[2] École Polytechnique Fédérale de Lausanne, EPFL IC LACAL, Switzerland
benjamin.wesolowski@epfl.ch

Abstract. Cryptographic schemes based on supersingular isogenies have become an active area of research in the field of post-quantum cryptography. We investigate the resistance of these cryptosystems to fault injection attacks. It appears that the iterative structure of the secret isogeny computation renders these schemes vulnerable to loop-abort attacks. Loop-abort faults allow to perform a full key recovery, bypassing all the previously introduced validation methods. Therefore implementing additional countermeasures seems unavoidable for applications where physical attacks are relevant.

Keywords: Supersingular isogeny cryptosystem, fault injection, real-world attacks, post-quantum cryptography.

1 Introduction

Public-key cryptography, the foundation of modern communication security, is confronted to the prospect of a technology capable of breaking today's most widely deployed primitives: quantum computers of sufficient scale to run Shor's algorithm [19]. This risk has given rise to the field of post-quantum cryptography (PQC), which aims at developing new primitives that would resist cryptanalysis by both classical and quantum computers.

Cryptographic schemes based on supersingular elliptic curve isogenies were introduced in [5] which proposed the hardness of path-finding in supersingular isogeny graphs and gave an application to cryptographic hash functions. It has since been used as an assumption for other cryptographic systems such as key-exchange, encryption, and signatures [12]. Several other primitives have subsequently been built based on supersingular isogeny problems, such as zero-knowledge proofs of identity and signatures [7,11,13,21]. Efficient implementations of these primitives have rapidly followed: in software [6,16], in hardware [15] and on embedded systems [1].

Even though the basic version of the key exchange protocol uses ephemeral secret values (as with classical Diffie-Hellman), some of these other schemes require static secret keys for at least one party. Such static secrets constitute

primary material for active attacks, and such an attack was indeed described in [10] that allows to find all n bits of the secret key with about n interactions with the victim. This attack can be prevented by the Kirkwood *et al.* [14] validation method — which is essentially a Fujisaki–Okamoto transform [9] applied in the context of supersingular isogenies.

The results of [1], together with the fact that primitives based on supersingular isogenies enjoy significantly smaller keys than the other main candidates for post-quantum cryptography, suggest that they might be well-suited for use on embedded devices. This opens new avenues of potential side-channel attacks.

In this paper, we present the first side-channel attack against supersingular isogeny-based primitives, exploiting a fault-injection technique known as loop-abort fault injection, previously introduced for pairing based cryptography [17]. The iterative structure of isogeny computations render them susceptible to loop-abort fault attacks, allowing an attacker to recover all n bits of the key, within $O(n)$ interactions with the token and a negligible amount of computation. This attack is not prevented by any of the validation methods previously discussed for isogeny-based cryptosystems. Loop-abort fault injections were proven to be feasible in practice [3], and should therefore be taken into serious consideration when implementing schemes based on supersingular isogenies in a context susceptible to physical attacks.

After a brief overview of supersingular isogeny-based cryptography in Section 2, we quickly discuss fault injection attacks in Section 3, with a particular focus on loop-abort faults. The attack is then described and analyzed in Section 4, while potential countermeasures are discussed in Section 5.

2 Cryptosystems from supersingular isogenies

This section recalls the necessary background and the cryptosystems based on supersingular elliptic curve isogenies introduced in [12] and [7].

2.1 Elliptic curves and isogenies

For any prime power $q = p^r$, let $\mathbf{F}_q$ denote the finite field with q elements. Let E be an elliptic curve defined over $\mathbf{F}_q$. The set $E(\overline{\mathbf{F}}_q)$ of rational points over the algebraic closure forms a group, in which the subset $E(\mathbf{F}_q)$ of $\mathbf{F}_q$-rational points forms a finite subgroup. Let E' be another elliptic curve defined over $\mathbf{F}_q$. An isogeny $E \to E'$ is a non-constant morphism sending the identity of E to the identity of E' (we then say that E' is isogenous to E). In particular, an isogeny is a surjective group homomorphism from $E(\overline{\mathbf{F}}_q)$ to $E'(\overline{\mathbf{F}}_q)$, of finite kernel. All isogenies that we consider in this paper are separable [20, Definition p.21], so by *isogeny* we always mean *separable isogeny*. Then, the *degree* of an isogeny is simply defined as the number of points in its kernel. For any finite subgroup $G \subset E(\overline{\mathbf{F}}_q)$, there is a quotient elliptic curve E/G and the canonical projection $\pi_G : E \to E/G$ is an isogeny of degree $|G|$. In fact, any isogeny arises in this form, up to an isomorphism of the target.

2.2 Elliptic curves for supersingular isogeny schemes

Fix a reference elliptic curve E_0. In cryptographic schemes based on supersingular isogenies, the public key of Alice is, essentially, an isogenous curve E_A, and its secret is the kernel G_A of an isogeny $\varphi_A : E_0 \to E_A$. Two things appear immediately: first, Alice needs an efficient way to represent the secret kernel. Second, given E_0 and E_A, it should be difficult to find the kernel G_A. The secret isogenies are simply chosen cyclic, so that the secret is a point $P \in E_0(\mathbf{F}_q)$ generating the kernel. This means in particular that $E_0(\mathbf{F}_q)$ should contain a lot of cyclic subgroups. It follows that the traditional constraint in elliptic curve cryptography that the order of $E_0(\mathbf{F}_q)$ should have a large prime factor does not apply here: on the contrary, $|E_0(\mathbf{F}_q)|$ should be very smooth.

Let m be any positive integer; then $E[m] \subset E(\overline{\mathbf{F}}_q)$ is the subgroup of m-torsion points in $E(\overline{\mathbf{F}}_q)$. Whenever m is coprime to p, we have that $E[m]$ is isomorphic to the group $(\mathbf{Z}/m\mathbf{Z})^2$. Now, fix a prime number $\ell \neq p$, and a positive integer k. Then $E[\ell^k] \cong (\mathbf{Z}/\ell^k\mathbf{Z})^2$ contains $\ell^{k-1}(\ell+1)$ distinct cyclic subgroups of order ℓ^k. If all the points of $E[\ell^k]$ are defined over $\mathbf{F}_q$, then the set of cyclic subgroups of order ℓ^k generated by a point defined over $\mathbf{F}_q$ is sufficiently large to constitute a space of secret keys. Typically, to build the cryptographic schemes, one needs two distinct primes ℓ_A and ℓ_B (one for the keys of Alice and one for the keys of Bob). Then, the reference elliptic curve E_0 should be chosen so that

$$E_0(\mathbf{F}_q) = E_0[\ell_A^n] \oplus E_0[\ell_B^m].$$

This might be difficult to obtain, so one can simply require $E_0(\mathbf{F}_q) = E_0[\ell_A^n] \oplus E_0[\ell_B^m] \oplus F$, where the subgroup F contains a few other $\mathbf{F}_q$-rational points whose orders are coprime to ℓ_A and ℓ_B.

The schemes also require the reference curve E_0 to be supersingular — see [20, Theorem 3.1] for a few of the numerous equivalent ways to define supersingularity. The first interesting consequence is that all supersingular curves defined over $\overline{\mathbf{F}}_q$ are actually defined over $\mathbf{F}_{p^2}$. The curve E_0 is constructed as follows. The primes ℓ_A and ℓ_B are fixed (typically 2 and 3), together with some exponents n and m whose size depend on the security level. Now, one can find a cofactor f coprime to ℓ_A and ℓ_B such that $p = \ell_A^n \ell_B^m f \pm 1$ is a prime number. Bröker [4] has shown that it is easy to find an elliptic curve E_0 over $\mathbf{F}_{p^2}$ such that $|E_0(\mathbf{F}_{p^2})| = (p \mp 1)^2 = (\ell_A^n \ell_B^m f)^2$. From [2, Theorem IX.20], the group $E_0(\mathbf{F}_{p^2})$ is isomorphic to $(\mathbf{Z}/\ell_A^n \ell_B^m f\mathbf{Z})^2$, and therefore we have the desired decomposition $E_0(\mathbf{F}_{p^2}) = E_0[\ell_A^n] \oplus E_0[\ell_B^m] \oplus F$, with $F = E_0[f]$.

This choice of ℓ_A, ℓ_B, n, m, p and E_0 is fixed for the rest of the paper. We are working on the class of all elliptic curves E that are $\mathbf{F}_{p^2}$-isogenous to E_0. They are all supersingular, and by Tate's isogeny theorem [22], they all have the same number of $\mathbf{F}_{p^2}$-rational points. It follows that any such E enjoys the same decomposition

$$E(\mathbf{F}_{p^2}) = E[\ell_A^n] \oplus E[\ell_B^m] \oplus F.$$

2.3 Computing smooth degree isogenies

Let $(\ell, k) \in \{(\ell_A, n), (\ell_B, m)\}$. For any elliptic curve E isogenous to E_0, the subgroup $E[\ell^k] \subset E(\mathbf{F}_{p^2})$ is a free $\mathbf{Z}/\ell^k\mathbf{Z}$-module of rank two, with a basis $\{P, Q\}$. Any point $R = c_1P + c_2Q$ has order ℓ^k if and only if either c_1 or c_2 is not divisible by ℓ. Any such point R of order ℓ^k induces an isogeny $\varphi_R : E \to E/\langle R\rangle$ of degree ℓ^k. As described in [7, Section 4.2.2], the only efficient way to compute the isogeny φ_R is as a sequence of k isogenies of degree ℓ, which can themselves be computed using Vélu's formulas [23]. We now recall that method. Set $E^0 = E$, $R^0 = R$, and for $0 \leq i < k$, recursively define

$$\begin{aligned} &E^{i+1} = E^i/\langle \ell^{k-i-1} R^i \rangle, \\ &\varphi_{i+1} : E^i \to E^{i+1} \text{ the canonical isogeny,} \\ &R^{i+1} = \varphi_{i+1}(R^i). \end{aligned}$$

Then, $E/\langle R\rangle = E^k$, each isogeny φ_i is of degree ℓ, and $\varphi_R = \varphi_k \circ \cdots \circ \varphi_1$. From this observation, [7, Section 4.2.2] describes a family of strategies to compute φ_R. These strategies allow to optimize the number of point multiplications and isogeny computations, but all boil down to computing the sequence of isogenies $(\varphi_i)_{i=1}^k$ in order.

2.4 Jao–De Feo protocols

We present here two examples of cryptographic protocols based on supersingular isogenies: a key exchange protocol, and a public key encryption protocol. Our attack could easily be adapted to other variants of isogeny-based schemes as we target a component they all have in common: the computation by Alice of a secret isogeny.

Key exchange. The following key exchange protocol is the first supersingular isogeny-based protocol, introduced in [12, Section 3.1].

Setup Choose a prime $p = \ell_A^n \ell_B^m f \pm 1$ and an elliptic curve E_0 as in Section 2.2. As $E_0[\ell_A^n]$ is a free $\mathbf{Z}/\ell_A^n\mathbf{Z}$-module of rank two, let $\{P_A, Q_A\}$ be a basis[3]. Similarly, let $\{P_B, Q_B\}$ be a basis of $E_0[\ell_B^m]$.

Key exchange Alice chooses two random elements $a_1, a_2 \in \mathbf{Z}/\ell_A^n\mathbf{Z}$, not both nilpotent[4], and Bob does the same, with $b_1, b_2 \in \mathbf{Z}/\ell_B^m\mathbf{Z}$. Let

$$\begin{aligned} R_A &= a_1P_A + a_2Q_A, \\ R_B &= b_1P_B + b_2Q_B. \end{aligned}$$

[3] In [6], the pair (P_A, Q_A) does not form a basis. The protocol still works, but some caution is required (see Appendix A).

[4] Note that an element $a \in \mathbf{Z}/\ell_A^n\mathbf{Z}$ is nilpotent if and only if it is the class of a multiple of ℓ_A.

Write $E_A = E_0/\langle R_A \rangle$ and $E_B = E_0/\langle R_B \rangle$, and let $\varphi_A : E_0 \to E_A$ and $\varphi_B : E_0 \to E_B$ denote the canonical isogenies. Alice computes and publishes $(E_A, \varphi_A(P_B), \varphi_A(Q_B))$, and Bob does the same for $(E_B, \varphi_B(P_A), \varphi_B(Q_A))$. Alice can compute the j-invariant of

$$E_{AB} = E_B/\langle a_1\varphi_B(P_A) + a_2\varphi_B(Q_A)\rangle,$$

while Bob can compute the j-invariant of

$$E_{BA} = E_A/\langle b_1\varphi_A(P_B) + b_2\varphi_A(Q_B)\rangle.$$

Fortunately $E_{AB} \cong E_0/\langle R_A, R_B\rangle \cong E_{BA}$, so $j(E_{AB}) = j(E_{BA})$ is the secret shared by Alice and Bob, from which they can derive a secret key.

Public-key encryption. The public-key encryption scheme described in [12, Section 3.2] is very similar to the key exchange. Here, Bob sends an encrypted message to Alice.

Setup As for the key exchange, choose a prime $p = \ell_A^n \ell_B^m f \pm 1$ and an elliptic curve E_0, together with bases $\{P_A, Q_A\}$ for $E_0[\ell_A^n]$ and $\{P_B, Q_B\}$ for $E_0[\ell_B^m]$. Let $\mathscr{H} = \{H_k \mid k \in K\}$ be a family of hash functions indexed by a finite set K, where each H_k is a function from $\mathbf{F}_{p^2}$ to the message space $\{0,1\}^w$.

Key generation Choose two random elements $a_1, a_2 \in \mathbf{Z}/\ell_A^n\mathbf{Z}$, not both nilpotent. Let

$$E_A = E_0/\langle a_1P_A + a_2Q_A\rangle,$$

and $\varphi_A : E_0 \to E_A$ the canonical isogeny. Choose a random element $k \in K$. The public key is $(E_A, \varphi_A(P_B), \varphi_A(Q_B), k)$, and the private key is (a_1, a_2, k).

Encryption Given a public key $(E_A, \varphi_A(P_B), \varphi_A(Q_B), k)$ and a message $m \in \{0,1\}^w$, choose two random elements $b_1, b_2 \in \mathbf{Z}/\ell_B^m\mathbf{Z}$, not both nilpotent. Fix

$$E_B = E_0/\langle b_1P_B + b_2Q_B\rangle,$$

and $\varphi_B : E_0 \to E_B$ the canonical isogeny, and let $E_{BA} = E_A/\langle b_1\varphi_A(P_B) + b_2\varphi_A(Q_B)\rangle$. Compute the hash value $h = H_k(j(E_{BA}))$ and $c = h \oplus m$. The ciphertext is $(E_B, \varphi_B(P_A), \varphi_B(Q_A), c)$.

Decryption Given the ciphertext $(E_B, \varphi_B(P_A), \varphi_B(Q_A), c)$, compute

$$E_{AB} = E_B/\langle a_1\varphi_B(P_A) + a_2\varphi_B(Q_A)\rangle.$$

Let $m = H_k(j(E_{AB})) \oplus c$. Since $j(E_{AB}) = j(E_{BA})$, m is the plaintext.

Remark 1. Observe that Alice's private key (a_1, a_2) is equivalent to (ra_1, ra_2) for any r coprime to ℓ_A, since $a_1P_A + a_2Q_A$ and $ra_1P_A + ra_2Q_A$ generate the same subgroup of $E_0[\ell_A^n]$. Recall that a_1 and a_2 are not both nilpotent. Seen as integers, this means that one of them is not divisible by ℓ_A, so is invertible in $\mathbf{Z}/\ell_A^n\mathbf{Z}$. Therefore (a_1, a_2) is equivalent to either $(1, a)$ (if a_1 is coprime to ℓ_A) or to $(a, 1)$ (if a_2 is coprime to ℓ_A).

2.5 Validation methods against active attacks

Various validation methods have been introduced in order to prevent active attacks against isogeny-based protocols. Classical methods allowing to check if an elliptic curve is supersingular, or if two given points P and Q lie on the curve, have the correct order and are independent are discussed in [6]. In [10, Section 2.4], some validation steps are simplified, and a new one is introduced that allows Alice to gain some assurance that the two points of Bob's public-key are the images of the two base points P_A, Q_A through an isogeny of degree ℓ_B^m. The active attack of [10] allows a dishonest party to send malicious points to Alice that nevertheless pass all these validations.

The only effective countermeasure that has been proposed to prevent this attack against the key exchange protocol is to apply the Fujisaki–Okamoto transform [9], as explained in [18], which in the context of the isogeny key exchange has been discussed by Kirkwood *et al.* [14]. This validation method essentially forces Bob to send to Alice honestly generated parameters, and to prove that he has been able to compute the shared secret. More precisely, the following describes the key exchange protocol resulting from applying the Fujisaki–Okamoto transform. Here, Alice has a static secret key (a_1, a_2), and Bob — the potential adversary — generates an ephemeral key.

Bob's key generation Bob chooses a random seed r, from which he derives his secret key $(b_1, b_2) = f(r)$ via a pseudo-random function f. He proceeds to compute his message $(E_B, \varphi_B(P_A), \varphi_B(Q_A))$ as in the regular key exchange. He computes the shared secret E_{AB} using Alice's public key $(E_A, \varphi_A(P_B), \varphi_A(Q_B))$, and derives a session key $\mathtt{sk}$ and a validation key $\mathtt{vk}$ via a key derivation function (KDF), as

$$(\mathtt{sk}, \mathtt{vk}) = \mathrm{KDF}(j(E_{AB})).$$

Key exchange Bob sends the tuple $(E_B, \varphi_B(P_A), \varphi_B(Q_A))$ and the ciphertext $c = \mathrm{Enc}_{\mathtt{vk}}(r \oplus \mathtt{sk})$ to Alice.

Validation Alice first derives E'_{AB}, then $\mathtt{sk}'$ and $\mathtt{vk}'$, to recover the value $r' = \mathrm{Dec}_{\mathtt{vk}'}(c) \oplus \mathtt{sk}'$. From this hypothetical seed r', she is able to recompute Bob's operations. If the result she finds coincides with the tuple $(E_B, \varphi_B(P_A), \varphi_B(Q_A))$ she has received, she terminates the protocol by accepting $\mathtt{sk}' = \mathtt{sk}$ as the shared secret. Otherwise, she returns an error.

In the fault injection attack we describe, Bob forces Alice to compute a wrong isogeny. By predicting the output of this wrong isogeny, Bob can make sure the fault is not detected by the validation, and Alice leaks some faulted information.

3 Fault injection attacks

Fault injection attacks are a standard kind of attacks assuming physical access to a device (Alice) using a static private key (in the present context, a pair (a_1, a_2)). They rely on the ability of the attacking party to tamper with Alice's execution

of the protocol, causing her to commit errors in her computations. For the rest of the paper, we fix $\ell_A = 2$ and $\ell_B = 3$ for simplicity, but the results can easily be applied to any ℓ_A and ℓ_B.

3.1 Tampering with bits and bytes

The secret key is a pair of integers $a_1, a_2 \in \{1, ..., 2^n - 1\}$, such that the kernel of the secret isogeny is $R_A = a_1 P_A + a_2 Q_A$. If it is possible to force a particular bit of (a_1, a_2) to, say, 0, then the value of that bit can be recovered by checking if the faulty outcome is valid. A folklore way to make this kind of attack ineffective is to regularly randomize the representation of the secret key. In the present context, this could be done by choosing a random odd integer r each time the private key is used, and to replace (a_1, a_2) by the equivalent pair (ra_1, ra_2).

There exist mostly three ways to alter the value of a bit during execution: forcing it to 0 or 1, flipping it, or randomizing it. The simple attack above assumes that a bit can be forced to some value, but the weakest of these three assumptions is obviously the bit randomization. Under this assumption, a particular bit can be made to take a random value among 0 or 1, unknown from the attacker. This cannot be exploited in such a straightforward way as the bit-forcing assumption, but suffices for the more elaborate attack we present.

Targeting a particular bit might turn out to be difficult. Rather than a precise bit error, one could aim for a byte error (or a word error): a fault is injected in a byte, and the new value of the byte cannot be predicted by the attacker.

3.2 Implementing loop-aborts

A powerful way to exploit fault injections is to implement loop aborts. The weakest of the above assumptions are sufficient to force a loop to come to an end. Loop-abort faults have been introduced by Page and Vercauteren in the context of pairing based cryptography [17], and recently used by Espitau *et al.* [8] to attack signature schemes based on lattices. It is explained in [8, Section 5] that a loop-abort can simply be implemented by injecting a random fault on the loop counter — or alternatively, by skipping the jump instruction. Loop-abort faults appear to be feasible in practice, and have already been successfully performed in attacks against pairing based cryptography by Blömer *et al.* [3].

As any fault attack, it requires some knowledge on the structure of the implementation being targeted, to know precisely at which place and time the fault should be injected. Such structure can be recovered by combining some knowledge of the algorithms involved (and maybe of standard implementations), reverse-engineering techniques and side-channel analysis. Then, a few circumstances make loop-abort attacks easier. First, most of the possible values of the loop-counter would cause the loop to abort, so it is not necessary to target a specific bit: some imprecise randomizations should do the trick. Second, to exit the loop after k iterations, it is sufficient to tamper with the counter at any time during the execution of the k-th iteration. With that strategy, if an iteration

takes time to execute, the timing of the fault injection need not be excessively precise.

4 The loop-abort fault attack

This attack takes advantage of the iterative structure of the isogeny computation, as described in Section 2.3. The critical point is Alice's computation of an isogeny $E \to E/\langle a_1 R + a_2 S\rangle$ based on her secret. One can inject a fault to cause Alice to compute this isogeny only partially, leaking information about the secret key (a_1, a_2).

4.1 Attack framework

Alice takes as input the public parameters of a party who wants to initiate a secure communication, and performs her side of the protocol using a static private key (a_1, a_2). We assume that the countermeasures discussed in [10] are implemented, and in particular the Kirkwood *et al.* validation method [14], which essentially prevents attacks based on tricking Alice into computing on maliciously cooked data. Then, Alice is modeled in the form of an oracle $\mathcal{O}(E, R, S, E', b_1, b_2)$ which returns 1 if

$$\begin{aligned} j(E') &= j(E/\langle a_1 R + a_2 S\rangle), \\ j(E) &= j(E_0/\langle b_1 P_B + b_2 Q_B\rangle) \text{ and} \\ (R, S) &= (\varphi_B(P_A), \varphi_B(Q_A)) \end{aligned}$$

and 0 otherwise. The first condition corresponds, as in the second oracle of [10, Section 3], to Alice taking Bob's protocol message, performing her side of the protocol, and returning an error if the shared key she finds does not coincide with Bob's. The second and third conditions account for the Kirkwood *et al.* [14] validation method: Bob can only use honestly generated parameters.

We further assume, and it is the foundation of our attack, that the attacker can make Alice abort the main loop after the k-th iteration during her computation of the isogeny $E \to E/\langle a_1 R + a_2 S\rangle$, using techniques discussed in Section 3.2. As described in Section 2.3, recall that after k iterations, Alice has computed the intermediate elliptic curve

$$E^k \cong E/\langle 2^{n-k}(a_1 R + a_2 S)\rangle$$

The fault injection is modeled by the oracle $\mathcal{O}^k(E, R, S, E', b_1, b_2)$, that returns 1 if $j(E') = j(E^k)$, $j(E) = j(E_0/\langle b_1 P_B + b_2 Q_B\rangle)$ and $(R, S) = (\varphi_B(P_A), \varphi_B(Q_A))$, and 0 otherwise.

4.2 The attack

Alice's key (a_1, a_2) is either equivalent to $(1, a)$ (if a_1 is odd) or $(a, 1)$ (if a_2 is odd), and the attacker recover one by one the bits of a from the least significant to

the most significant[5]. The attacker, playing the role of Bob, chooses a (random) secret key $(b_1, b_2) \in \{1, \cdots, 3^m\}^2$ that he is going to use for the attack. He computes $E_B = E_0/\langle b_1 P_B + b_2 Q_B \rangle$ and lets $\varphi_B : E_0 \to E_B$ be the canonical isogeny. Also write $P'_A = \varphi_B(P_A)$ and $Q'_A = \varphi_B(Q_A)$. First, the attacker needs to decide whether the key can be represented as $(1, a)$ or $(a, 1)$. We have

$$\begin{aligned} E_B^1 &\cong E_B/\langle 2^{n-1}(a_1 P'_A + a_2 Q'_A)\rangle \\ &= E_B/\langle (2^{n-1}a_1 \bmod 2^n)P'_A + (2^{n-1}a_2 \bmod 2^n)Q'_A)\rangle. \end{aligned}$$

Therefore a_1 is even if and only if the first isogeny $E_B \to E_B^1$ has kernel $\langle 2^{n-1}Q'_A\rangle \subset E[2]$. This can be decided by determining the kernel $\kappa \subset E_B[2]$ of the first isogeny computed by Alice. The attacker guesses one of the three possible proper subgroups $\kappa \subset E_B[2]$, and queries the oracle $\mathcal{O}^1$ on the input $(E_B, \varphi_B(P'_A), \varphi_B(Q'_A), E_B/\kappa, b_1, b_2)$. If the output is 1, then the guess of κ was correct. If the output is 0, either the guess is incorrect or the fault injection failed. This allows simultaneously to determine if the key can be represented as $(1, a)$ or $(a, 1)$, and to determine the least significant bit of a. More precisely,

if $\kappa = \langle 2^{n-1}Q'_A\rangle$ then a_1 is even and a_2 is odd, so the key can be represented as $(a, 1)$, and the least significant bit of a is 0.
if $\kappa = \langle 2^{n-1}P'_A\rangle$ then a_2 is even and a_1 is odd, so the key can be represented as $(1, a)$, and the least significant bit of a is 0.
if $\kappa = \langle 2^{n-1}(P'_A + Q'_A)\rangle$ then both a_1 and a_2 are odd, so the key can be represented as $(1, a)$, and the least significant bit of a is 1.

In the following, we assume without loss of generality that the key is represented as $(1, a)$. Now, it remains to recover all the other bits of a, from the least significant to the most significant. In order to match the structure of the attack, the bits of a are indexed from the least significant to the most significant. Observe that

$$E_B^k \cong E_B/\langle 2^{n-k}(P'_A + aQ'_A)\rangle = E_B/\langle 2^{n-k}P'_A + (2^{n-k}a \bmod 2^n)Q'_A)\rangle$$

only depends on the k least significant bits of a. If we know the first $k-1$ bits of the key, we can guess the k-th one and compute the corresponding degree-ℓ^k isogeny. Then, it suffices to compare the result with the faulty outcome to determine if the guess is correct. More precisely, at the k-th step, once the $k-1$ first bits of the key a have already been recovered, we perform the following steps:

1. Choose a guess $b \in \{0, 1\}$;
2. Compute
$$\tilde{E}_B^k = E_B/\langle 2^{n-k}P'_A + \tilde{a}2^{n-k}Q'_A)\rangle$$
where $\tilde{a} = b \cdot 2^{k-1} + (a \bmod 2^{k-1})$ is the concatenation of b with the first $k-1$ bits of a;

[5] Note the contrast with the simple attack of Section 3.1, in which the way Alice internally represents her secret key is crucial. In this more evolved attack, Alice's representation is irrelevant.

3. Query the oracle $\mathcal{O}^k$ on $\left(E_B, P'_A, Q'_A, \tilde{E}_B^k, b_1, b_2\right)$;
4. If the oracle returns 1 (*i.e.* it does not detect an error), then the k-th bit is b; otherwise, either the bit is $1-b$, or the fault injection failed.

Assuming that the fault injection is a success (*i.e.*, the oracle $\mathcal{O}^k$ is called successfully), it is clear that the oracle returns 1 only if $\tilde{E}_B^k$ is indeed the k-th elliptic curve in the sequence of isogenies that Alice should have computed, meaning that b is a correct guess of the k-th bit of a.

4.3 Analysis of the attack

Let μ denote the probability of successfully aborting the computation[6]. The attack above requires about $2n/\mu \approx \log_2 p/\mu$ fault injections, and as many interactions with Alice[7]. Considering that Alice is modeled as an oracle that outputs only one bit, $O(n)$ is optimal in the sense that we cannot hope to recover more than one bit of the key per interaction. The additional factor μ accounts for the potential difficulty of injecting faults. In Section 4.4, we describe a way to reduce the number of faults, assuming a stronger oracle which leaks more information.

To abort after k iterations, the fault must be injected during the execution of the k-th iteration. All known implementations of the isogeny computation share the same iterative structure, however the duration of an iteration is not necessarily a constant, and depends on the choice of a *strategy* — in the sense of [7, Section 4.2.2]. For instance, in the SIDH Library [16], the first iteration is the longest, taking about $4.0 \cdot 10^6$ CPU cycles (measured on an Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz), while the other iterations take between $7.9 \cdot 10^4$ and $1.8 \cdot 10^6$ CPU cycles. Finding the right moment to inject the fault can be easy if Alice is using a public implementation [6], and otherwise requires to reverse-engineer the strategy by side-channel analysis.

This attack is the first one against isogeny-based schemes that bypasses the costly Kirkwood *et al.* validation method [14]. In essence, this validation method checks these two statements: first, Bob's parameters are honestly generated (and in our attack, they indeed are), and Bob knows the shared key computed by Alice (this is precisely the guess $\tilde{E}_B^k$, which has a probability $1/2$ of being correct).

4.4 Alternative with less faults but stronger oracle

Instead of the weak oracle we have considered so far, which outputs a single bit at each call, one could also consider a more powerful oracle, closer to the first oracle of [10, Section 3] (but still weaker). Let $\mathcal{H}$ be a function with very rare collisions (it could be the identity, or a hash function), and consider the oracle

[6] For simplicity, we assume that this probability is independent of the number k of iterations after which we want to abort.

[7] More precisely, if there exists a way to determine that a fault was successful (for instance, if $\mu = 1$), we can get rid of the factor 2, because a failure brings the information that the guess is wrong, so the bit is $1-b$.

$\mathcal{O}(E, R, S, b_1, b_2)$ which outputs $\mathcal{H}(j(E/\langle a_1 R + a_2 S\rangle))$ if $j(E) = j(E_0/\langle b_1 P_B + b_2 Q_B\rangle)$ and $(R, S) = (\varphi_B(P_A), \varphi_B(Q_A))$, and 0 otherwise. We can then hope to reduce the number of faults necessary for the attack, at the cost of additional computations.

As previously, we also define the oracle $\mathcal{O}^k(E, R, S, b_1, b_2)$ which outputs

$$\mathcal{H}(j(E/\langle 2^{n-k}(a_1 R + a_2 S)\rangle))$$

if $j(E) = j(E_0/\langle b_1 P_B + b_2 Q_B\rangle)$ and $(R, S) = (\varphi_B(P_A), \varphi_B(Q_A))$, and 0 otherwise. The idea is to consider batches of $s > 1$ bits instead of recovering one bit at a time, thereby reducing the required number of faults from n to $\lceil \frac{n}{s} \rceil$.

As in the attack of Section 4.2, the attacker generates parameters b_1, b_2, E_B and φ_B. Assuming the key is represented as $(1, a)$, the s least significant bits of a can be recovered as follows: call the oracle $\mathcal{O}^s(E_B, \varphi_B(P_A), \varphi_B(Q_A), b_1, b_2)$, and call the output h. Find by exhaustive search the value $x \in \{0, 1, \ldots, 2^s - 1\}$ such that $h = \mathcal{H}(E_B/\langle 2^{n-s}(P'_A + xQ'_A)\rangle)$. Unless a collision occurred in $\mathcal{H}$, this x is exactly $a \bmod 2^s$, corresponding to the s least significant bits of a. Generalizing this to the other bits is straightforward. The time complexity to find a batch of s bits clearly grows as $O(2^s)$.

Note that the Kirkwood *et al.* validation method prevents this trade-off: Alice always terminates in a non-accepting state unless Bob is able to guess in advance the value $j(\tilde{E}_{AB})$ of the (faulty) shared secret computed by Alice, bringing the probability of successfully recovering the s bits down to $\mu/2^s$.

This trade-off can be analyzed by simulating the attack in software, based on the open-source implementation of [6,16]. The loop-abort is simulated by adding a `break` instruction at the appropriate moment. We use the default parameters that are provided, that is the curve $E : y^2 = x^3 + x$ defined over $\mathbf{F}_p$, for $p = 2^{372}3^{239} - 1$, claimed to offer 186 bits of classical security.

Timings for various batch sizes (*i.e.*, s in the above description) are provided in Table 1. The sizes are even because the implementation of [16] computes 4-isogenies instead of 2-isogenies, so the bits have to be considered by pairs.

size of batches	2	4	6	8	10	12	14	16	18	20
time (in seconds)	7.13	9.6	21.1	65.3	201	681	2389	7946	22809	84763
number of faults	185	93	62	47	37	31	27	24	21	19

Table 1. Time of the computations depending on the size of the batches. The tests were run on an Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz.

5 Countermeasures and conclusion

Previously introduced validation methods are not sufficient to prevent the fault injection attack we have presented. To patch this vulnerability, it is necessary to

implement new countermeasures when such a physical attack is relevant. A few generic countermeasures have already been discussed in [8].

First, simply checking after the loop if the value of the counter is exactly the expected number of iterations provides a first protection against attackers who cannot inject faults with a sufficiently high precision (for instance, attackers who can only inject random errors in a memory word). This countermeasure can be strengthened further by adding an additional (or multiple) parallel counter, and checking that both counters have the expected final value. This protects against single faults, and to some extent, against random faults.

These countermeasures are very cheap and easy to implement, as they are not related to the underlying mathematical structures but are simply meant to check that the loop completed the correct number of iterations.

Acknowledgements

This work has been supported in part by the European Union's H2020 Programme under grant agreement number ERC-669891. The second author was supported by the Swiss National Science Foundation under grant number 200021-156420.

References

1. Azarderakhsh, R., Koziel, B., Jalali, A., Kermani, M.M., Jao, D.: NEON-SIDH: Efficient implementation of supersingular isogeny Diffe-Hellman key-exchange protocol on ARM. Cryptology ePrint Archive, Report 2016/669 (2016), `http://eprint.iacr.org/2016/669`
2. Blake, I.F., Seroussi, G., Smart, N.P.: Advances in Elliptic Curve Cryptography, London Mathematical Society Lecture Note Series, vol. 317. Cambridge University Press (2004)
3. Blömer, J., Gomes da Silva, R., Günther, P., Krämer, J., Seifert, J.: A practical second-order fault attack against a real-world pairing implementation. In: 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2014. pp. 123–136 (2014)
4. Bröker, R.: Constructing supersingular elliptic curves. Journal of Combinatorics and Number Theory 1, 269–273 (2009)
5. Charles, D., Goren, E., Lauter, K.: Cryptographic hash functions from expander graphs. Cryptology ePrint Archive, Report 2006/021 (2006), `http://eprint.iacr.org/2006/021`
6. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Advances in Cryptology - CRYPTO 2016. Proceedings, Part I. pp. 572–601 (2016)
7. De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. J. Mathematical Cryptology 8(3), 209–247 (2014)
8. Espitau, T., Fouque, P.A., Gérard, B., Tibouchi, M.: Loop-abort faults on lattice-based Fiat-Shamir and hash-and-sign signatures. Cryptology ePrint Archive, Report 2016/449 (2016), `http://eprint.iacr.org/2016/449`

9. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Advances in Cryptology - CRYPTO '99. Proceedings. pp. 537–554 (1999)
10. Galbraith, S.D., Petit, C., Shani, B., Ti, Y.B.: On the security of supersingular isogeny cryptosystems. In: Advances in Cryptology - ASIACRYPT 2016. Proceedings, Part I. pp. 63–91 (2016)
11. Galbraith, S.D., Petit, C., Silva, J.: Signature schemes based on supersingular isogeny problems. Cryptology ePrint Archive, Report 2016/1154 (2016), http://eprint.iacr.org/2016/1154
12. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011. Proceedings. pp. 19–34 (2011)
13. Jao, D., Soukharev, V.: Isogeny-based quantum-resistant undeniable signatures. In: Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014. Proceedings. pp. 160–179 (2014)
14. 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 on the security of supersingular isogeny cryptosystems. Workshop on Cybersecurity in a Post-Quantum World (2015), http://csrc.nist.gov/groups/ST/post-quantum-2015/presentations/session7-motley-mark.pdf
15. Koziel, B., Azarderakhsh, R., Kermani, M.M., Jao, D.: Post-quantum cryptography on FPGA based on isogenies on elliptic curves. Cryptology ePrint Archive, Report 2016/672 (2016), http://eprint.iacr.org/2016/672
16. Microsoft Security and Cryptography: SIDH Library (2016), available from https://www.microsoft.com/en-us/research/project/sidh-library/
17. Page, D., Vercauteren, F.: A fault attack on pairing-based cryptography. IEEE Transactions on Computers 55(9), 1075–1080 (2006)
18. Peikert, C.: Lattice cryptography for the internet. In: Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014. Proceedings. pp. 197–219 (2014)
19. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (1997)
20. Silverman, J.H.: The Arithmetic of Elliptic Curves, Graduate Texts in Mathematics, vol. 106. Springer-Verlag, New York, 2nd edn. (2009)
21. Sun, X., Tian, H., Wang, Y.: Toward quantum-resistant strong designated verifier signature from isogenies. In: 2012 Fourth International Conference on Intelligent Networking and Collaborative Systems, INCoS 2012. pp. 292–296 (2012)
22. Tate, J.: Endomorphisms of abelian varieties over finite fields. Inventiones mathematicae 2(2), 134–144 (1966)
23. Vélu, J.: Isogénies entre courbes elliptiques. Comptes Rendus de l'Académie des Sciences de Paris 273, 238–241 (1971)

A When P and Q are not a basis of the torsion

The implementation proposed by [6,16] uses a pair of points P and Q in $E[\ell^k]$ that does not generate the full group $E[\ell^k]$, in order to achieve better compression. The point P is chosen to be a point of order ℓ^k, and Q is set as the image of P by the distortion map $(x, y) \mapsto (-x, iy)$ (where $i^2 = -1$).

They prove that because of this construction, when $\ell = 2$, the sum $P+Q$ has order 2^{k-1} (instead of the expected 2^k). Thus every point of the form $P + [a]Q$

for a even has order 2^k. Caution is required when applying to P and Q results that are meant to be applied to a basis of $E[2^k]$. It appears for instance in [10, Lemma 3.2], where the factor 2^{k-1} should be replaced by 2^{k-2} when using this pair (P, Q).

Also, if a is generated following the guidelines of [6] (as $a = 2m$ for $m \in \{1, 2, \ldots, 2^{k-1}\}$), then its most significant bit is superfluous. Indeed, the kernel of the first isogeny is necessarily the group generated by $[2^{k-1}]P = -[2^{k-1}]Q$. Then, the image of $P + [a]Q$ under this isogeny is the same as the image of $P + [a + 2^{k-1}]Q$. It follows that the secret a leads to the same shared secret as its reduction $a \bmod 2^{k-1}$. Therefore the secret $a = 2m$ could be chosen with $m < 2^{k-2}$.