Practical Fault Injection Attacks on Constant Time CSIDH and Mitigation Techniques

Tinghung Chiu
Virginia Tech
Blacksburg, VA, USA
kennychiu0818@vt.edu

Jason LeGrow
Virginia Tech
Blacksburg, VA, USA
jlegrow@vt.edu

Wenjie Xiong*
Virginia Tech
Blacksburg, VA, USA
wenjiex@vt.edu

ABSTRACT

Commutative Supersingular Isogeny Diffie-Hellman (CSIDH) is an isogeny-based key exchange protocol which is believed to be secure even when parties use long-lived secret keys. To secure CSIDH against side-channel attacks, constant-time implementations with additional dummy isogeny computations are employed. In this study, we demonstrate a fault injection attack on the constant-time real-then-dummy CSIDH to recover the full static secret key. We prototype the attack using voltage glitches on the victim STM32 microcontroller. The attack scheme, which is based on existing research which has yet to be practically implemented, involves getting the faulty output by injecting the fault in a binary search fashion. Our attack reveals many practical factors that were not considered in the previous theoretical fault injection attack analysis, e.g., the probability of a failed fault injection. We bring the practice to theory and developed new complexity analysis of the attack. Further, to mitigate the possible binary search attack on real-then-dummy CSIDH, dynamic random vector CSIDH was proposed previously to randomize the order of real and dummy isogeny operations. We explore fault injection attacks on dynamic random vector CSIDH and evaluate the security level of the mitigation. Our analysis and experimental results demonstrate that it is infeasible to attack dynamic random vector CSIDH in a reasonable amount of time when the success rate of fault injection is not consistent over time.

CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures**; **Tamper-proof and tamper-resistant designs**.

KEYWORDS

Fault injection attack, real-then-dummy CSIDH, dynamic random vector CSIDH, post-quantum cryptography

ACM Reference Format:
Tinghung Chiu, Jason LeGrow, and Wenjie Xiong. 2024. Practical Fault Injection Attacks on Constant Time CSIDH and Mitigation Techniques. In *Proceedings of the 2024 Workshop on Attacks and Solutions in Hardware Security (ASHES '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3689939.3695789

*Author list in alphabetical order.

1 INTRODUCTION

Isogeny-based cryptography is a promising branch of post-quantum cryptography, whose security based on the difficulty of finding certain structured maps (isogenies) between elliptic curves. These protocols have extraordinarily small communication overhead compared with most other post-quantum protocols, which makes them excellent candidates for scenarios when communication complexity outweighs computation time when determining protocol latency, such as when communicating over unreliable networks. There have been two frameworks proposed for isogeny-based key establishment protocols: Supersingular Isogeny Diffie-Hellman (SIDH) [17] and Commutative SIDH (CSIDH) [13]. For a time, the active attacks [20, 22] on SIDH meant that CSIDH was the only candidate for isogeny-based static/ephemeral key establishment. Later, total breaks [12, 31, 37] rendered SIDH-based protocols completely insecure, leaving CSIDH—which, despite cryptanalysis [6, 7, 26, 36], remains secure—as the only isogeny-based option for key establishment. As well, CSIDH is a very flexible protocol, and is currently the most viable isogeny-based option for many advanced public-key functionalities [5, 18, 27, 40]. For these reasons, CSIDH is one of the most ubiquitous and important protocols studied in the isogeny-based cryptographic literature.

The original implementation of CSIDH was not constant time, and hence was vulnerable to timing attacks. A series of papers [14, 16, 25, 32, 35] introduced and optimized versions of CSIDH which use **dummy operations** to achieve constant running time. In short, these constant-time implementations of CSIDH always construct the same number of isogenies, regardless of the value of the secret key, and the results of the dummy computations are simply discarded. While this does make the protocol constant-time, it has the unfortunate side effect of creating a new attack possibility: a fault injection attack. In particular, in the static/ephemeral setting where one party has a fixed secret key, an attacker can repeatedly initiate key establishment sessions and introduce faults into the isogeny computations. When a fault is injected into a real isogeny, we expect that the output (the shared key) will be incorrect, while injecting a fault into a dummy isogeny should not disrupt the shared key. In this way, the attacker can determine the number of real isogenies of each degree, which is equivalent to determining the static secret key (possibly up to sign, depending on the implementation).

While fault attacks on static/ephemeral CSIDH have been the subject of many theoretical studies [2, 10, 14, 29, 30], these works do not consider many practical aspects of fault injection attacks, such as the possibility that a fault injected into a real isogeny will fail to produce faulty output, the probability that injecting an isogeny into a real or dummy isogeny will cause the program to fail to output anything at all, or the parameters that must be tuned to improve fault attack effectiveness (such as when and for how long to inject faults into isogeny computations). While theory informs the design and analysis of these fault attacks, significantly more experimental work must be done in order to understand the impact that these attacks have on the security of CSIDH in real-world situations.

This paper makes the following contributions:

(1) We demonstrate a voltage glitch attack on real-then-dummy constant-time CSIDH with signed private keys. We are the first to describe the complete attack pipeline. We perform extensive experiments to analyze and optimize this success probability for key recovery by finding suitable injection time (tuned to each isogeny degree separately), fault duration, and voltage.

(2) Building on our experimental evidence and considering the success rate of single fault injection, we identify the gap between the existing theory and practice, and derive *new* bounds on the number of faults required to achieve high success probability when the isogenies are in real-then-dummy order, or use dynamic random vector ordering as in [30].

(3) We are the *first* to bring the theory of [30] to practice, by incorporating the dynamic random vector isogeny reordering technique into our implementation of CSIDH, and trying to launch a fault attack on this implementation. Our experiments reveal previously unknown complications that arise when analyzing the results of fault attacks on CSIDH when isogenies are ordered randomly in each session. These results suggest that the mitigation techniques of [30] may be **substantially more secure** then originally believed.

Our dynamic random vector isogeny reordering CSIDH implementation and voltage glitch attack control code is **open-sourced** at https://github.com/bearhw/RandomVector_CSIDH_FI.

2 BACKGROUND

2.1 CSIDH

We provide only a cursory explanation of the relevant background; for a more complete explanation, see [13, Section 3].

Fix a prime p of the form $p = 4\ell_1\ell_2\cdots\ell_n - 1$ where $\ell_1, \ell_2, \ldots, \ell_n$ are distinct small odd primes—usually, $\ell_1, \ldots, \ell_{n-1}$ are taken to be the first $n-1$ odd primes, and ℓ_n is the smallest prime which is larger than ℓ_{n-1} and for which the resulting p is itself prime. Let $\mathcal{E}_p$ denote the set of supersingular elliptic curves defined over $\mathbb{F}_p$ and whose $\mathbb{F}_p$-rational endomorphism ring is isomorphic to $O = \mathbb{Z}[\sqrt{-p}]$ (this set is nonempty since it contains, for instance, $E_0\colon y^2 = x^3 + x$). By [13, Theorem 7], the ideal class group $\mathrm{cl}(O)$ acts freely and transitively on $\mathcal{E}_p$.

By our choice of p, there are a number of distinguished elements of $\mathrm{cl}(O)$ that are used in CSIDH. In particular, each principal ideal (ℓ_i) of O splits as $(\ell_i) = \mathfrak{l}_i\bar{\mathfrak{l}}_i$, where $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\bar{\mathfrak{l}}_i = (\ell_i, \pi + 1)$, where π is the Frobenius map. The action of the corresponding ideal classes $[\mathfrak{l}_i]$ and $[\bar{\mathfrak{l}}_i] \in \mathrm{cl}(O)$ can be computed efficiently using Vélu's formulas [41], since the required torsion points can be chosen so that their x-coordinates are in $\mathbb{F}_p$. Since (ℓ_i) is principal, we have $[\bar{\mathfrak{l}}_i] = [\mathfrak{l}_i]^{-1}$ in $\mathrm{cl}(O)$. With this setup, the CSIDH key establishment protocol proceeds as follows: Alice selects $\vec{e}^A \in [-b, b]^n$, where b is a small, predetermined integer chosen to ensure sufficient security of the protocol. Alice constructs the elliptic curve $E_A = ([\ell_1]^{e_1^A}[\ell_2]^{e_2^A}\cdots[\ell_n]^{e_n^A}) * E_0$, where $E_0\colon y^2 = x^3 + x$. Bob proceeds analogously, choosing $\vec{e}^B$ and constructing E_B. Alice and Bob exchange E_A and E_B. To construct the shared key, Alice computes $([\ell_1]^{e_1^A}[\ell_2]^{e_2^A}\cdots[\ell_n]^{e_n^A}) * E_B = ([\ell_1]^{e_1^A+e_1^B}[\ell_2]^{e_2^A+e_2^B}\cdots[\ell_n]^{e_n^A+e_n^B}) * E_0$ and Bob analogously computes $([\ell_1]^{e_1^B}[\ell_2]^{e_2^B}\cdots[\ell_n]^{e_n^B}) * E_A$, which is equal to Alice's key by the commutativity of $\mathrm{cl}(O)$.

2.2 Constant-Time Implementations of CSIDH with Dummy Isogenies

In a naïve implementation of the protocol described in Section 2.1, Alice would compute exactly $\|\vec{e}^A\|_1 = |e_1^A| + |e_2^A| + \cdots + |e_n^A|$ isogenies to construct the shared secret. In this case, Alice's computation time would leak information about her secret key $\vec{e}^A$; if she constructs the shared secret relatively quickly, it's likely that $\|\vec{e}^A\|_1$ is small, for instance. The standard mitigation technique is to ensure that Alice always computes exactly b isogenies of each degree: she computes the $|e_i^A|$ isogenies of degree ℓ_i as usual, but also computes $b - |e_i^A|$ more isogenies of degree ℓ_i, whose outputs she ignores. These additional isogenies are called "dummy" isogenies.

Real-then-Dummy Implementation. Meyer, Campos, and Reith were the first to implement CSIDH with dummy isogenies [32]. In this implementation, all $|e_i|$ real isogenies of degree ℓ_i are computed first, followed by all $f_i = b - |e_i|$ dummy isogenies of degree ℓ_i. We call this the real-then-dummy setting; the algorithm is depicted in Algorithm 1 in the Appendix. LeGrow and Hutchinson [30] showed that, in a model where fault injections in real isogenies always causes a faulty key, an attacker can learn $|e_i^A|$ using $\log_2(b+1)$ faults using a binary search approach, and so can learn the magnitudes of all key values using $n\lceil\log_2(b+1)\rceil$ faults in total.

Reordered Isogenies. LeGrow and Hutchinson [30] proposed variants of constant-time CSIDH in which the order of real and dummy isogeny is randomized, preventing the binary search method described in Algorithm 2 in Appendix A. In particular, they consider two regimes for reordering the isogenies, which they call *fixed* and *dynamic*. In the fixed regime, the ordering of the real and dummy isogenies is encoded in a vector—the *decision vector*—which is chosen when the private key is sampled and is fixed for all time. In the dynamic regime, the decision vector is chosen fresh and uniformly at random each time key establishment is initiated. These isogeny reordering techniques introduce minimal overhead. While the fixed decision vector has only a small impact on the number of faults required for a successful fault injection attack, dynamic reordering has a more dramatic impact: the number of faults required for a successful attack increases from $\Theta(\log b)$ to $\Theta(b^2 \log b)$. Although this is an exponential increase from the original attack, the parameter b is polynomial in the security parameter, and so these attacks still run in polynomial time.

2.3 Fault Injection Attacks

A fault injection attack is an active attack where the attacker intentionally induces a fault on the device during the execution of security-critical operations. The attacker can analyze the faulty outputs caused by the fault during the operation, resulting in compromising security measures or extracting sensitive information such as a secret key.

Inducing a Fault. The fault injection attacks typically pose the target device in an operation condition that is not normal, so the device may enter some unstable state to have errors in the execution. There are a number of fault injection methods, such as electromagnetic pulse [19, 33], laser [38, 39], voltage glitch [3, 8], DRAM row hammer [23, 34], etc. In the experiments of this paper, we use voltage glitches to induce faults, which are easy and inexpensive to set up. It has shown that voltage glitches have relatively high resolution in time.

Meanwhile, the unstable state does not guarantee errors, i.e., the fault injection may not always yield an error in the execution. The attacker may need to try several times to induce an actual error. In this paper, we define **faulty rate (p_f)** as the probability the attacker successfully induces an actual error in the execution on an attempt to induce a fault. Also, the unstable state might trigger other errors in the system or trigger a protection mechanism to reset the whole system, and the victim execution might not be complete to generate the desired faulty output. We define **reset rate (ρ)** as the probability the attacker triggers the system reset or crash in an attempt to induce a fault. Although in this paper we demonstrate a fault injection attack with voltage glitches, the unstable nature also exists in other type of fault injection attacks.

Fault Analysis. The most widely used fault analysis method is *differential fault attack (DFA)*. In the attack, the cryptographic operation is executed twice, once to generate the correct output and once with an induced fault at a certain instruction or memory location to generate a faulty output. The attacker then uses the faulty and the correct output to extract the key [28].

Statistical Fault Attack (SFA) is another fault analysis method. The attacker induces faults in all the execution runs, and analyzes the distribution of the faulty outputs [21]. As a special case, the output can be whether the execution failed or not, i.e., depending on the secret an actual fault does not result in a faulty execution output and can be seen as a "safe error". One example of such an attack is the constant time RSA implementation [42]. Due to the dummy operation in constant-time CSIDH, CSIDH can be vulnerable to safe error, and this is the focus of this paper.

It is possible that the attacker intends to inject a fault but the execution is not actually perturbed, i.e., the faulty rate is less than one. In DFA, this is not an issue, because the attacker can observe a faulty output when a fault is induced successfully. If the attacker does not observe a faulty output, the attacker can simply retry the fault injection until a faulty output is observed. However, in SFA, the attacker's observation is directly affected by the faulty rate of the fault injection, and the attacker has to take the faulty rate into consideration during the faulty analysis. In this paper, we consider the results of the empirical results of real fault injection (i.e., faulty rate and reset rate) into the fault analysis process.

3 RELATED WORKS

Attack Models of Fault Attacks on CSIDH. Most theoretical analyses of fault attacks on CSIDH [2, 29, 30] use the following abstract model of fault injection: an attacker is able to disrupt an isogeny computation in such a way that the output curve is uniformly random, rather than the correct curve. There are several models of how much information can be obtained about the isogeny can be disrupted; for instance, in the the weak "shotgun at the CSIDH" model of [10], the attacker knows nothing about the degree of the isogeny which is disrupted or in which round it occurs. In this weak attack model, the attacker is only able to learn aggregate information about the proportion of isogenies that are real, reducing the feasible keyspace slightly. Most works consider the stronger, but still quite reasonable attack model in which an attacker can disrupt an isogeny of known or chosen degree in a given round. While the theoretical analysis consider the case when the fault injection is perfect (i.e., a fault injected into a real isogeny always yields incorrect output, $p_f = 1$), our experiments demonstrate that this is not a realistic model. A particular consequence of this is that some proposed fault attack countermeasures—particularly the "dynamic random vector" proposal of [30]—are more effective than the original analysis suggests.

Empirical fault attacks on CSIDH. The first published work which contains experimental results is [10], in which the authors use voltage glitches to corrupt isogeny output. Our attack is different from [10] in the following fundamental ways: (1) In their attacks other than their weak attack model 1, the attacker can induce error accurately at certain critical instructions of a desired isogeny, but we assume a more realistic scenario that the attacker only knows when the isogeny operation starts. Our assumption is more realistic, because if an attacker can know all the instructions being executed, the attacker can already identify if the isogeny is real without fault injection. (2) The experiments of [10] are limited: the keyspace size is significantly reduced to only 3^2, fault injection is tested only for two prime isogeny degrees 3 and 5, only the real-then-dummy isogeny order is considered, and only the final percentage of faulty output is reported—the complete attack pipeline is not analyzed. We conduct extensive experiments to analyze the confidence and complexity of the attack and explore mitigation such as randomly reordered isogenies. (3) In their theory analysis, they assume the attacker can reliably induce a fault without considering the success rate of fault injection. The only other experimental work is [11], which uses voltage glitches to corrupt a specific assignment instruction of certain variables used in isogeny computations, to cause faulty output if there is an fault in the dummy isogeny and correct output for real isogeny. The work of [11] suffers from the same drawbacks as [10], and only shows experimental results for attack single isogeny operations instead of a complete CSIDH operation.

Countermeasures of fault attacks on CSIDH. There have been a number of theoretical works studying both fault injection attacks and their countermeasures in the context of CSIDH. The first work to consider fault attack and their countermeasures is [14], where the "dummy-free" method is introduced (and later optimized in [16]). This technique uses a modified keyspace, and essentially replaces dummy isogenies with pairs of real isogenies that cancel one another out. The modified keyspace results in a protocol with a running time roughly *double* that of a similar implementation which uses dummy isogenies. This method totally prevents fault attacks, in principle. In 2023 LeGrow proposed the two-curve method [29]. Like the dummy-free method of [14], all isogenies computed in this setting are real. Isogenies which would have been real in the naïve implementation are performed as usual, while isogenies which would have been dummy are applied to a second

Table 1: A summary of existing fault attack countermeasures. The bold ones are studied in this paper

Implementation	Latency (Mcycles)	Public Key Size (Bytes)	Generic?	Fault attack prevention
Variable time [13]	106	64	✓	✗
Optimized constant time [25]	229	64	✓	✗
Optimized dummy-free [14, 16]	431	64	✓	✓
Two-curve method [29]	417	128	✓	✓
Consistency checks [10]	245[1]	64	✗	✓[2]
Pseudo y-coordinates [2]	242[1]	64	✓	✓[2]
Reordered isogenies [30]	229[1]	64	✓	✗[3]

[1]Estimated.
[2]In a less general fault model.
[3]Fault attack complexity is significantly increased, but attacks are not fully prevented.

curve; in the end, both curves are used to construct a shared secret key. This protocol introduces less computational overhead than the dummy-free method, but the communication complexity of the protocol doubles.

Taking a different approach, the authors of [10] propose to thwart fault attacks not by removing dummy isogenies, but by adding additional consistency checks to the isogeny computations. These checks cause the protocol to abort if an isogeny is tampered with, regardless of whether it is real or dummy. This is a very lightweight countermeasure, introducing only between 5% and 7% overhead. Unfortunately, this countermeasure is not generic, and in particular, it is not compatible with the √élu technique[1] which has been used to speed up isogeny computations in CSIDH considerably [4]. Similarly, in [2], the authors consider a fault attack model in which an attacker is explicitly targetting the `IsSquare` function which is present in most CSIDH implementations—while the authors provide countermeasures for these fault attacks, this is also not a generic model of fault attack on CSIDH. These countermeasures and their properties are summarized in Table 1.

Table 1 explains why fault injection attack countermeasures for CSIDH have not seen widespread adoption: the methods that prevent fault injection attacks introduce substantial overhead, while the methods which are efficient either do not apply to all implementations of CSIDH, or merely mitigate (rather than fully prevent) fault injection attacks. In this paper, we study the two constant-time implementations with low latency, bold in Table 1. Especially, we leverage empirical results to analyze the security level of Reordered isogenies [30], which has almost no performance overhead.

4 THE ATTACK MODEL

In the static/ephemeral setting, Alice has a long-lived public key $E_A = [\prod_{i=1}^{n} \ell_i^{e_i^A}] * E_0$ which she uses for a key encapsulation mechanism in the style of Elgamal. In particular, suppose that Bob wishes to communicate with Alice using a symmetric-key cryptosystem. To establish a shared session key with Alice, he will choose an ephemeral CSIDH secret $\vec{e}^B$, construct the corresponding ephemeral public key E_B and shared secret E_{BA}. He then chooses a uniformly random key $k \in \{0,1\}^\kappa$ and sends $(E_B, s = h(M(E_{BA})) \oplus k)$ to Alice, where $M(E_{BA}) \in \mathbb{F}_p$ is the Montgomery coefficient of E_{BA} and $h\colon \{0,1\}^* \to \{0,1\}^\kappa$ is a cryptographic hash function. Upon receiving (E_B, s) from Bob, Alice constructs E_{AB} using her long-lived secret key, and hence recovers $k = s \oplus h(M(E_{AB}))$—Alice and Bob now have the shared key k.

An attacker Eve who wants to learn Alice's long-lived secret proceeds as follows: Eve repeatedly establishes session keys with Alice as described above and injects faults into Alice's computations. Upon receiving the first message from Alice, if Eve tires to inject a fault when Alice constructs the shared curve, then with overwhelming probability she will recover the incorrect symmetric key, and Eve can detect this by decrypting Alice's first message, since the decrypted message will, with overwhelming probability, not be semantically meaningful. With the attack described in the next section, Eve can eventually learn Alice's long-lived secret $\vec{e}^A$.

The following statements describe the assumptions that the attacker requires to recover the secret key from the target device:

Offline profiling: We assume the attacker Eve has physical access to the target device used by the victim Alice for the voltage glitch attack. To find the appropriate parameters for fault injection, Eve must be able to profile the CSIDH execution on the target device or a device of the same model.

Knowing the parameters of CSIDH: We assume that adversary knows the system parameters; in particular, the key bound b, and the order in which the isogenies will be computed in each round. This information is not sensitive, and so it may be public. But it also does not need to be public—indeed, two parties participating in key establishment need not use the same key bound and isogeny ordering. Even if this information is not published, the attacker can reverse engineer these parameters. For simplicity, we assume that the adversary simply knows this information. The parameter of CSIDH is not a secret of the algorithm. Meanwhile, the attacker can do the reverse engineering or compare the execution time of each degree isogeny after profiling the target device to know the order of prime numbers and the max value of each degree isogeny.

Triggering CSIDH key exchange repeatedly: As the secret key recovery necessitates multiple shared secrets, the attacker should be able to trigger CSIDH key exchanges repeatedly with the target device. e.g., Eve can repeatedly establish session key with Alice.

Knowing when CSIDH key exchange starts: Since the attacker can trigger CSIDH key exchange, we assume the execution would start immediately, and the attacker can use that as a reference time for fault injection.

Injecting faults during isogeny execution: The attacker will induce faults by voltage glitches or other fault injection methods, potentially resulting in obtaining a faulty shared secret to recover the secret key. We assume the attacker knows when isogeny computation starts by profiling the CSIDH execution. In contrast with prior works [10, 11], we do not assume the attacker can or must accurately inject fault into specific instructions.

Observing if CSIDH key exchange fails: The attacker can repeatedly participate in key establishment sessions with Alice, and determine whether Alice has constructed the correct shared curve.

5 ATTACK SCHEME

The basic idea of dummy-based constant time CSIDH secret key recovery by fault injection attack is based on observing whether induced fault will impact the shared secret. Assuming the attacker can induce the fault during the isogeny operation of each degree.

[1]Pronounced "square root Vélu," this improves on Vélu's formulas, reducing the complexity of computing an ℓ-isogeny from $\tilde{\Theta}(\ell)$ operations in $\mathbb{F}_p$ to $\tilde{O}(\sqrt{\ell})$.

The final shared secret will not be impacted if the induced fault occurs during the dummy isogeny operation. Otherwise, when the fault occurs during the real isogeny operation, the faulty calculated curve will be propagated, and the attacker will obtain a different shared secret. Then, the attacker can infer the number of real isogeny operations in each degree, which is Alice's secret $\vec{e}^A$. Therefore, we first focus on inducing the fault during the isogeny operation.

5.1 Voltage Glitches during Isogeny Operation

There are several parameters that the attack requires to achieve the goal of inducing the fault by voltage glitch during the isogeny operation: (1) **Injection Time:** The point in the algorithm in which we inject the fault. (2) **Duration:** The duration of time the voltage is lowered. (3) **Voltage:** The extent to which the voltage is changed during the glitch. The voltage and duration of the glitch have an impact on the number of instructions that are disrupted. These parameters will directly affect the accuracy of the recovery key and must be determined appropriately.

5.1.1 Empirical Study of Fault Injection on Isogeny Operations. While previous theoretical fault analysis assumes fault injection in any part of the real isogeny should induce fault in the output [2, 10, 14, 29, 30], here, we study the effects of voltage glitches in different parts of the real isogeny empirically. We conduct test on a microcontroller, more details about the injection setup can be found in Section 6.1. We use the glitch duration and voltage that results in the best faulty rate to test different injection positions. Our fault injects position focuses on the main for-loop operation, which accounts for 92% of the isogeny operation. We divide the execution time of the for loop inside the isogeny operation into a hundred points, from 0% to 99% execution time, and take 10 trials for each point. The fault injection results for degree 353 and 359 are shown in Figure 1 and Figure 2. The results of more degrees can be found in Figure 5 in the appendix.

There are three possible outputs when inducing the fault during the isogeny operation. If the shared secret get tampered by the fault, we call it **faulty output**. Another is **correct output**, that the fault does not change the shared secret. The last one is **reset**, where the board does not output a shared secret within a reasonable time due to the system failure.

In Figure 1 and Figure 2 the x-axis indicates the injection time being the percentage of the execution time of for-loop, and the y-axis the percentage of faulty output and reset. Figure 1 is the result of fine-grained search on different degree isogeny operations, which we use to determine the injection time. It shows that the distribution of faulty and reset rates is highly different between each degree of isogeny operation. Moreover, Figure 2 shows that the results of fine grain search may be different between different rounds of profiling, even though inducing the fault on the same degree isogeny operation.

5.1.2 Challenges for Injecting Faults in the Isogeny. In our voltage glitch experiments, we observed the following features that made key recovery using fault injection attacks challenging:

The faulty rate for real isogeny is not high. In the previous theoretical papers which analyze fault injection attacks on CSIDH,

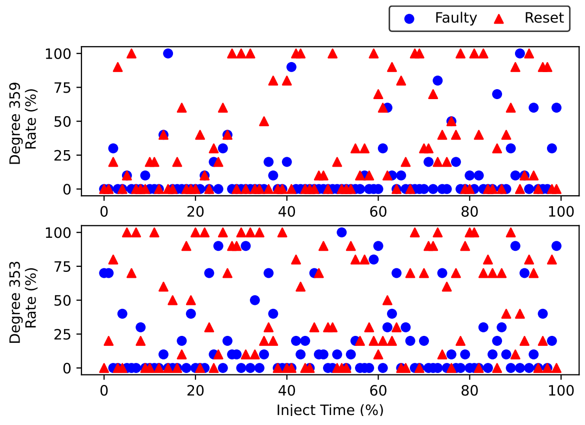

Figure 1: Faulty and reset rate when inducing faults on real isogenies of degrees 359 and 353.

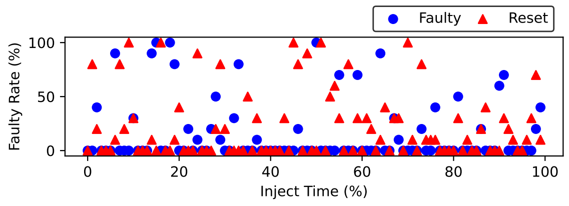

Figure 2: Faulty rate of isogenies of degree 359.

the models assume that the injecting a fault into a real isogeny always yields incorrect output (i.e., the faulty rate is 100%). However, Figure 1 shows that at most injection points in an isogeny, injecting a fault will not cause faulty output with high probability—indeed, most injection points have a 0% faulty rate and a high reset rate, which will make the fault injection attack have less accuracy of recovery key and increasing the time required for the end-to-end attack.

Fault injection parameters depend on isogeny degree. Figures 1 and 5 show profiling results for isogeny computations of five different degrees. The faulty rates distributions are very different among the different degrees, meaning that the attacker must profile each isogeny degree separately.

The reset rate is not low. A fault could make a system unresponsive, in which case the attacker has to retry. The reset rate is related to the number of fault injections (and hence time consumption) of an end-to-end attack. As shown in Figures 1 and 5, the board resets frequently—especially if the fault injection point is not chosen well. The attacker should consider faulty and reset rates when choosing the appropriate variables for inducing faults.

The faulty rate is not necessarily consistent across time. While isogenies of different degrees have different appropriate variables, even the same degree isogeny operations may have different faulty rates over time. Figure 2 shows the result fine-grained search for optimal fault injection parameters on isogenies of degree 359—note the differences from Figure 1, which is a different round of testing. The degree-359 result in Figure 1 has more points with high faulty and reset rate when inducing the fault after 60% execution time.

However, the result in Figure 2 has more points high faulty and reset rate when inducing the fault before 40%. The difference between the two results indicates that the attacker may obtain different appropriate variables across time, increasing the attack difficulty. **Determining appropriate fault injection parameters.** Figure 1 shows that the attacker needs to carefully choose the injection time to achieve high faulty rate. In addition to the injection time, attackers also need to profile to find the optimal glitch parameters such as voltage and duration.

5.2 End-to-end Attack

The following two sections describe the end-to-end attack on real-then-dummy CSIDH and dynamic random vector CSIDH. We split the end-to-end attack into two phases, the profiling phase and the online phase. The profiling phase is the phase that extracts the appropriate variables we have indicated important in the previous section, and the online phase is the phase that processes the fault injection attack on the victim device.

5.2.1 *Real-then-Dummy CSIDH.*

Profiling phase: To determine the parameters for voltage glitches, the attacker should profile the fault injection parameters on the profiling device, which has an identical structure to the victim device. In our experiments, we never observe a faulty output if a fault is injected in a dummy isogeny, and thus the faulty rate on the real isogeny is the metric we want to optimize for. To profile the real isogeny operation, the possible complexity of searching the best variables could be $t_d \cdot v \cdot N_{dv} \cdot I \cdot N_I$, where t_d is the number of duration we test, v is the number of voltage values we test, I is the number of injection time points, N_{dv} is the number of the trials we conduct at each (voltage, duration) pair, and N_I is the number of the trials we conduct at each injection time.

However, we found the duration and voltage are relatively independent of the injection time, which has different effects on fault injection attacks. Therefore, we search them independently. The number of total required tests to obtain the appropriate variables is $t_d \cdot v \cdot N_{dv} + I \cdot N_I$.

Attack phase: After determining the appropriate parameters for inducing faults in the isogeny operations, the attacker can induce faults on the target device, and determine the secret key by observing the outputs. In real-then-dummy CSIDH, the order of real and dummy isogeny operations are fixed, and dummy isogeny operations follow the real isogeny operations. The attacker can recover the key entry magnitude $|e_i|$ by repeatedly inducing the fault with the binary search method during the isogeny operations to determine in which round the first dummy isogeny operation occurs, as depicted in Figure 3. The complete algorithm is in Algorithm 3 in the appendix. If the private key uses unsigned integers, this is sufficient to recover the private key. If the private key uses signed integers, the attacker can use meet-in-the-middle approach to recover the signs of the e_i's [30, Section 3.3].

In contrast with the theoretical analysis of [30], in practice we should account for the probability p_f that injecting a fault into a real isogeny will lead to an incorrect shared secret. When $p_f < 1$, we will repeatedly inject faults into a single isogeny across multiple key establishment sessions. When we observe an incorrect

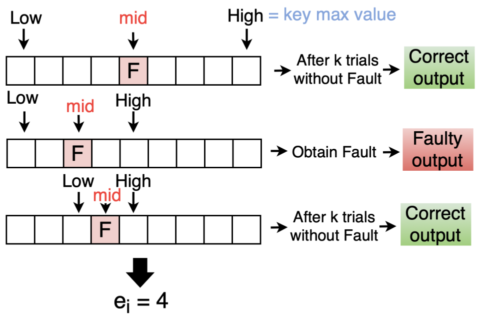

Figure 3: Binary search for the secret key.

output, we know that the isogeny was real, while if we observe no incorrect outputs after some number k of trials, we believe that the isogeny is dummy. In this way we always correctly identify dummy isogenies, and correctly identify real isogenies with probability $1-(1-p_f)^k$. To determine $|e_j|$ using binary search requires $\lceil \log_2 b \rceil$ rounds of this process, and so the attack correctly recovers $|e_j|$ with probability $(1-(1-p_f)^k)^{\lceil \log_2 b \rceil}$ in the worst case. When recovering n key magnitudes, the success probability reduces to $(1-(1-p_f)^k)^{n\lceil \log_2 b \rceil}$; overall, we must take

$$k > \frac{\ln\left(1-(1-\varepsilon)^{\frac{1}{n\lceil \log_2 b \rceil}}\right)}{\ln(1-p_f)}$$

to achieve success probability $1-\varepsilon$ in recovering the entire key magnitudes—for our observed $p_f \in [0.427, 0.714]$ and CSIDH parameters $n = 74, b = 5$ (in the signed setting), we see that $k = 10$ injections per isogeny suffices to achieve success probability $\frac{1}{2}$. In our experiments with $n = 2$, it suffices to take $k = 4$.

Complexity of the attack: We use the expected total number of fault injections required (denoted by T) to represent the complexity. For each fault injection, a complete CSIDH key exchange needs to be run. With k fixed, T satisfies $\frac{n\alpha}{1-\rho}\lceil \log_2 b \rceil \le T \le \frac{nk}{1-\rho}\lceil \log_2 b \rceil$, where ρ is the probability that injecting a fault causes the hardware to reset and $\alpha = p_f \sum_{j=1}^{k-1} j \cdot (1-p_f)^{j-1} + k(1-p_f)^{k-1}$ is the expected number of faults that will be injected into a real isogeny, until it is either correctly identified as real or incorrectly identified as dummy. For our observed $p_f \in [0.427, 0.714]$, $\rho = 0.209$, and CSIDH parameters $n = 74$, $b = 5$ we take $k = 10$ to achieve success probability $\frac{1}{2}$. In this regime expected total number of fault injections required satisfies $2506 \le T \le 10622$. In our experiments with $n = 2$ and $k = 4$, the expected total number of fault injections required satisfies $69 \le T \le 114$.

5.2.2 *End-to-end Attack on dynamic random vector CSIDH.*

Profiling phase: The same as in Section 5.2.1.

Attack phase: The order of real and dummy isogeny operations is not fixed in dynamic random vector CSIDH (Algorithm 2 in appendix), and each CSIDH will use a fresh random order V. This means the attacker can not recover the absolute value $|e_i|$ by locating the first dummy isogeny operation like in real-then-dummy CSIDH.

Nonetheless, the attacker can recover the absolute value $|e_i|$ by injecting multiple faults and analyzing the ratio of correct and faulty shared secrets, whose value depends on $|e_i|$. After recovering the value of $|e_i|$, if the private key is signed integers, the attacker can use a meet-in-the-middle approach to find the signs of it.

In [30], the authors prove that if the the key value e_j is chosen between 0 and b, and fault injection is perfect—that is, if injecting a fault in a real isogeny always yields incorrect output, and injecting a fault in a dummy isogeny always yields correct output $p_f = 1$—then $2b^2 \ln \frac{2}{\delta}$ faults are sufficient to learn the value of e_j with probability $1-\delta$. In particular, if an attacker observes f incorrect outputs and c correct outputs, and $f + c > 2b^2 \ln \frac{2}{\delta}$, then $e_j = \left\lceil b \frac{f}{f+c} \right\rfloor$ with probability at least $1-\delta$. Unfortunately, this analysis does not apply in a straightforward fashion in the setting where the fault injection is imperfect. In particular, in our experiments, we observe that while injecting a fault into a dummy isogeny, we always observe correct output, the converse is not true—that is, injecting a fault into a real isogeny may still yield correct output. This creates two problems: first, our estimate for the key value (as a function of the number of faults injected and the number of incorrect outputs observed) must be updated; and second, the number of faults required to achieve a given level of confidence will change.

To begin, we rephrase the original formula used to estimate e_j. If we inject faults into t isogenies and see f incorrect outputs and c correct outputs, the reason that we estimate $e_j = \left\lceil b \frac{f}{f+c} \right\rfloor$ is because we believe that f of the isogenies were real, and c were dummy. If r and d are the number of real and dummy isogenies into which we inject faults, respectively, then of course in expectation, $\frac{r}{r+d} = \frac{e_j}{b}$. In the setting of perfect faults, we know that $r = f$ and $d = c$, but in the setting of imperfect fault, we require new estimates. Let p_f be the probability that injecting a fault into a real isogeny yields an incorrect output, and q be the probability that injecting a fault into a dummy isogeny yields correct output. Assuming that a sufficiently large number of faults are injected, we expect

$$\begin{cases} f \approx p_f \cdot r + (1-q) \cdot d \\ c \approx (1-p_f) \cdot r + q \cdot d \end{cases} \implies \begin{cases} r \approx \frac{1}{p_f+q-1} \left(q \cdot f - (1-q) \cdot c \right) \\ d \approx \frac{1}{p_f+q-1} \left(-(1-p_f) \cdot f + p_f \cdot c \right) \end{cases}$$

Thus, in this setting, we estimate $\frac{e}{b} \approx \frac{r}{r+d} = \frac{q \cdot f - (1-q) \cdot c}{(p_f+q-1)\cdot(f+c)}$. In practice, we have $q = 1$, which yields the much simpler formula $\frac{e_j}{b} \approx \frac{f}{p_f(f+c)}$, and we use this formula to estimate key values in our attacks in this setting. With this new formula, much the same analysis as in [30] applies. For fixed values of e_j and p_f, the probability that injecting a fault into an isogeny of degree ℓ_j yields incorrect output is $\frac{p_f e_j}{b}$, and so in T total injections, the expected number of incorrect outputs is $\mathbb{E}[f] = \frac{p_f t e_j}{b}$. The naïve estimate $e_j = \left\lceil \frac{bf}{p_f t} \right\rfloor$ is correct as long as $\left| f - \frac{p_f t e_j}{b} \right| < \frac{p_f T}{2b}$, which by Hoeffding's inequality [24] holds with probability at least

$$\mathbb{P}\left[\left| f - \frac{p_f t e_j}{b} \right| < \frac{p_f T}{2b} \right] \geq 1 - 2\exp\left(-\frac{T}{2} \left(\frac{p_f}{b} \right)^2 \right).$$

Complexity of the attack: To achieve success probability $1-\delta$ in determining e_j, it suffices to inject $T \geq \frac{2b^2}{p_f^2} \ln \frac{2}{\delta}$ faults into degree-ℓ_j isogenies; to achieve success probability $1-\varepsilon$ of recovering n key values correctly, it suffices to inject $nT \geq \frac{2nb^2}{p_f^2} \ln \frac{2}{1-\sqrt[n]{1-\varepsilon}}$ faults in total. In our experiments, we consider $b = 5$ and observe $p_f \in [0.427, 0.714]$. In the full CSIDH-512 setting we have $n = 74$, and so this formula implies that 1.09×10^5 successful fault injections (i.e., fault injections that do not lead to a board reset) are required to recover the entire secret key magnitudes with probability $\frac{1}{2}$. In our experimental setup we take $n = 2$, for which 1053 successful faults are required.

6 EXPERIMENTAL EVALUATION

6.1 Physical Setup

Our experiments use STM32F407, a 32-bit microcontroller (MCU) with ARM Cortex-M4 processor as the victim device. We use a voltage glitch controller from Riscure and develop a Python script to communicate with the controller from the host PC (instead of the inspector software from Riscure), which arming the voltage glitch to the board results in injecting the fault. The physical setup for our experiments is shown in Figure 4.

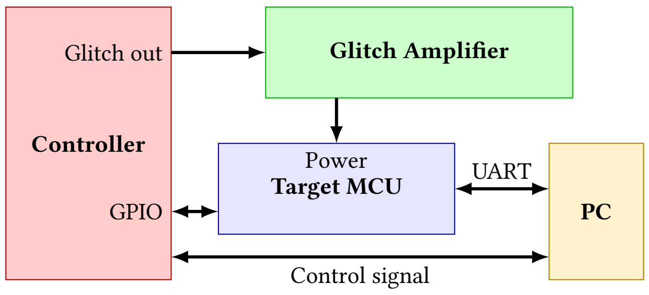

Figure 4: The physical setup for our fault injection attack.

Our constant time CSIDH is based on the C implementation in [10]. In our implementation, the isogeny operations are calculated randomly, i.e., the points are generated by *Elligator* [32]. Due to the time consumption of CSIDH on the target board, we demonstrate the attack on a toy CSIDH parameter set with a key space of size 11^2, which shortened the time required for experiments. Our private key range is $[-5, 5]$, and we use only two small prime degree isogenies: 359 and 353. All the experiments run under the above setting, and the average time consumption of executing one CSIDH is around 150 seconds. Note that previous experiment works [10, 11] focus on a key space of size 3^2; in particular, those works consider only one round of CSIDH, and so do not need to implement binary search for key recovery.

6.2 Profiling Phase

Duration and voltage. We profile the duration and voltage of the voltage glitch setup. The duration of 175ns and voltage of 1.85v give the bast faulty rate, and is used in all of our experiments. More details of the profiling can be found in the appendix-B.

Table 2: Results of injecting faults during real isogeny operations in toy CSIDH. C, F, and R respectively denote the number of correct outputs, the number of incorrect outputs, and the number of resets. Our parameters are marked with *.

Degree	Injection Time (%)	Round 1			Round 2			Round 3			Round 4			Round 5		
		C	F	R	C	F	R	C	F	R	C	F	R	C	F	R
	14	26	6	18	25	17	8	50	0	0	49	0	1	0	33	17
359	41*	29	13	8	7	13	30	16	31	3	41	5	4	26	18	6
	91	41	1	8	43	7	0	20	17	13	47	0	3	37	2	11
	31	32	0	18	37	5	8	38	2	10	41	1	8	49	0	1
353	52	0	14	36	1	5	44	1	25	24	1	20	29	20	18	12
	60*	15	19	16	33	8	9	2	47	1	4	24	22	0	3	47

Table 3: Result of injecting faults during dummy isogeny operations in Toy CSIDH.

Degree	Injection Time (%)	Round 1			Round 2			Round 3			Round 4			Round 5		
		C	F	R	C	F	R	C	F	R	C	F	R	C	F	R
	14	0	0	50	1	0	49	0	0	50	0	0	50	5	0	45
359	41*	41	0	9	38	0	12	40	0	10	40	0	10	31	0	19
	91	27	0	23	13	0	37	8	0	42	14	0	36	31	0	19
	31	43	0	7	39	0	11	50	0	0	0	0	50	26	0	24
353	52	26	0	24	47	0	3	47	0	3	11	0	39	49	0	1
	60*	16	0	34	36	0	14	25	0	25	41	0	9	45	0	5

Injection time. To determine the appropriate injection time, we used fine-grained search with 100 candidate injection time points, inducing 10 faults to measure the faulty rate of each point in the profiling phase. The results are shown in Figure 1 in Section 5.1.1. For our end-to-end attack, we randomly pick one points that have the highest faulty rate in the profiling phase. Specifically, we use 41% for degree 359 and 60% for degree 353.

Faulty rate and reset rate of the chosen fault parameter in the profiling phase. To test how reliable the profiling result will be in real attacks after the profiling, we test the three points that have the highest faulty rate in the profiling. We inject faults at the chosen injection time in repeated runs to evaluate the faulty rate and reset rate in isogeny operations. We take 50 trials for each point and test five rounds in total. The interval time between each round is more than an hour. Table 2 is the result of inducing faults during real isogeny operation, and Table 3 is the result of inducing faults during dummy isogeny operation.

The results of inducing the fault during the isogeny operations are different across time. For example, we did not get any faulty output, with injection time 14% for degree 359, in Round 3 of real isogeny operation, but we obtained 33 faulty outputs and 17 reset with the same injection time in Round 5. Therefore, we calculated the success rate and the number of required tests with the average value and deviation of faulty rate and reset rate, provided in Table 4. The injection time we used in the experiments is 41% in degree-359, which has, on average, a 42.7% faulty rate and 20.4% reset rate inducing the fault during real isogeny operations, and has, on average, a 0% faulty rate and 24% reset rate inducing the fault during dummy isogeny operations. For degree-353, the injection time we used in the experiments is 60%, which has, on average, a 71.4% faulty rate and 38% reset rate inducing the fault during real isogeny operations, and has, on average, a 0% faulty rate and 34.8% reset rate inducing the fault during dummy isogeny operations.

Table 4: Average value and deviation of Table 2 and Table 3. F and R respectively denote the fault rate and the reset rate.

Degree	Injection Time(%)	Real				Dummy			
		F		R		F		R	
		avg	dev	avg	dev	avg	dev	avg	dev
	14	0.318	0.372	0.176	0.153	0	0	0.976	0.039
359	41*	0.427	0.209	0.204	0.201	0	0	0.240	0.073
	91	0.135	0.169	0.140	0.097	0	0	0.628	0.176
	31	0.039	0.051	0.164	0.116	0	0	0.368	0.352
353	52	0.844	0.193	0.580	0.217	0	0	0.280	0.301
	60*	0.714	0.302	0.380	0.313	0	0	0.348	0.213

Note that the faulty rate and reset rate could depend on the injection setup and victim device. Comparing the faulty rate with [10] (32.8% with key value $[-1, 1]$, where all isogenies are real), our experiment has a higher faulty rate on real isogenies, which is 42.7% on average. Thus, we believe our results are reasonable.

During profiling and testing on a single isogeny, we never get any faulty output when we inject the fault in a dummy isogeny operation. However, in our end-to-end attack, we occasionally get faulty output for dummy isogeny as well. Our complexity equation still considers the faulty rate of inducing the fault during a dummy isogeny operation, thus the attacker can adopt the analysis to another fault attacks.

Choice of k for end-to-end attack. The success rate of key recovery using our attack technique is $(1-(1-p_f)^k)^{n\lceil \log_2 b \rceil}$. The lower bound of k which achieves a 50% success rate is 4. We set $k = 5$ to increase the success rate. With $k = 5$ and faulty rate $p_f = 0.427$, the estimated success rate of recovering the key is 86.2% in theory.

6.3 End-to-End Attack Result

6.3.1 Real-then-dummy CSIDH. After the profiling phase, we test the binary search fault injection attack on real-then-dummy CSIDH. We launch binary search attacks ten times in total with different key values. To test whether the isogeny operation is real or dummy, we take 5 trials for each testing isogeny operation (value of k in algorithm 3 is 5). Table 5 is the result of fault injection attack on real-then-dummy CSIDH based on binary search algorithm with key value [1,1] and [4,4]. In terms of the success rate of the attacks. Most of the attacks can at least partially recover the key. When true key is [4,4], smaller key values are recovered sometimes, this indicates some real isogenies are treated as dummy isogeny. That means in some attacks the faulty rate is lower than expected.

For true key [1,1], the average number of group actions (T in Section 5.2.1) is 62 times, including a minimal value of 38 times and a maximum value of 103 times. The number of group actions is affected by the reset rate. For example, in the ninth trial of key value [1,1], it took 66 group actions because there were fewer resets than in the tenth trial, which took 94 group actions. In Table 5, comparing the attack when the true key is [1,1] with the case when the true key is [4,4], we observed a lower average value of the number of group actions, which is 42 times. This is because key [4,4] has more real isogeny computations and real isogenies are identified more quickly on average than dummy isogeies—to determine that an isogeny is dummy, the attacker needs to run CSIDH k times, but

Table 5: Fault injection attack on real-then-dummy CSIDH results with true key $[1, 1]$ (left) and $[4, 4]$ (right).

Trial	Recovered Key	T
1	[1, 1] Correct	62
2	[1, 1] Correct	71
3	[0, 1] Partial Recovery	103
4	[1, 1] Correct	38
5	[1, 1] Correct	40
6	[0, 1] Partial Recovery	46
7	[1, 1] Correct	48
8	[1, 1] Correct	48
9	[1, 1] Correct	66
10	[1, 1] Correct	94

Trial	Recovered Key	T
1	[4, 2] Partial Recovery	49
2	[1, 4] Partial Recovery	28
3	[2, 4] Partial Recovery	17
4	[4, 2] Partial Recovery	35
5	[0, 2] Incorrect	49
6	[4, 4] Correct	78
7	[3, 3] Incorrect	51
8	[4, 4] Correct	27
9	[4, 3] Partial Recovery	44
10	[3, 3] Incorrect	45

Table 6: Key recovery for dynamic random vector CSIDH.

Key Value	Degree-359		Degree-353		T
	Faulty Rate	Recovered e_i	Faulty Rate	Recovered e_i	
(1,1)	0.120	2 (Incorrect)	0.105	1 (Correct)	1090
(2,2)	0.355	5 (Incorrect)	0.295	3 (Incorrect)	1196
(3,3)	0.445	6 (Incorrect)	0.680	5 (Incorrect)	1043
(4,4)	0.615	8 (Incorrect)	0.325	3 (Incorrect)	869
(5,5)	0.890	11 (Incorrect)	0.910	7 (Incorrect)	996

for a real isogeny the attack will finish in at most k iterations, since we stop as soon as one faulty output is observed.

Considering the reset rate $\rho = 0.209$, the estimated expected total number T of fault injections required is $69 \leq T \leq 114$, which is calculated by $\frac{n\alpha}{1-\rho}\lceil \log_2 b \rceil \leq T \leq \frac{nk}{1-\rho}\lceil \log_2 b \rceil$, as in Section 5.2.1. In our experiment, with key values [1,1] and [4,4], we observed $T = 62$ and $T = 42$, respectively. The faulty rate and reset rate in the profiling do not exactly match those in the real attacks. Still, our experimental results are close to the estimated T values.

6.3.2 Dynamic Random Vector CSIDH. To recover the secret key of dynamic random vector CSIDH, we induced the fault during the first isogeny operation of each degree repeatedly, and use $|e_j| = \left\lceil b \frac{f}{p_f(f+c)} \right\rfloor$ to guess the possible secret key value. We processed the fault injection attack on different secret key values to measure the accuracy of fault injection attack on dynamic random vector CSIDH, and evaluate the security level of this mitigation. The results of this experiment appear in Table 6. For each degree isogeny, we run 200 trials of injecting faults into a real isogeny of that degree, retrying whenever the board resets. For instance, $|e_{353}|$ of key value (1,1), the possible secret key value is $e_{353} = \left\lceil 5 \cdot \frac{0.105}{0.714} \right\rfloor = 1$, which is correct. $|e_{359}|$ of key value (5,5), the possible secret key value is $e_{359} = \left\lceil 5 \cdot \frac{0.89}{0.427} \right\rfloor = 11$, which is incorrect[2]. The success rate of the attack on dynamic random vector CSIDH is 20%, and the average time consumption is 14273 seconds. According to the high deviation of faulty rate in Table 4, it is reasonable that the success rate of the attack on dynamic random vector CSIDH is lower than the attack on real-then-dummy CSIDH. The attack method of dynamic random vector CSIDH is based on the statistic of the number of faulty output and correct output. Therefore, if the faulty rate is not stable, the success rate of the attack on dynamic random vector CSIDH is low.

[2]Obviously, $e_j = 11$ is not a reasonable guess at the key entry, since $b = 5$. Our analysis in Section 5.2.2 does account for this; as long as a sufficient number of faults are successfully injected, if the probability p that injecting a fault into a real isogeny yields an incorrect output is constant and estimated accurately, then the estimate $e_j = \left\lceil b \frac{f}{p(f+c)} \right\rfloor$ will be correct (and in particular, not larger than b) with high probability.

7 DISCUSSION

Aside from CSIDH alternatives designed to prevent side-channel and fault injection attacks, there are other proposals designed to make CSIDH faster or more secure against traditional attacks [1, 9, 15]. Our attacks will apply essentially without modification to SQALE [15] and "high security CSIDH" [9] with dummy isogenies. CTIDH [1] is more difficult to attack because isogenies of different degrees are batched and cannot be distinguished by timing information. Nevertheless, the attack we implement here applies in a limited way, reducing the key entropy of CTIDH-512 from 2^{256} to at most 2^{174} (likely significantly less). We leave implementations of our attack in these CSIDH variant settings as future work.

8 CONCLUSION

We have demonstrated the fault injection attack on real-then-dummy CSIDH based on binary search, as described in [30], for the first time. We conducted substantial tests and analysis to find optimized parameters in the context of our physical setup. Compared with prior work, our experiment achieves a higher success rate in key recovery in substantially more realistic conditions (e.g., with realistic key magnitudes). Our results demonstrate that efficacy of these fault attacks is very sensitive to the attack parameters, and so extensive profiling is crucial to fault attack success.

Beyond our attacks on the standard real-then-dummy CSIDH, we for the first time implemented dynamic random vector CSIDH [30], and evaluated its security against fault injection attacks. According to our experiment results (q.v. Section 6.3), dynamic random vector CSIDH is a more effective fault attack prevention mechanism than initially believed since imperfect fault injection success rates make it difficult to estimate the number of real isogenies. These experimental results demonstrate the significant gap between the theory of fault attacks on CSIDH and the practical aspects of launching them.

ACKNOWLEDGEMENTS

This paper was supported in part by the Commonwealth Cyber Initiative (CCI), an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org. This project is also partially supported by the National Science Foundation (NSF) under grant CCF-2153748, and by the Air Force Office of Scientific Research under award number FA9550-22-1-0548. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

REFERENCES

[1] Gustavo Banegas, Daniel J Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. 2021. CTIDH: faster constant-time CSIDH. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 4 (2021), 351–387.

[2] Gustavo Banegas, Juliane Krämer, Tanja Lange, Michael Meyer, Lorenz Panny, Krijn Reijnders, Jana Sotáková, and Monika Trimoska. 2023. Disorientation faults in CSIDH. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 310–342.

[3] Noemie Beringuier-Boher, Kamil Gomina, David Hely, Jean-Baptiste Rigaud, Vincent Beroulle, Assia Tria, Joel Damiens, Philippe Gendrier, and Philippe Candelier. 2014. Voltage Glitch Attacks on Mixed-Signal Systems. In *2014 17th*

Euromicro Conference on Digital System Design. 379–386. https://doi.org/10.1109/DSD.2014.14

[4] Daniel J Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. 2020. Faster computation of isogenies of large prime degree. *Open Book Series* 4, 1 (2020), 39–55.

[5] Ward Beullens, Shuichi Katsumata, and Federico Pintore. 2020. Calamari and Falafl: logarithmic (linkable) ring signatures from isogenies and lattices. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 464–492.

[6] Jean-François Biasse, Annamaria Iezzi, and Michael J Jacobson Jr. 2018. A note on the security of CSIDH. In *International Conference on Cryptology in India*. Springer, 153–168.

[7] Xavier Bonnetain and André Schrottenloher. 2020. Quantum security analysis of CSIDH. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*. Springer, 493–522.

[8] Robert Buhren, Hans-Niklas Jacob, Thilo Krachenfels, and Jean-Pierre Seifert. 2021. One glitch to rule them all: Fault injection attacks against amd's secure encrypted virtualization. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2875–2889.

[9] Fabio Campos, Jorge Chavez-Saab, Jesús-Javier Chi-Domínguez, Michael Meyer, Krijn Reijnders, Francisco Rodríguez-Henríquez, Peter Schwabe, and Thom Wiggers. 2023. Optimizations and Practicality of High-Security CSIDH. Cryptology ePrint Archive, Paper 2023/793. https://eprint.iacr.org/2023/793

[10] Fabio Campos, Matthias J Kannwischer, Michael Meyer, Hiroshi Onuki, and Marc Stöttinger. 2020. Trouble at the CSIDH: protecting CSIDH with dummy-operations against fault injection attacks. In *2020 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE, 57–65.

[11] Fabio Campos, Juliane Krämer, and Marcel Müller. 2021. Safe-error attacks on SIKE and CSIDH. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 104–125.

[12] Wouter Castryck and Thomas Decru. 2023. An Efficient Key Recovery Attack on SIDH. In *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part V (Lecture Notes in Computer Science, Vol. 14008)*, Carmit Hazay and Martijn Stam (Eds.). Springer, 423–447. https://doi.org/10.1007/978-3-031-30589-4_15

[13] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. 2018. CSIDH: an efficient post-quantum commutative group action. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24*. Springer, 395–427.

[14] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. 2019. Stronger and Faster Side-Channel Protections for CSIDH. In *Progress in Cryptology – LATINCRYPT 2019*, Peter Schwabe and Nicolas Thériault (Eds.). Springer International Publishing, Cham, 173–193.

[15] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. 2022. The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents. *Journal of Cryptographic Engineering* 12, 3 (2022), 349–368.

[16] Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. 2022. Optimal strategies for CSIDH. *Advances in Mathematics of Communications* 16, 2 (2022), 383–411. https://doi.org/10.3934/amc.2020116

[17] Luca De Feo, David Jao, and Jérôme Plût. 2014. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology* 8, 3 (2014), 209–247. https://doi.org/doi:10.1515/jmc-2012-0015

[18] Luca De Feo and Michael Meyer. 2020. Threshold schemes from isogeny assumptions. In *Public-Key Cryptography–PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part II 23*. Springer, 187–212.

[19] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, Philippe Orsatelli, Philippe Maurine, and Assia Tria. 2012. Injection of transient faults using electromagnetic pulses Practical results on a cryptographic system. IACR ePrint Archive.

[20] Samuel Dobson, Steven D. Galbraith, Jason LeGrow, Yan Bo Ti, and Lukas Zobernig. 2020. An adaptive attack on 2-SIDH. *International Journal of Computer Mathematics: Computer Systems Theory* 5, 4 (2020), 282–299.

[21] Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. 2013. Fault attacks on AES with faulty ciphertexts only. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 108–118.

[22] Steven D Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. 2016. On the security of supersingular isogeny cryptosystems. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*. Springer, 63–91.

[23] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer. js: A remote software-induced fault attack in javascript. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*. Springer, 300–321.

[24] Wassily Hoeffding. 1994. Probability inequalities for sums of bounded random variables. *The collected works of Wassily Hoeffding* (1994), 409–426.

[25] Aaron Hutchinson, Jason LeGrow, Brian Koziel, and Reza Azarderakhsh. 2020. Further Optimizations of CSIDH: A Systematic Approach to Efficient Strategies, Permutations, and Bound Vectors. In *Applied Cryptography and Network Security*, Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi (Eds.). Springer International Publishing, Cham, 481–501.

[26] David Jao, Jason LeGrow, Christopher Leonardi, and Luis Ruiz-Lopez. 2020. A subexponential-time, polynomial quantum space algorithm for inverting the CM group action. *Journal of Mathematical Cryptology* 14, 1 (2020), 129–138.

[27] Shuichi Katsumata, Yi-Fu Lai, Jason T. LeGrow, and Ling Qin. 2023. CSI-Otter: Isogeny-based (partially) blind signatures from the class group action with a twist. In *Annual International Cryptology Conference*. Springer, 729–761.

[28] Chong Hee Kim and Jean-Jacques Quisquater. 2007. Fault attacks for CRT based RSA: New attacks, new results, and new countermeasures. In *IFIP International Workshop on Information Security Theory and Practices*. Springer, 215–228.

[29] Jason T LeGrow. 2023. A faster method for fault attack resistance in static/ephemeral CSIDH. *Journal of Cryptographic Engineering* 13, 3 (2023).

[30] Jason T LeGrow and Aaron Hutchinson. 2021. (Short Paper) Analysis of a strong fault attack on static/ephemeral CSIDH. In *International Workshop on Security*. Springer, 216–226.

[31] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. 2023. A Direct Key Recovery Attack on SIDH. In *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V (Lecture Notes in Computer Science, Vol. 14008)*, Carmit Hazay and Martijn Stam (Eds.). Springer, 448–471. https://doi.org/10.1007/978-3-031-30589-4_16

[32] Michael Meyer, Fabio Campos, and Steffen Reith. 2019. On lions and elligators: An efficient constant-time implementation of CSIDH. In *Post-Quantum Cryptography: 10th International Conference, PQCrypto 2019, Chongqing, China, May 8–10, 2019 Revised Selected Papers 10*. Springer, 307–325.

[33] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. 2013. Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. 77–88.

[34] Koksal Mus, Yarkın Doröz, M Caner Tol, Kristi Rahman, and Berk Sunar. 2023. Jolt: Recovering tls signing keys via rowhammer faults. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1719–1736.

[35] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. 2019. (Short Paper) A faster constant-time algorithm of CSIDH keeping two points. In *Advances in Information and Computer Security: 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28–30, 2019, Proceedings 14*. Springer, 23–33.

[36] Chris Peikert. 2020. He gives C-sieves on the CSIDH. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 463–492.

[37] Damien Robert. 2023. Breaking SIDH in Polynomial Time. In *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part V (Lecture Notes in Computer Science, Vol. 14008)*, Carmit Hazay and Martijn Stam (Eds.). Springer, 472–503. https://doi.org/10.1007/978-3-031-30589-4_17

[38] Joaquin Rodriguez, Alex Baldomero, Victor Montilla, and Jordi Mujal. 2019. LLFI: Lateral Laser Fault Injection Attack. In *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 41–47.

[39] Sergei P Skorobogatov and Ross J Anderson. 2003. Optical fault induction attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002: 4th International Workshop*. Springer, 2–12.

[40] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. 2021. Post-quantum adaptor signature for privacy-preserving off-chain payments. In *International Conference on Financial Cryptography and Data Security*. Springer, 131–150.

[41] Jacques Vélu. 1971. Isogénies entre courbes elliptiques. *Comptes-Rendus de l'Académie des Sciences* 273 (1971), 238–241.

[42] Sung-Ming Yen and Marc Joye. 2000. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on computers* 49, 9 (2000), 967–970.

APPENDICES

A CSIDH SUBROUTINES

Here we present the standard real-then-dummy implementation of the class group action used in most existing implementations of CSIDH, and the random vector generation process we use in our dynamic random vector variant.

Algorithm 1 shows the Real-then-dummy constant time CSIDH. Real isogenies are computed on lines 12-14 conduct the real isogenies; once e_i becomes zero (updated on line 14), $f_i = b - |e_i|$ dummy isogenies will be computed.

Algorithm 1 Real-then-dummy constant time class group action

```
Input: $E_A : y^2 = x^3 + Ax^2 + x, A \in \mathbb{F}_p; (e_1, \ldots, e_n), e \in \{-b, \ldots, b\}$
Output: $E_{A'} = [\mathfrak{l}_1]^{e_1} \cdots [\mathfrak{l}_n]^{e_n} * E_A$
1:  Set $f_i = b - |e_i|,\ i \in 1, \ldots, n$
2:  while $e_i \neq 0$ or $f_i \neq 0$ do
3:      Set $S = \{i \mid e_i \neq 0\ or\ f_i \neq 0\}$
4:      Set $k = \prod_{i \in S} \ell_i$
5:      Generate $P_0 \in E_A[\pi - 1]$ and $P_1 \in E_A[\pi + 1]$ by Elligator
6:      $P_0 \leftarrow [(p+1)/k]P_0$ and $P_1 \leftarrow [(p+1)/k]P_1$
7:      for $i \in S$ do
8:          Set $s$ equal to the sign bit of $e_i$
9:          $K = [k/\ell_i]P_s$
10:         $P_{1-s} \leftarrow [\ell_i]P_{1-s}$
11:         if $K \neq \infty$ then
12:             if $e_i \neq 0$ then
13:                 Compute a real $\ell_i$−isogeny $\varphi$ :
14:                 $E_A \rightarrow E_{A'}$ with $\ker(\varphi) = \langle K \rangle$
15:                 $A \leftarrow A',\ P_0 \leftarrow \varphi(P_0),\ P_1 \leftarrow \varphi(P_1),\ e_i \leftarrow e_i - 1 + 2s$
16:             else
17:                 Compute dummy isogeny.
18:                 $P_s \leftarrow [\ell_i]P_s,\ f_i \leftarrow f_i - 1$
19:             end if
20:         end if
21:         $k \leftarrow k/\ell_i$
22:     end for
23: end while
24: return $E_{A'}$
```

Algorithm 2 Generate Random Vector

```
Input: $b$, the number of isogenies of each degree to be compute, $W$ the vector of secret key values, $S$ number of isogeny degrees used.
Output: $V$ the matrix which encodes the order of real and dummy isogenies.
1:  Allocate matrix $V$ of size $b \times S$ and initialize $V$ to 0
2:  for $i$ from 0 to $S - 1$ do
3:      for $j$ from 0 to $W[i] - 1$ do
4:          randomIndex ← random number between 0 and $b[i] - 1$
5:          while $V[i]$[randomIndex] = 1 do
6:              randomIndex ← random number between 0 and $b[i] - 1$
7:          end while
8:          $V[i]$[randomIndex] ← 1
9:      end for
10: end for
11: return $V$
```

B BINARY SEARCH ALGORITHM

Algorithm 3 details the binary search algorithm we use to efficiently recover the secret key in real-then-dummy CSIDH.

Algorithm 3 Binary search for recovering a static CSIDH secret key in the real-then-dummy setting

```
Input: $b = \max_i\{e_i\}$, $T$ the injection time, $c$ the isogeny index, $k$ the number of tests to determine whether an isogeny is real or dummy isogeny.
Output: Private key absolute value $|e_i|$
1:  $low \leftarrow -1$, $high \leftarrow \text{len}(b)$
2:  while $low < high$ do
3:      $mid \leftarrow \lfloor (low + high)/2 \rfloor$
4:      $i \leftarrow 0$
5:      while $i < k$ do
6:          $output \leftarrow$ inject_fault($c, T$)
7:          if $output\ \neq$ correct output then
8:              break // finish test when we get a faulty output
9:          else
10:             $i \leftarrow i + 1$
11:         end if
12:     end while
13:     if $output\ \neq$ correct output then
14:         $low \leftarrow mid$ //real isogeny, keep search right
15:     else
16:         $high \leftarrow mid$ //dummy isogeny, keep search left
17:     end if
18:     if $high - low = 1$ and $output$ = faulty then
19:         if $output$ = faulty then
20:             return $mid$
21:         else
22:             return $mid - 1$
23:         end if
24:     end if
25: end while
26: return 0
```

C PROFILING GLITCH PARAMETERS

To determine the appropriate glitch duration and voltage, we fixed the injection time and compared the faulty rate with different durations and voltages. We profile 19 durations from 140ns to 230ns, the six voltage levels from 1.8v to 2.2v with each entry taking 50 trials. Of course, the optimal duration and voltage might vary on different board. Figure 6 is the heatmap of faulty rate (i.e., $p_f = \frac{f}{f+c}$) as a function of duration and voltage, and Figure 7 is the heatmap of reset rate. We choose 175ns and 1.85v, which have the highest value in Figure 6 to ensure a high faulty rate for increasing the accuracy of the key recovery and the lowest value in Figure 7 to reduce the time consumption which is affected by reset rate.

Following the analysis of Section 5.2.1, our experiment requires 6700 (= 19×6×50+100×10) trials. Assuming the during the profiling the attacker can read the intermediate value to identify faults early before the whole CSIDH, such profiling takes 18 hours. The attacker can increase the number of tests to extract more precise variables for voltage glitches, but of course this will increase profiling time.

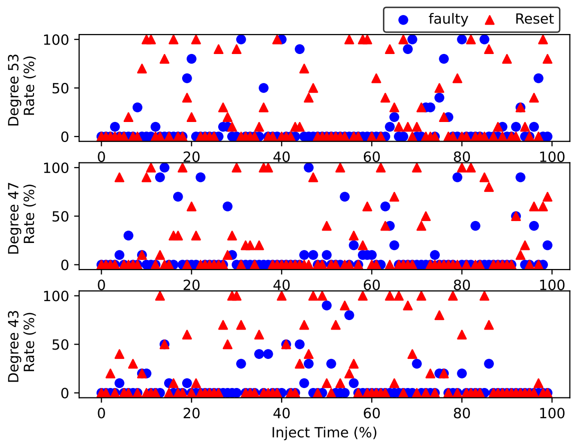

Figure 5: Faulty and reset rate when inducing the faults on real isogenies of various degrees.

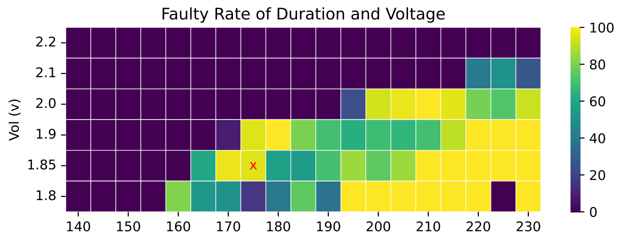

Figure 6: The rate of faulty output as a function of glitch voltage and duration. Our parameters are marked with X.

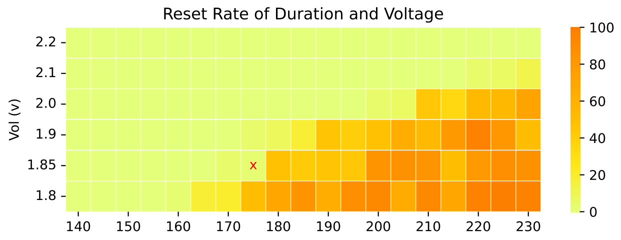

Figure 7: The probability of reset as a function of glitch voltage and duration. Our parameters are marked with X.