

Side-Channel Secure CSIDH with Projective Coordinates

Jonas Meers¹ , Anna Guinet¹ , Georg Land² , Elisabeth Krahmer¹
and Tim Güneysu¹

¹ Ruhr University Bochum, Bochum, Germany

² Intel, Feldkirchen, Germany

Abstract.

CSIDH is an isogeny-based Non-Interactive Key Exchange (NIKE) proposed at ASIACRYPT'18. In this work, we present the first masked version of CSIDH and, in fact, any isogeny-based scheme. We develop gadgets to efficiently mask all arithmetics in the underlying finite field and prove them secure in the d -probing model. In particular, we develop new gadgets for the Montgomery ladder using the fact that projective values already represent a multiplicative sharing in two variables. The technique (dubbed quotient masking) might be of independent interest. Lastly, we provide an efficient implementation based on High-Security CSIDH (Communications in Cryptology 2024). We show that the relative computational overhead of masking the latter—compared to other state-of-the-art CSIDH implementations—is similar to other masked implementations of Post-Quantum Cryptography (PQC) primitives.

Keywords: higher-order masking · csidh · pqc · d -probing model

1 Introduction

CSIDH is a NIKE based on isogenies between supersingular elliptic curves [CLM⁺18]. It can be used to instantiate many fundamental primitives such as Key Encapsulation Mechanisms (KEM) and Authenticated Key Exchange (AKE) [FHKP13], making it a versatile tool to build cryptographic protocols. In this context, CSIDH stands out by offering post-quantum security and small public keys, making it an attractive drop-in replacement for the classical Diffie-Hellman NIKE [DH76]. Indeed, this set of features remains exclusive to CSIDH (and other isogeny-based variants), as the only other post-quantum secure NIKE (called Swoosh) suffers from significantly larger public keys [GdKQ⁺24]. Therefore, as the transition to quantum-safe cryptography continues, CSIDH is a viable option for protocols like Signal's X3DH handshake [MP] and OPTLS [KW16] that currently lack post-quantum security.

The growing popularity of CSIDH has led to an increased interest in providing implementations resilient to Side-Channel Analysis (SCA) attacks. The latter exploit the physical effects of an implementation of cryptographic schemes, such as timing behavior [Koc96] or power consumption [KJJ99], that carry information about the processed data and inner operations, and therefore can leak sensitive values. To protect against timing attacks, several software constant-time implementations of CSIDH were proposed soon after its introduction [OAYT19, BBC⁺21, MCR19, CCSC⁺24]. However, the recent Hertzbleed attack [WPH⁺22] demonstrates that, despite having constant-time implementations, timing attacks can still be performed by exploiting data-dependent frequency changes

E-mail: research@meers.org (Jonas Meers), email@annagui.net (Anna Guinet), mail@georg.land (Georg Land), elisabeth.krahmer@rub.de (Elisabeth Krahmer), tim.guneysu@rub.de (Tim Güneysu)



on modern x86 processors. The attack is able to recover the full key of a constant-time implementation of the isogeny-based scheme SIKE. It also presents a threat for CSIDH, however, it was not demonstrated in practice yet.

Additionally, the authors in [MN23] recently developed a methodology to exploit partial leakage of intermediate values of CSIDH to recover the shared key, thus providing the theoretical foundation for a practical SCA attack. The latter work falls in the category of *Hidden Number Problems (HNP)*, which were pioneered by Boneh and Venkatesan [BV96] and have since then been adapted to many other settings [XHS20, Sha17, JJV07, BS01]. Although initially these problems were largely theoretical results, it has recently been shown that, in the case of classical Diffie-Hellman, HNP-type attacks can be realized in practice [HR23, MBA⁺21]. It is therefore reasonable to expect them to be also practically feasible in the case of CSIDH.

A countermeasure to both the Hertzbleed timing attack [WPH⁺22] and HNP attacks against CSIDH [MN23] is the *masking* technique [CJRR99, GP99]. Indeed, masking is a well-known countermeasure against physical attacks exploiting data-driven operations, studied for more than two decades. The core idea is to split a secret value into multiple *shares* such that the knowledge of all-but-one shares does not allow for the recovery of the original secret. The challenge lies in designing sound masked implementations and proving their security against a SCA adversary. Because of its effectiveness in mitigating SCA attacks, masking has been widely adopted in post-quantum cryptography, leading to numerous implementations for schemes like Kyber, Dilithium, BIKE and NTRU [MGTF19, BGR⁺21, CGTZ23a, CGTZ23b, DR24]. However, despite its effectiveness in other cryptographic settings, applying masking to CSIDH (and isogeny-based cryptography in general) is much more challenging as isogenies lack many of the homomorphic properties of, for instance, lattices. Furthermore, while isogeny-based cryptography is somewhat related to classical Elliptic Curve Cryptography (ECC), it has different—partly even opposite—requirements, thus preventing the reuse of existing ECC protection techniques. It therefore remains an open question how to efficiently mask isogeny-based schemes and, in particular, CSIDH.

1.1 Contributions

We present for the first time a higher-order masked implementation of CSIDH that is provably secure in the d -probing model. In fact, to the best of our knowledge this represents the first masked implementation of *any* isogeny-based scheme. Specifically, we provide a masked implementation for computing and evaluating isogenies over a finite field $\mathbb{F}_p$, leaving other subroutines that do not handle secret values unmasked. Along the way, we develop an efficient masking technique for *projective* variables that requires only half as many shares as the naive approach. Unfortunately, however, this technique is currently only used and proved secure for two shares (i.e., first-order masking with $d = 1$) due to the point multiplication in CSIDH.

Note that seemingly similar approaches already exist to secure classical ECC against side-channel attacks, like point blinding and randomizing projective coordinates [Cor99, TJ10]. However, our work clearly distinguishes from those by being provably secure and providing gadget-based implementation with high generalization potential.

SENSITIVITY ANALYSIS. As a first contribution, we identify relevant and sensitive values in the CSIDH key exchange that need protection against SCA attacks. We observe that masking requirements of CSIDH are orthogonal to classical ECC; values that needed protection in ECC do not need protection in CSIDH and vice versa. This observation implies that, unfortunately, we cannot rely on the homomorphic properties of elliptic curve points that are used to mask classical ECC routines. Therefore, we are left with the generic approach of masking all arithmetic operations over $\mathbb{F}_p$.

Table 1: Performance comparison between various isogeny-based group actions, their security level and side-channel security. Timings were obtained by averaging 100 runs on an 11th Gen Intel Core i7-1165G7 on a single core at 2.80 GHz with Turbo Boost disabled.

Variant	Instantiation	Code	Execution Time (seconds/gigacycles)	Side-Channel Security
SCALLOP [DFK ⁺ 23]	CSIDH-1024	C++	510/1423	–
SCALLOP-HD [CLP24]	CSIDH-1024	Sage	646/1814	–
PEARL-SCALLOP [ABE ⁺ 24]	CSIDH-1500	C++	1015/2848	–
KLaPoTi [PPS25]	CSIDH-512	Rust	2.81/8.162	–
PEGASIS [DEF ⁺ 25]	CSIDH-2048	Sage	27.43/76.65	–
dCSIDH [CCSC ⁺ 24]	CSIDH-2048	C	8.66/24.19	Constant Time
SQALE-OAYT [CSCJR22]	CSIDH-3072	C	3.87/10.91	Constant Time
CTIDH [BBC ⁺ 21]	CSIDH-2048	C	0.63/1.72	Constant Time
dCTIDH [CHMR25]	CSIDH-2048	C	0.54/1.49	Constant Time
This work ($d = 1$)	CSIDH-2048	C	133.32/378	Constant Time Masked

QUOTIENT MASKING. Our second contribution is a novel and more efficient way of masking *projectivized* values, called *quotient* masking, which results in a speed-up of 10% (cf. Table 2). The approach is based on the observation that projective coordinates of an affine value represent a *multiplicative* sharing with two shares. Hence, only one of the two projective coordinates needs to be further masked to achieve security against probing attacks. The other coordinate stays unmasked and can be treated as public. This further leads to an interesting optimization opportunity: depending on the arithmetic operation, leaving either the first or second coordinate unmasked can lead to significant speed-ups. We elaborate more on this aspect in Section 4.3. Currently, however, we use quotient masking only for the first-order masked implementation and resort to the naive approach for higher orders to provide side-channel resistance.

IMPLEMENTATION. Our third contribution is an optimized, masked implementation of CSIDH that supports arbitrary high masking orders d . To achieve this, we transform the deterministic and constant-time CSIDH implementation found in [CCSC⁺24] (called dCSIDH) by replacing every arithmetic operations by d -th order masked ones. We ensure their composability by following principles from the formal verification community for masking schemes [BBD⁺15, BBD⁺16, CS20], enabling us to reason about the security of our masked implementation rather than relying on empirical leakage assessments as evidence of security. Furthermore, we projectivize all sensitive values to employ our novel quotient masking technique for $d = 1$, which has the added benefit of not requiring any field inversions. We also adopt many of the low-level optimizations of [CCSC⁺24], yielding very efficient $\mathbb{F}_p$ -arithmetics. We evaluate the performance of our masked algorithm with an implementation in the C programming language for up to 5 shares ($d \leq 4$). A single first-order masked execution runs in roughly 133 seconds, which represents a factor $\times 15.3$ slowdown compared to the unmasked implementation (cf. Table 2). It is noteworthy that the relative loss in performance is much smaller compared to, e.g., a masked implementation of BIKE which has a slowdown factor $\times 300$ [DR24].

The comparison to other isogeny-based group actions is even more favorable (cf. Table 1). Indeed, the SCALLOP family of isogeny-based group actions for instance is more than $\times 5$ slower than our masked implementation while featuring no side-channel protection and a lower security level. Furthermore, the more efficient KLaPoTi and PEGASIS group actions also do not feature any side-channel protection and it is expected that incorporating side-channel protection to these group actions will lead to a significantly worse performance.

1.2 Technical Details and Related Work

CSIDH uses a large prime finite field $\mathbb{F}_p$ to carry out its computations; thus, we mainly use masking techniques dedicated to finite field arithmetics. Specifically, we use additive arithmetic masking to split a variable $x \in \mathbb{F}_p$ into shares $x_0, \dots, x_d$ such that $x = \sum_{i=0}^d x_i \bmod p$. We then replace every arithmetic operation found in CSIDH with an equivalent operation (gadget) that works on $d + 1$ shares.

MASKING PROJECTIVE VALUES. Projectivizing an affine variable x is achieved by interpreting the value as a fraction $x = A/B$. Subsequently, both numerator and denominator are considered as two separate coordinates $(A : B)$, where the original value can then always be recovered by taking their quotient. The technique is frequently used for x -only arithmetics on elliptic curves to avoid field inversions, both in classical ECC and in isogeny-based cryptography. A trivial way of masking a projective tuple $(A : B)$ is by masking the coordinates A and B separately, requiring $2(d + 1)$ shares in total. However, we observe that only one of the two coordinates needs to be masked since $(A : B)$ and $(\lambda A : \lambda B)$ are equivalent projective representations of x , knowledge of either coordinate does not determine the other coordinate or their fraction. Hence, one coordinate can be left unmasked and the other coordinate is masked using $d + 1$ shares. We name this technique *quotient masking*. For $d = 1$, one may ask whether masking one of the coordinate is necessary since the two coordinates A and B can already be viewed as multiplicative shares of the affine variable $x = A/B$. However, since our CSIDH implementation performs multiple computations on both coordinates at the same time (e.g., computing $A + B$ during the conversion to a twisted Edwards curve), it is unclear how to prove first-order security for all the (sub-)procedures of our algorithm without masking at least a coordinate. To tackle that issue, we choose to additively share the numerator A and keep B as a non-secret value, allowing us to rely on a state-of-the-art provable secure composability notion [CS20] to combine masked operations (gadgets) without compromising the security of the overall scheme. For that, we first adapt already existing gadgets for masked squaring [CMM⁺23] and masked multiplication [CGLS21] to be compatible with quotient masking and be securely composable. More details are provided in Section 4.3.

THE COST OF MASKING. As shown in Tables 1 and 2, our first-order masked CSIDH implementation induces an overhead factor of $\times 15.3$ compared to the unmasked one of [CCSC⁺24]. Although this leads to large *absolute* timings, the *relative* overhead of masking is comparable with other masked implementations of PQC primitives. Indeed, [ABC⁺23] reports an overhead factor of $\times 7.3$ for masking Dilithium and in [CGTZ23a] a masked implementation of NTRU is proposed that exhibits an overhead factor of $\times 30.4$ for decapsulation and an overhead factor of $\times 3.6$ for key generation. Additionally, the masked implementation of BIKE proposed in [DR24] has a slowdown factor of $\times 300$.¹ Furthermore, CSIDH deviates from the aforementioned protocols in that it is a *non-interactive* key exchange. Therefore, a slow execution of the protocol is tolerable in many scenarios as parties can pre-compute the shared key in an offline phase before interacting in an online phase. In contrast, signatures and KEM are inherently interactive protocols that need to perform computations *at the time of invocation*.

REUSE OF PROTECTED ECC PRIMITIVES. Research on side-channel resistant isogeny-based cryptography has only recently begun to draw the attention of the community. However, efforts on protecting operations on elliptic curves in the context of ECC can look back on a far longer history. Several approaches like *scalar splitting*, *scalar* or *basepoint blinding*, *projective coordinates randomization* or using *curve isomorphisms* had been suggested and tested extensively to mitigate SCA attacks when operating on elliptic curves [FGDM⁺10, FV12, AVL21, BCH⁺22, BR23]. Therefore, it would be desirable

¹All values are with respect to $d = 1$, or two shares. For higher values of d , the comparison becomes even more favorable for CSIDH as shown in Fig. 1.

to re-use those techniques from ECC in the CSIDH setting. Unfortunately, there is a fundamental difference between sensitive and non-sensitive values during the computations in (classical) elliptic curve and isogeny-based cryptography. In particular, the curve itself, while being secret in isogeny-based cryptography, is a public parameter in ECC, hence is never considered sensitive in pre-existing ECC-specific protection methods. This makes them unsuitable for our use-case.

PROTECTING AGAINST THE MONODROMY LEAK. As a result of masking the CSIDH algorithm, we provide a masking scheme for the Montgomery ladder at any masking order d . The latter is desirable due to the recent Monodromy leak attack by Robert [Rob24], which recovers the secret scalar m from the projective coordinates of $P = (X : Z)$ and $[m]P = (X' : Z')$.² To carry this attack, the adversary needs to observe only two variables, even if the coordinates are re-randomized as suggested in [Rob24]. Several SCA-resistant Montgomery ladder implementations have been proposed, such as scalar blinding and point randomization [SG18], or a combination of point randomization and scalar masking [BR23], for ECC implementations. However, these point randomization techniques does not prevent the aforementioned attack. Furthermore, both countermeasures were designed for ECC implementation requirements, which differ from those of CSIDH, as discussed above. Consequently, we propose an alternative SCA-resistant implementation of the Montgomery ladder that is both suitable for a SCA-resistant CSIDH implementation and secure against the Monodromy leak attack.

2 Preliminaries

2.1 Notation

We denote by $\llbracket \cdot \rrbracket$ a Boolean test that returns 1 if the enclosed statement is true and 0 otherwise. We write $[a : b]$ for the interval $\{x \in \mathbb{Z} : a \leq x \leq b\}$. Unless stated otherwise, we use lowercase letters for affine values and capital letters for projective values: $x = (X : Z)$. We keep the same formatting of letters for fractions $x = X/Z$. We use the upper-case calligraphic font for sets $\mathcal{S}$ and we denote by $|\mathcal{S}|$ their cardinality. Vectors are written with uppercase letters in bold font, e.g., $\mathbf{A} = (A_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$. We use $\lambda \mathbf{A}$ as a shorthand notation for $(\lambda A_i)_{0 \leq i \leq d}$.

2.2 Elliptic Curves and Isogenies

The following facts on elliptic curves and isogenies are mostly taken from Silverman [Sil09].

2.2.1 Elliptic Curves.

Let $E/\mathbb{F}_p$ be an elliptic curve over a finite field $\mathbb{F}_p$ with p an odd prime. We denote the point at infinity with ∞_E , or simply ∞ if the context is clear. For an extension field $\mathbb{K} \supseteq \mathbb{F}_p$, we denote the set of $\mathbb{K}$ -rational points by $E(\mathbb{K})$. An elliptic curve is called *supersingular* if $\#E(\mathbb{F}_p) = p + 1$ and *ordinary* otherwise. From now on, we only consider supersingular elliptic curves.

If $\#E(\mathbb{F}_p) = 0 \pmod{4}$ then E can be written in *Montgomery form*

$$E : y^2 = x^3 + Ax^2 + x$$

where $A^2 \neq 4 \in \mathbb{F}_p$ is called the *Montgomery coefficient*. Such curves benefit from fast x -only arithmetic and compact representation. On such curves, scalar multiplication of elliptic curve points is done via the *Montgomery ladder*, which combines a double-and-add

²For the attack, it is important that the coordinates $(X' : Z')$ are *not* normalized as $(X'/Z' : 1)$.

style approach with x -only addition and doubling formulas. These algorithms can further be improved by considering projective coordinates that keep track of the numerator and denominator separately, and only perform costly field divisions when strictly needed. In particular, the point $P = (x, y) \in \mathbb{F}_p^2$ has projective coordinates $P = (X : Y : Z) \in \mathbb{F}_p^3$ where $x = X/Z$ and $y = Y/Z$. Here $Z \neq 0$ if and only if $P \neq \infty$. In the case of x -only arithmetic, the XZ -coordinates are sufficient, hence we may simply write $P = (X : Z)$.

2.2.2 Isogenies.

An isogeny is a morphism $\varphi : E \rightarrow E'$ between elliptic curves E, E' such that $\varphi(\infty_E) = \infty_{E'}$. The degree of φ is its degree as a morphism and we call φ *separable* if $p \nmid \deg \varphi$. If $\deg \varphi = d$, we may call φ a d -*isogeny*. An isogeny can be expressed as a fraction of polynomials, and we call two elliptic curves *isogenous* if there exists an isogeny between them. An isogeny is called an isomorphism if it has an inverse (which may be defined over the algebraic closure of $\mathbb{F}_p$). In that case, the inverse is again an isogeny. One can check whether two elliptic curves are isomorphic by comparing their j -invariant, which is a simple algebraic expression in the coefficients of the curve equation.

Any isogeny $\varphi : E \rightarrow E'$ is automatically a group homomorphism from E to E' and as such its kernel is a finite subgroup of E . In the case where φ is separable, we have that $\deg \varphi = \# \ker \varphi$. Conversely, any finite subgroup $G \subset E$ corresponds to a separable isogeny $\varphi : E \rightarrow E'$ with kernel $\ker \varphi = G$ that is unique up to post-composition with an isomorphism. Since E' is uniquely determined by $\ker \varphi$ (again up to isomorphism), we may write $E' = E/G$ from now on. One can compute φ and E/G in time polynomial in the size of the kernel. In the particular case of a Montgomery curve $E_A : y^2 = x^3 + Ax^2 + x$ and subgroup $G \subset E_A$ not containing $(0, 0)$, the codomain of the isogeny $\varphi : E_A \rightarrow E_A/G$ can again be written in Montgomery form and has the Montgomery coefficient

$$B = \tau(A - 3\sigma), \quad \text{where } \tau = \prod_{P \in G \setminus \{\infty\}} x_P, \quad \sigma = \sum_{P \in G \setminus \{\infty\}} \left(x_P - \frac{1}{x_P} \right), \quad (1)$$

as shown in [Ren18, Proposition 1]. Furthermore, the isogeny can be explicitly expressed as the map

$$(x, y) \mapsto (f(x), \sqrt{\tau} y f'(x)), \quad \text{where } f(x) = x \prod_{P \in G \setminus \{\infty\}} \frac{xx_P - 1}{x - x_P}. \quad (2)$$

An isogeny from E to itself is called an *endomorphism*. For an integer n , we denote the multiplication-by- n map by $[n]$, which is an endomorphism of E . Its kernel is the n -*torsion subgroup* $E[n] = \{P \in E : [n]P = \infty_E\}$.

2.3 CSIDH

CSIDH is based on a group action on supersingular Montgomery curves over $\mathbb{F}_p$. Computing the group action essentially consists of elementary manipulations of elliptic curve points and isogeny computations via Eq. (1). In the following, we introduce CSIDH mainly from an algorithmical perspective. The mathematical details can be found in [CLM⁺18].

Let $p = c \prod_{i=1}^n \ell_i - 1$ be a prime such that $p + 1 \equiv 0 \pmod{4}$, where $c \in \mathbb{N}$ is a small cofactor and the ℓ_i are small odd primes. The starting point of the group action is the supersingular curve $E_0 : y^2 = x^3 + x$. It has $p + 1$ many points over $\mathbb{F}_p$, implying the existence of an $\mathbb{F}_p$ -rational subgroup of prime order ℓ_i for each i . We call these groups the *positive* subgroups. Additionally, for each i , there exists a second subgroup of order ℓ_i whose points have x -coordinates in $\mathbb{F}_p$, but y -coordinates in $\mathbb{F}_{p^2} \setminus \mathbb{F}_p$. We call these groups the *negative* subgroups. Subsequently, any positive/negative subgroup is uniquely

Algorithm 1: CSIDH

Input: Montgomery coefficient $A \in \mathbb{F}_p$ and secret key $(\mathbf{sk}_1, \dots, \mathbf{sk}_n) \in [-D : D]^n$
Output: Montgomery coefficient $B \in \mathbb{F}_p$

```

1 Compute the curve  $E_A : y^2 = x^3 + Ax^2 + x$  over  $\mathbb{F}_p$ 
2 while  $\exists j$  such that  $\mathbf{sk}_j \neq 0$  do
3   Sample a random  $x \in \mathbb{F}_p$  and lift it to a point  $P = (x : y) \in E_A(\mathbb{F}_{p^2})$ 
4   if  $\llbracket y \in \mathbb{F}_p \rrbracket$  xor  $\llbracket \text{sign}(\mathbf{sk}_j) = 1 \rrbracket$  then                                Mismatch between sign
5     | goto Line 3                                                         and subgroup
6   end
7    $P \leftarrow [(p+1)/\ell_j]P$ 
8   if  $P = \infty$  then                                                          $P$  does not have order  $\ell_j$ 
9     | goto Line 3
10  end
    //  $P$  now generates the subgroup  $G(j, \text{sign}(\mathbf{sk}_j))$ 
11  Compute isogeny  $\varphi : E_A \rightarrow E_B$  with  $\ker \varphi = \langle P \rangle$  via Eq. (1)
12   $E_A \leftarrow E_B$ 
13   $\mathbf{sk}_j \leftarrow \mathbf{sk}_j - \text{sign}(\mathbf{sk}_j)$ 
14 end
15 return  $A$ 

```

determined by a tuple $(i, b) \in [1 : n] \times \{0, 1\}$, where the first component determines the order ℓ_i of the subgroup and the second component determines the sign $(-1)^b$. Lastly, since the order of $G = G(i, b)$ is prime, any point $P \in G$ is a generator of the subgroup.

Each subgroup $G = G(i, b)$ now defines an ℓ_i -isogeny $\varphi : E_0 \rightarrow E_0/G$. We say that φ points in the *positive* or *negative* direction depending on b . The codomain E_0/G is again a supersingular elliptic curve that can be written in Montgomery form $E_A : y^2 = x^3 + Ax^2 + x$. Since isogenies preserve the order of elliptic curves, E_A also has $p+1$ many $\mathbb{F}_p$ -rational points. Thus, we can again sample a tuple (i', b') that defines an $\ell_{i'}$ -isogeny φ' with domain E_A and codomain E_B . This process can now be repeated indefinitely.

Remark 1. Since Eq. (1) only depends on the x -coordinates of the points contained in G , there is no difference between positive and negative subgroups, computationally speaking.

Most notably, computing isogenies in CSIDH is commutative: it does not matter in which order the tuples (i, b) and (i', b') are processed, the resulting codomain E_B will remain identical [CLM⁺18, Theorem 7]. This, of course, extends to arbitrarily many tuples, making a key exchange similar to Diffie-Hellman possible. Concretely, secret keys take the form $\mathbf{sk} \in [-D : D]^n$ for some small $D \in \mathbb{N}$, and each entry $\mathbf{sk}_i$ represents $|\mathbf{sk}_i|$ -many ℓ_i -isogenies in either the positive or negative direction depending on $\text{sign}(\mathbf{sk}_i)$. To compute the public key $\mathbf{pk}$ corresponding to $\mathbf{sk}$, one starts at E_0 , chooses a non-zero entry $\mathbf{sk}_j$ of $\mathbf{sk}$, and depending on its sign, computes an ℓ_j -isogeny in either the positive or negative direction. The codomain of this isogeny now replaces E_0 , and $\mathbf{sk}_j$ is decreased (or increased) by 1. This process is repeated until no non-zero entries in $\mathbf{sk}$ are left. Lastly, the public key $\mathbf{pk}$ is set to the unique Montgomery coefficient of the final curve. These steps are summarized in Alg. 1, where we highlight that Lines 3 through 10 are a (naive) way to sample the correct subgroup $G(j, \text{sign}(\mathbf{sk}_j))$.

To compute a shared key, Alice and Bob both choose a secret key $\mathbf{sk}_A, \mathbf{sk}_B \in [-D : D]^n$. They then each compute their public key $\mathbf{pk}_A = \text{CSIDH}(0, \mathbf{sk}_A)$ and $\mathbf{pk}_B = \text{CSIDH}(0, \mathbf{sk}_B)$, which are both elements in $\mathbb{F}_p$. The public keys are exchanged and the shared secret is then $K = \text{CSIDH}(\mathbf{pk}_B, \mathbf{sk}_A) = \text{CSIDH}(\mathbf{pk}_A, \mathbf{sk}_B)$.

2.4 Masking

To protect against power analysis attacks, presented by the seminal work of Kocher et al. [KJJ99], masking was proposed and remains the most widely adopted technique among the many presented countermeasures. In this work, we follow the principles of *private circuits* [ISW03] to transform an unprotected implementation to a masked one. In brief, it requires to encode the secret value into shares, mask the inner operations, and finally, decode the shares to retrieve the output.

Encoding and Decoding. Let $d \in \mathbb{N}$. A secret x is split into $d + 1$ shares x_i , where $i = 0, \dots, d$. The sharing is defined by $x = x_0 \circ x_1 \circ \dots \circ x_d$, where $\circ$ is a field operation in a prime finite field $\mathbb{F}_p$. For $p = 2$, the sharing scheme is referred to as *Boolean masking*, which was independently proposed in [CJRR99, GP99]. For p being a prime, $p > 2$, we called it *arithmetic masking*, following its first introduction in [Mes01]. The encoding is called additive (resp. multiplicative) masking when the operation $\circ$ is the addition (resp. multiplication). It ensures that the value of x can not be recovered with the knowledge of any proper subset of shares $\{x_i \mid i = 0, \dots, d\}$. We call *encoding* the generation of the shares from a secret variable and *decoding* the reconstitution of the original secret from the shares. Both procedures are assumed to not be part of the masked algorithms.

Gadget. In order to preserve the sharing in an algorithm, all operations initially carried on secret variables x in $\mathbb{F}_p$ are transformed to operate on a set (or subset) of their shares and they output shared results. Examples of operations are the addition, the multiplication, the inverse, etc. One popular method to adapt the operations so that they manipulate shared values, is to replace them by *gadgets* [ISW03]. More specifically, in this context, gadgets are operations in $\mathbb{F}_p$ that manipulate the $d + 1$ shares of the variables (i.e., *input shares*), with the optional addition of random values, to output $d + 1$ values (i.e., *output shares*) such that it preserves the operation carried on the unshared values. The optional use of additional random values $r \in \mathbb{F}_p$ provides further protection against recovering some shares of a secret: If we add a uniform and independent random value r to an intermediate computation depending on secret shares, which is not necessarily uniformly distributed, we say that we *refresh* it to mean that the sum behaves like a uniform random variable and thus can be approximated by the *refreshing* variable r .

More formally, let g be an operation that manipulates two secret variables (x, y) such that $g : (\mathbb{F}_p)^2 \rightarrow \mathbb{F}_p : (x, y) \mapsto g(x, y)$. This operation is replaced by a gadget G in the masking scheme with $G : (\mathbb{F}_p^{d+1})^2 \times (\mathbb{F}_p)^s \rightarrow \mathbb{F}_p^{d+1}$ that manipulates up to $d + 1$ shares of the secret variables, with s random refreshing values ($s \in \mathbb{N}$), to return $d + 1$ output shares, such that $\bigcirc_{i=0}^d G(x_0, \dots, x_d, y_0, \dots, y_d, r_0, \dots, r_{s-1}) = g(x, y)$. The operation $\bigcirc$ depends on the chosen encoding (additive or multiplicative). The function g and the associated gadget G can be generalized to manipulate any number of secret variables.

2.5 Security Model and Properties

The SCA capabilities of an adversary are formally defined by adversary models, which reflect the actual abilities of an attacker with various degrees of abstraction. Those adversary models are conventionally used since they allow us to reason about security and draft security proofs for masked algorithms, which makes it possible to verify their soundness without relying on empirical and setup-dependent leakage assessment. The highest level of abstraction is the algorithmic level, or the d -probing model, as proposed by Ishai et al. [ISW03], which received the most attention since it does not depend on the specificities of the underlying system on which the algorithm is implemented. It is not only simple to reason about, but also it was proven that security in the d -probing model implies security in the more realistic noisy leakage model [DDF14, PGMP19].

Definition 1 (*d*-probing model [ISW03]). An adversary can observe up to d signals carrying intermediate values during the processing of data for a given implementation. The observed signals are said to be *probed*, and the intermediate values carried by those probes are assumed to be observable and to leak independently of each other.

In this paper, we focus on proving a sound masking scheme in the *d*-probing model, where it is sufficient to reason about the algebraic expressions of the intermediate values of the gadgets to ensure security. However, since the input shares of a gadget depends on the output shares of another gadget, it creates a chain of dependencies between variables named *probe propagation*. It means that the sequential composition of gadgets being individually secure against d probes, may be not itself secure against d probes [CPRR14]. From this observation, designing a whole secure masking scheme from gadgets in the *d*-probing model to prevent such effects is a non-trivial task, specially for higher-order masking ($d > 1$ shares), as well as verifying its security. A solution is to develop stronger security notions for gadgets to ensure their secure *composition* within a masking scheme: Those notions are referred to as *composability notions* and were firstly proposed by Barthe et al. in [BBD⁺15] and [BBD⁺16]. For a given gadget, they differentiate between probes located on intermediate operations of a gadget, or *internal probes*, and on output shares, also called *output probes* for short.

Definition 2 (*d*-Non-Interference (NI) [BBD⁺15, BBD⁺16]). Let t_1 and t_2 be any numbers of probes such that $t_1 + t_2 \leq d$. A gadget is *d*-NI iff for any set of t_1 internal probes and every set of t_2 output probes, an adversary can learn at most $t_1 + t_2$ shares of each secret variable.

Definition 3 (*d*-Strong Non-Interference (SNI) [BBD⁺16]). Let t_1 and t_2 be any numbers of probes such that $t_1 + t_2 \leq d$. A gadget is *d*-SNI iff for any set of t_1 internal probes and every set of t_2 output probes, an adversary can learn at most t_1 shares of each secret variable.

The (Strong) Non-Interference ((S)NI) notions aim at limiting the amount of information an adversary can learn about secret shares in gadgets. The stronger SNI notion was proposed in addition to NI to prevent probe propagation when a probe is located at the output of a gadget. It is limited to single-output gadget, but it was later extended to multiple-output one in [CS20]. To learn t_2 or no share of a secret at the output of the gadget, as required respectively by NI and SNI, we need to refresh some inner computations to be secure against probing since the implicit assumption is that the refreshed computation behaves like an independent and uniform random variable and thus it does not reveal any information about the underlying refreshed share(s). The composition of gadgets satisfying the SNI security notion is as well SNI.

A drawback of the SNI security notion is that the trivial implementation of linear operations are not SNI, unless the output shares are refreshed, which is not only costly but it also does not simplify the task of designing masking algorithm. To tackle this problem, another security approach was proposed to limit operations carried on the shares of a secret depending on the value of their index, or *share domain*, called Probe-Isolating Non-Interference (PINI) [CS20]. The composition of PINI gadgets is also PINI. It simplifies the design of masking schemes by reasoning about the indices i of shares x_i with $i = 0, \dots, d$, known to an adversary. The implicit assumption is that refreshed variables do not have an associated index as previously explained.

Definition 4 (*d*-PINI [CS20]). Let $\mathcal{I}$ be a set of at most t_1 internal probes. Let $\mathcal{O}$ be a set of output probes. Let $\mathcal{A}$ be the set of indices of the output shares in $\mathcal{O}$ of size t_2 , such that $t_1 + t_2 \leq d$. A gadget is *d*-PINI iff, for any internal probe set $\mathcal{I}$ and output probe set $\mathcal{O}$, there exists a set $\mathcal{B}$ of share indices, with $|\mathcal{B}| \leq t_1$, such that the joint probability distribution of the values of the probes in $\mathcal{I} \cup \mathcal{O}$ is the same as if the values were computed using only input shares whose indices are in $\mathcal{A} \cup \mathcal{B}$ and refreshing variables.

The above definition implies that $|\mathcal{B}| \leq |\mathcal{I}| \leq t_1$ and $|\mathcal{O}| = |\mathcal{A}| = t_2$ since every output probe is associated to a unique share index i to preserve the encoding (cf. Section 2.4). Consequently, in a d -PINI-secure gadget, every internal probe should be associated to at most one index. As a result, before performing an operation on shares with different indices, e.g., $x_i x_j$ with $i \neq j$, the share with index j appearing in the expression of the output probe of index i should be refreshed to behave like a random variable and then its index is discarded to build the set $\mathcal{B}$.

In this paper, we ensure composability by using PINI gadgets in our higher-order masked CSIDH implementation.

3 A Masking-Friendly Version of CSIDH

In this section, we introduce a variant of CSIDH that lends itself well to be masked, mainly because we projectivize all sensitive values, allowing the use of quotient masking (cf. Section 4.3). This also comes with the added benefit of not requiring any field inversions. In Section 3.1, we briefly introduce High-Security CSIDH [CCSC⁺24] which forms the basis of our implementation as it provides highly optimized C code. We then introduce additional well-known optimization techniques in Section 3.2. Lastly, we provide pseudocode for the *unmasked* version of our CSIDH implementation (dubbed ProjectiveCSIDH) that includes all optimizations, in Alg. 2 to 6.

3.1 High Security CSIDH

In [CCSC⁺24], a thorough security analysis of CSIDH was conducted. Based on their findings and conservative security estimates, the authors determined different parameter sets and subsequently tried to optimize CSIDH on an algorithmical and implementation level. Furthermore, they took SCA attacks into consideration, mostly focusing on a constant-time implementation. Since our implementation is based on [CCSC⁺24], we now present relevant aspects of their design choices that we adopt for our implementation.

3.1.1 Elligator Technique.

A major issue, both in terms of performance and side-channel resistance, is the computation of the correct subgroup $G(i, b)$ in Alg. 1 (Lines 3 to 10) due to the use of rejection sampling. To overcome this issue, one can sample two *full-order* torsion points T_0 and T_1 , which have the property that $[(p+1)/\ell_i]T_b$ generates $G(i, b)$. That is, the point T_b is suitable for all positive (resp. negative) subgroups *simultaneously*. Furthermore, these full-order torsion points can be sampled via the deterministic Elligator map $(A, s) \mapsto (T_0, T_1)$, which takes a Montgomery coefficient $A \in \mathbb{F}_p$ and a seed $s \in \mathbb{F}_p$ as input [CCC⁺19]. The seed s is then included in the public key, effectively making the computation of $G(i, b)$ deterministic. Additionally, a static seed s_0 is used for the public starting curve E_0 as part of the public parameters. The approach however requires updating T_0 and T_1 at each step. Concretely, after having computed the isogeny $\varphi : E \rightarrow E/G(i, b)$, the points need to be set to $T_0 \leftarrow \varphi(T_0)$ and $T_1 \leftarrow \varphi(T_1)$ via Eq. (2) to ensure correctness of the key exchange.

3.1.2 SQALE Approach.

First analyzed in [CSCJR22], the SQALE approach consists of choosing secret keys from the set $\{-1, 1\}^n$, which in combination with the Elligator technique yields a constant time variant of CSIDH. Indeed, this specific choice implies that exactly one ℓ_i -isogeny is computed for each $i \in [1 : n]$ with the only variable being the sign. Since, algorithmically speaking, there is no difference between computing an isogeny in the positive or negative direction (cf. Remark 1), one easily gets a deterministic algorithm. Subsequently, the

number n of secret key bits needs to be increased to achieve the same level of security. This also implies a larger characteristic p as the product $\prod_{i=1}^n \ell_i$ increases as well.

3.1.3 Finite Field Arithmetics and the Shape of p .

As mentioned above, p is a prime such that $p + 1 = c \prod_{i=1}^n \ell_i$. In High-Security CSIDH, the cofactor c is chosen to be a power of 2^{64} to allow for *Montgomery reduction*, which is a standard tool to improve the performance of modular reduction in finite fields [Mon85]. Furthermore, the authors of [CCSC⁺24] extended the MULX instruction of [CLM⁺18] to larger prime sizes and combined it with the Karatsuba multiplication [KO63].

3.2 Additional Optimizations

In addition to the optimizations already included in [CCSC⁺24], we implemented further well-known improvements that lead to significant speed-ups in practice.

3.2.1 Montgomery Ladder without Checks.

Internally, the Montgomery ladder calls the two subroutines `xADD` and `xDBL` for adding and doubling elliptic curve points, respectively. In a typical implementation, these subroutines check separately whether one of the inputs is equal to ∞ or whether one of the inputs doubles to ∞ . Although these checks are simple algebraic expressions, their computation becomes quite expensive in a masked implementation, especially since the Montgomery ladder is called three times per execution of `SingleStep`, i.e., per secret bit b (cf. Alg. 3). However, the aforementioned cases do not occur when calling the Montgomery ladder exclusively on inputs that come from the Elligator map as the points T_0 and T_1 have full odd order. Therefore, we omit these checks in our masked implementation.

3.2.2 Meyer-Reith Approach.

Projective coordinates for elliptic curve points are a common optimization to avoid costly field inversions when computing scalar multiplications. In the isogeny setting, however, field inversions are also necessary when computing the codomain and image points under an isogeny via the formulae in Eqs. (1) and (2). In [MR18], the authors suggest using *twisted Edwards curves* together with projectivized Montgomery coefficients to avoid field inversions entirely. In the following, let $G(i, b)$ be a subgroup of E_A and $\varphi : E_A \rightarrow E_A/G(i, b)$ be the corresponding isogeny. We now explain how to efficiently evaluate φ at some point $Q = (X : Z)$ and how to compute its codomain.

We first use an observation by Costello and Hisil [CH17] to rewrite Eq. (2) in projective coordinates. For some $P \in G(i, b)$ and $j \in [1 : (\ell_i - 1)/2]$, we write $[j]P = (X_j : Z_j)$. Then, the evaluation $\varphi(Q) = (X' : Z')$ satisfies

$$X' = X \cdot \left(\prod_{j=1}^{(\ell_i-1)/2} [(X - Z)(X_j + Z_j) + (X + Z)(X_j - Z_j)] \right)^2, \text{ and} \quad (3)$$

$$Z' = Z \cdot \left(\prod_{j=1}^{(\ell_i-1)/2} [(X - Z)(X_j + Z_j) - (X + Z)(X_j - Z_j)] \right)^2, \quad (4)$$

allowing for an efficient evaluation of the isogeny without field inversions.

To efficiently compute the codomain, the authors in [MR18] first projectivize the Montgomery coefficient A as $(A_0 : A_1)$ where $A = A_0/A_1$. They then make use of *twisted Edwards curves* [BBJ⁺08], which are curves of the form

$$E_{a,d} : ax^2 + y^2 = 1 + dx^2y^2$$

Algorithm 2: ProjectiveCSIDH

Input: Montgomery coefficient $A \in \mathbb{F}_p$, Elligator seed $s \in \mathbb{F}_p$, small odd primes $\ell_1, \dots, \ell_n \in \mathbb{N}$ and secret key $\mathbf{sk} = (\mathbf{sk}_1, \dots, \mathbf{sk}_n) \in \{0, 1\}^n$

Output: Montgomery coefficient $B \in \mathbb{F}_p$

```

1  $T_0, T_1 \leftarrow \text{Elligator}(A, s)$ 
2  $k = \prod_{i=1}^n \ell_i$  Ignore the cofactor  $c$ 
3  $B = (B_0 : B_1) \leftarrow (A : 1)$ 
4 for  $i = 1, \dots, n$  do
5    $k \leftarrow k / \ell_i$ 
6    $(B, T_0, T_1) \leftarrow \text{SingleStep}(B, \ell_i, T_0, T_1, k, \mathbf{sk}_i)$ 
7 end
8 return  $B_0/B_1$ 

```

with $a, d \in \mathbb{F}_p$ under the condition that $ad \neq 0, d \neq 1$ and $a \neq d$. The Montgomery curve E_A with coefficient $(A_0 : A_1)$ is isomorphic to the twisted Edwards curve $E_{a,d}$ with $(a, d) = (A_0 + 2A_1, A_0 - 2A_1)$ [MOT20]. The main benefit is now that the codomain $E_A/G(i, b)$ is isomorphic to the Edwards curve $E_{a',d'}$ with

$$a' = a^{\ell_i} \cdot \pi_Z^8, \quad d' = d^{\ell_i} \cdot \pi_Y^8,$$

$$\pi_Y = \prod_{j=1}^{(\ell_i-1)/2} (X_j - Z_j), \quad \pi_Z = \prod_{j=1}^{(\ell_i-1)/2} (X_j + Z_j).$$

Computing the codomain in twisted Edwards form thus does not require any field inversions. Additionally, it is possible to share common sub-expressions with Eqs. (3) and (4). In a last step, the Montgomery coefficient of the codomain can be recovered via $A' = (A'_0 : A'_1) = (2(a' + d') : a' - d')$.

Remark 2. There exist many additional techniques that can be used to speed up CSIDH like radical isogenies [CDV20] or $\sqrt{\ell}$ u [BDFLS20]. However, we chose not to include them in our masked implementation as these subroutines are very complex and thus costly to mask, hence nullifying their potential speed-up.

4 Masking CSIDH

In this section, we first investigate which variables should be considered secret for masking purposes. Then, we briefly present the well-known additive masking technique to encode all the identified sensitive variables (including both coordinates of projectivized variables) for higher masking orders as the straightforward and naive approach. Finally, we present the novel quotient masking described for any order d and we explain, unfortunately, why we only use it for the first order ($d = 1$) of our masked CSIDH algorithm.

4.1 Sensitivity Analysis

In this section, we analyze the sensitivity of all variables and operations in the CSIDH key exchange. First, we note that the key generation and shared key derivation are almost identical. To be more precise, the key generation first samples a random secret key $\mathbf{sk} \in \{0, 1\}^n$, executes $\mathbf{pk}_A = \text{ProjectiveCSIDH}(0, s_0, \mathbf{sk})$ —where $(0, s_0)$ are the Montgomery coefficient and Elligator seed of the public starting curve E_0 —, and then computes an Elligator seed s_A for the resulting public key curve E_A . Deriving the shared secret then amounts to a single execution of ProjectiveCSIDH without the need to sample an Elligator

Algorithm 3: SingleStep

Input: Montgomery coefficient $A = (A_0 : A_1) \in \mathbb{F}_p^2$, odd integer $\ell \in \mathbb{N}$, points $T_0 = (X_0 : Z_0), T_1 = (X_1 : Z_1) \in \mathbb{F}_p^2$, scalar $k \in \mathbb{F}_p$ and secret key bit $b \in \{0, 1\}$

Output: Montgomery coefficient $B = (B_0 : B_1) \in \mathbb{F}_p^2$, points $T'_0, T'_1 \in \mathbb{F}_p^2$

```

1   $P \leftarrow T_b$  Choice of P
2   $P \leftarrow \text{ML}(k, P, A)$ 
3   $(a, d) \leftarrow (A_0 + 2A_1, A_0 - 2A_1)$  Convert to twisted Edwards curve
4   $(\pi_Y, \pi_Z) \leftarrow (X_P - Z_P, X_P + Z_P)$ 
5   $X'_0 \leftarrow (X_0 - Z_0)(X_P + Z_P) + (X_0 + Z_0)(X_P - Z_P)$  Trick by Costello
6   $Z'_0 \leftarrow (X_0 - Z_0)(X_P + Z_P) - (X_0 + Z_0)(X_P - Z_P)$  and Hisil
7   $X'_1 \leftarrow (X_1 - Z_1)(X_P + Z_P) + (X_1 + Z_1)(X_P - Z_P)$ 
8   $Z'_1 \leftarrow (X_1 - Z_1)(X_P + Z_P) - (X_1 + Z_1)(X_P - Z_P)$ 
9   $Q, D \leftarrow P$ 
10  $P \leftarrow \text{xDBL}(A, P)$ 
11 for  $k = 2, \dots, (\ell - 1)/2$  do Meyer-Reith: Codomain formula for Edwards
12    $\pi_Y \leftarrow \pi_Y \cdot (X_P - Z_P)$  + isogeny evaluation for Montgomery
13    $\pi_Z \leftarrow \pi_Z \cdot (X_P + Z_P)$ 
14    $X'_0 \leftarrow X'_0 \cdot ((X_0 - Z_0)(X_P + Z_P) + (X_0 + Z_0)(X_P - Z_P))$ 
15    $Z'_0 \leftarrow Z'_0 \cdot ((X_0 - Z_0)(X_P + Z_P) - (X_0 + Z_0)(X_P - Z_P))$ 
16    $X'_1 \leftarrow X'_1 \cdot ((X_1 - Z_1)(X_P + Z_P) + (X_1 + Z_1)(X_P - Z_P))$ 
17    $Z'_1 \leftarrow Z'_1 \cdot ((X_1 - Z_1)(X_P + Z_P) - (X_1 + Z_1)(X_P - Z_P))$ 
18    $D' \leftarrow P$ 
19    $P \leftarrow \text{xADD}(P, Q, D)$ 
20    $D \leftarrow D'$ 
21 end
22  $(a, d) \leftarrow (a^\ell \cdot \pi_Z^8, d^\ell \cdot \pi_Y^8)$ 
23  $B \leftarrow (2(a + d) : a - d)$  Convert Edwards to Montgomery
24  $T'_0 \leftarrow ((X'_0)^2 \cdot X_0 : (Z'_0)^2 \cdot Z_0)$ 
25  $T'_1 \leftarrow ((X'_1)^2 \cdot X_1 : (Z'_1)^2 \cdot Z_1)$ 
26  $T'_0 \leftarrow \text{ML}(\ell, T'_0, B)$  Get rid of the  $\ell$  torsion
27  $T'_1 \leftarrow \text{ML}(\ell, T'_1, B)$ 
28 return  $B, T'_0, T'_1$ 

```

seed at the end. Since the Elligator seed can be computed from the *public* key curve E_A alone, there is no need to mask the Elligator subroutine. Therefore, the only subroutine that requires masking is indeed ProjectiveCSIDH.

4.1.1 ProjectiveCSIDH and SingleStep.

Naturally, the secret key itself is a sensitive value and needs to be masked. In particular, the secret key bit b is only used a single time and influences the choice of the point P in the first line of **SingleStep**. Afterwards, all computations only depend on the choice of P . Therefore, it is necessary to mask the coordinates of P and the coordinates of Q, D (which depend on P). Furthermore, this also encompasses the values $X_\beta, Z_\beta, X'_\beta, Z'_\beta$ for $\beta \in \{0, 1\}$ as they are computed from the coordinates of P .

Lastly, in **SingleStep**, we also need to mask the Montgomery coefficients A, B and the Edwards parameters a, d . Indeed, for a fixed input A and ℓ there are only two possible output coefficients B depending on the secret key bit b . Therefore, leaking the final output coefficient B immediately leaks the corresponding secret key bit. The same argument applies to the input coefficient A as it is equal to the output B of the previous **SingleStep**

Algorithm 4: Montgomery Ladder ML

Input: Scalar $\ell = (\ell_{n-1}, \dots, \ell_0) \in \{0, 1\}^n$ with $\ell_{n-1} = 1$, point $P = (X_P : Z_P) \in \mathbb{F}_p^2$ and Montgomery coefficient $A = (A_0 : A_1) \in \mathbb{F}_p^2$
Output: Point $Q = (X_Q : Z_Q) \in \mathbb{F}_p^2$ with $Q = [\ell]P$

```

1  $(Q_0, Q_1) \leftarrow (P, \text{xDBL}(P, A))$ 
2 for  $i = n - 2, \dots, 0$  do
3   if  $\ell_i = 0$  then
4      $(Q_0, Q_1) \leftarrow (\text{xDBL}(Q_0, A), \text{xADD}(Q_0, Q_1, P))$ 
5   else
6      $(Q_0, Q_1) \leftarrow (\text{xADD}(Q_0, Q_1, P), \text{xDBL}(Q_1, A))$ 
7   end
8 end
9 return  $Q_0$ 

```

Algorithm 5: xADD

Input: Points $P = (X_P : Z_P)$, $Q = (X_Q : Z_Q)$ and $P - Q = (X_{P-Q} : Z_{P-Q}) \in \mathbb{F}_p^2$
Output: Point $R = (X_R : Z_R) \in \mathbb{F}_p^2$ with $R = P + Q$

```

1  $X_R \leftarrow Z_{P-Q} \cdot ((X_P - Z_P)(X_Q + Z_Q) + (X_P + Z_P)(X_Q - Z_Q))^2$ 
2  $Z_R \leftarrow X_{P-Q} \cdot ((X_P - Z_P)(X_Q + Z_Q) - (X_P + Z_P)(X_Q - Z_Q))^2$ 
3 return  $(X_R : Z_R)$ 

```

execution (cf Line 2 in the for loop of ProjectiveCSIDH).

In brief, for ProjectiveCSIDH and SingleStep, respectively, we highlight in blue the public variables that should *not* be masked in Alg. 2 and Alg. 3. Note that for ProjectiveCSIDH, only the first instance of the coefficient B is public; it is masked thereafter in SingleStep.

4.1.2 Montgomery Ladder, xADD and xDBL.

These subroutines are called almost exclusively on inputs that have already been identified as sensitive from the previous paragraph. This includes the Montgomery coefficient A and the point P . The only exception is the scalar k , which does not depend on any secret. This is in contrast to classical ECC where the scalar represents the secret key and the curve parameter A is public.

For the Montgomery Ladder, we highlight in blue the public variables in Alg. 4 to not be masked. For xADD in Alg. 5 and xDBL Alg. 6, all variables are considered sensitive.

4.2 Additive Masking on Both Coordinates

We apply the well-known additive encoding to the sensitive variables identified previously. In a first straightforward and naive approach for a projectivized secret variable $x \in \mathbb{F}_p$ with $x = A/B$, we encode both coordinates $A \in \mathbb{F}_p$ and $B \in \mathbb{F}_p \setminus \{0\}$ to have at the d -th order

Algorithm 6: xDBL

Input: Point $P = (X_P : Z_P) \in \mathbb{F}_p^2$ and Montgomery coefficient $A = (A_0 : A_1) \in \mathbb{F}_p^2$
Output: Point $R = (X_R : Z_R) \in \mathbb{F}_p^2$ with $R = [2]P$

```

1  $X_R \leftarrow A_1(X_P^2 - Z_P^2)$ 
2  $Z_R \leftarrow (4X_P Z_P)(A_0 X_P Z_P + A_1(X_P^2 + Z_P^2))$ 
3 return  $(X_R : Z_R)$ 

```

$A = \sum_{i=0}^d A_i$ and $B = \sum_{i=0}^d B_i$. We remind that both shared coordinates are processed according to Alg. 2 and the fraction A/B is not computed explicitly; the latter is only a convenient representation to write the projectivized variables.

Concerning the choice of additive sharing, CSIDH performs numerous non-linear operations, therefore one may question whether a multiplicative sharing would not be more suited than an additive one. However, after conducting a preliminary cost analysis, performing an addition, or any linear operation, with a multiplicative sharing requires to (1) convert the multiplicative shares to additive ones, (2) carry out the linear operation, then (3) convert back to multiplicative sharing to perform following non-linear operations. The latter conversion is particularly costly in $\mathbb{F}_p$ since it requires d field inversions for $d+1$ shares (according to the A2M conversion algorithm in [KLRBG22]). It would nullify the optimizations made to have a masking-friendly version of CSIDH without field inversion to increase the speed-up of the overall implementation (cf. Section 3). Thus, despite the high number of multiplications in CSIDH, it is therefore counter-intuitively cheaper to opt to additive masking to protect it against SCA attacks.

For the masked operations on the shares, we use gadgets available in the literature for additive sharing that satisfy the PINI security notion to ensure secure composition for higher masking orders (cf. Definition 4).

Addition Gadget. Every addition is replaced by a sharewise addition gadget. The trivial implementation of linear functions is d -PINI for any d [CS20, Proposition 4], since the output shares are computed from input shares that have the same index.

Multiplication Gadget. We use the Hardware Private Circuit 1 (HPC1) gadget by Cassiers et al. [CGLS21] for the multiplication, which follows a refresh-then-multiply strategy that practically consists of an SNI refreshing gadget followed by a Domain-Oriented Masking (DOM)-*indep* multiplication from Groß et al. [GMK16]. The designers of HPC1 prove it to satisfy the PINI composability security in $\mathbb{F}_2$. To be more precise, it was demonstrated to be d -glitch-robust PINI, which is a stronger PINI security notion to take into account the physical defaults of hardware implementations (glitches). A d -glitch-robust PINI gadget is d -PINI. We simply extend it to operate for any prime field $\mathbb{F}_p$ and therefore is still d -PINI.

Squaring Gadget. The first PINI squaring gadget has been proposed, for any masking order in prime fields, by Cassiers et al. in [CMM⁺23, Algorithm 1] also in an extension of the d -probing model to take into account glitches. It was proved to be d -glitch-robust PINI, thus is as well d -PINI (see [CMM⁺23, Proposition 1]). Since we work in the probing model to be device-agnostic, we do not explicitly specify the registers needed to protect against glitches in hardware.

CSIDH Point Selection Gadget. At the very beginning of SingleStep (Line 3, Alg. 3), the algorithm chooses which point, P_0 or P_1 , to perform the following operations on, depending on the value of a secret bit b . Not only the secret bit and the coordinates of those points are sensitive values to protect (cf. Section 4.1), but also the choice of P itself. To mask those information, we design the CSIDH Point Selection gadget defined for any number of shares $d+1$ of both the numerator and denominator. The basic unmasked operation in $\mathbb{F}_p$ is

$$P = b \cdot P_0 + (1 - b) \cdot P_1, \quad (5)$$

with P_0 and P_1 represented by X_0/Z_0 and X_1/Z_1 , respectively. In our paper, X_0, X_1, Z_0 and Z_1 are additively masked as well as the secret bit b . We write the d -th order additive

encoding of X_i by $X_0^{(i)} + X_1^{(i)} + \dots + X_d^{(i)}$. From Eq. (5), it yields

$$\frac{(b_0 + \dots + b_d)(X_0^{(0)} + \dots + X_d^{(0)})}{Z_0^{(0)} + \dots + Z_d^{(0)}} + \frac{(1 - b_0 - \dots - b_d)(X_0^{(1)} + \dots + X_d^{(1)})}{Z_0^{(1)} + \dots + Z_d^{(1)}} \left(= \frac{X_P}{Z_P} \right). \quad (6)$$

Then, we assign the resulting shared numerator and denominator to the shared output coordinates X_P, Z_P , respectively. Each i -th output share of the numerator $X_i^{(P)}$ is computed to include only with the i -th shares $X_0^{(i)}$ and $X_1^{(i)}$ of X_0 and X_1 , respectively, and all the shares of b . The numerator and the denominator of this gadget are d -PINI, respectively.

Proposition 1. *The CSIDH Point Selection gadget is d -PINI on the numerator and denominator individually.*

Proof. For the numerator in Eq. (6), we perform four multiplications and one addition with existing d -PINI gadgets. For the multiplication $(1 - b)X_1$, we rearrange the shares of b by setting $(b'_0, b'_1, \dots, b'_d) = (1 - b_0, -b_1, \dots, -b_d)$. Then, this overall computation is a composite gadget made of d -PINI ones, and [CS20, Proposition 3] ensures that it satisfies the d -PINI security property. The denominator Z_P is already shared and not transformed any further. Since the output variables X_P, Z_P are stored separately, i.e., we do not explicitly compute the fraction in Eq. (6), the whole gadget is d -PINI secure. $\square$

4.3 Quotient Masking

We present a novel sharing technique when the secret $x \in \mathbb{F}_p$ is a quotient $x = A/B$, where $A \in \mathbb{F}_p$ is the numerator and $B \in \mathbb{F}_p \setminus \{0\}$ is the denominator of x . The presented technique is tailored for projectivized secret values, since an affine sensitive value x is written as a pair of separate coordinates $x = (A : B)$, which can be interpreted as a fraction A/B and encodes uniquely the original value x . In this case, it is necessary and sufficient to share only the numerator A or the denominator B to ensure d -probing security; the other value remains unshared and considered non-secret. Indeed, the knowledge of either the numerator or the denominator does not make it possible to recover the original secret x , thus we can leave one value unmasked and focus on protecting the other one. It is especially convenient in our CSIDH implementation, since we frequently manipulate both projective coordinates at the same time. In this section, we first present the quotient masking technique for any order d , before justifying why we only apply it to the first order implementation ($d = 1$) of SingleStep (cf. Alg. 3).

More specifically, for a chosen encoding $\circ$ in $\mathbb{F}_p$, the sharing of the numerator of the secret quotient x is defined by $x = \bigcirc_{i=0}^d A_i / B$, where the denominator B is not shared. On the other hand, the sharing of the denominator of the secret quotient x is expressed by $x = A / \bigcirc_{i=0}^d B_i$ and the numerator A is not shared. As a result, it reduces the number of required shares for a given secret and it also simplifies the design of gadgets and reduces the number of operations to compute, which lead to a performance speed-ups (cf. Table 2).

For our application to CSIDH, we choose to additively share the numerator. Our choice of additive encoding is motivated by the observations made in Section 4.2. In addition to that, we choose to share the numerator over the denominator because we can easily adapt the existing gadgets described in Section 4.2, we do not have to carefully take into account the zero value unlike for the denominator, and we can design a gadget for the CSIDH Point Selection in a simple fashion for the sharing-only of the numerator.

In the remainder of the section, we specify gadgets to handle the addition, multiplication, squaring and CSIDH Point Selection carried on masked quotients whose numerators are additively shared for any number of shares $d + 1$. However, we only present the first-order necessary refactoring of both shared numerator and denominator to fit quotient masking

rules as it breaks security for higher orders in the d -probing model (cf. Section 4.3.1). We want to make it clear that the gadgets are written to handle and output quotient values for the sake of consistency with the fraction representation; and, in our CSIDH masked implementation, we translate that to projectivized coordinates where we compute and store separately the first coordinate (numerator) and the second one (denominator).

Before presenting the gadgets, we clarify that, for our CSIDH masked implementation, `SingleStep` (Alg. 3) and its subroutines `Montgomery Ladder` (Alg. 4), `xADD` (Alg. 5) and `xDBL` (Alg. 6) perform linear and non-linear operations with the first and second projective coordinates of affine variables. An example of such computation is $X_P - Z_P$ (cf. Alg. 3 or Alg. 6). In this case, we trivially add the public value Z_P to the first share of X_P , which does not violate the PINI security notion, without the use of the following addition or multiplication gadgets.

Addition Gadget. We provide in Alg. 7 the addition gadget of two inputs where both numerators are additively shared from quotient masking. Each output share is expressed with a common denominator. The gadget is specified for any number of shares $d + 1$. The trivial implementation of linear functions is d -PINI for any d (cf. Section 4.2). In our case, we multiply the input shares by public values V, B to output a fraction with a common denominator. Nevertheless, it does not interfere with the PINI principles because it does not provide any extra information about our secret numerator. As a result, Alg. 7 is d -PINI. For the sake of completeness, we include a proof in the following proposition.

Proposition 2. *The addition gadget for an additively shared numerator in quotient masking (Alg. 7) is d -PINI.*

Proof. Intuitively, the gadget sums A_i/B and U_i/V in a sharewise manner, which is d -PINI according to [CS20, Proposition 4], and groups the result under a common denominator of public values, which does not interfere with the PINI security notion. In more details, we provide a proof using the same notation as in Definition 4. We have $|\mathcal{O}|$ output and $|\mathcal{I}|$ ($\leq t_1$) internal probes. Each output probe $C_i \cdot B^{-1} \cdot V^{-1}$ is associated to one unique index i (B, V are public and unshared values). Therefore, $|\mathcal{O}| = |\mathcal{A}| = t_2$. (We recall that $t_1 + t_2 \leq d$.) The internal probes are each associated to at most one index (e.g., $A_i V$, $U_i B$). This index is either already included in $\mathcal{A}$, or not and then stored in the set $\mathcal{B}$. As a result, the condition $|\mathcal{B}| \leq t_1$ holds. It results that the joint probability distribution of the output values of the probes in $\mathcal{O} \cup \mathcal{I}$ can be replicated by taking input shares with indices drawn from the set $\mathcal{A} \cup \mathcal{B}$ and public values. Therefore, the gadget is d -PINI. $\square$

Algorithm 7: d th-order PINI Addition Gadget for Additively Shared Numerator.

Input: $\mathbf{A} = (A_i)_{0 \leq i \leq d}$, $\mathbf{U} = (U_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$ and $B, V \in \mathbb{F}_p$ such that
 $\sum_i A_i \cdot B^{-1} = x \bmod p$ and $\sum_i U_i \cdot V^{-1} = y \bmod p$
Output: $\mathbf{C} = (C_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$ and $B, V \in \mathbb{F}_p$ such that
 $\sum_i C_i \cdot (BV)^{-1} = x + y \bmod p$
1 **for** $i = 0$ **to** d **do**
2 $C_i \leftarrow A_i V + U_i B \bmod p$
3 **end**
4 **return** $\mathbf{C} \cdot B^{-1} \cdot V^{-1}$

Multiplication Gadget. Alg. 8 details the two-input multiplication of the d -PINI HPC1 gadget as described in Section 4.2, adapted for quotient masking with an additively shared numerator of $d + 1$ shares. The HPC1 gadget operating on Boolean values was proven to be d -PINI in [CGLS21, Section 7.2] (cf. Section 4.2) and remains so in our adaption.

Proposition 3. *The multiplication gadget, adapted from HPC1, for an additively shared numerator in quotient masking (Alg. 8) is d -PINI.*

Proof. The two-input HPC1 multiplication gadget first refreshes the shares of one input, then multiply the latter with the other shared input. The whole gadget was proven d -PINI in $\mathbb{F}_2$ in [CGLS21, Section 7.2]. For our quotient masking with additively shared numerators, we simply extend the original HPC1 gadget from $\mathbb{F}_2$ to $\mathbb{F}_p$ to multiply both shared input numerators. This results is divided by the product of the denominators, i.e., the public values, which does not interfere with PINI security. About the extension of HPC1, we simply define the refreshing gadget of the original HPC1 gadget (see SNI refreshing gadget in Alg. 12) over $\mathbb{F}_p$ instead of $\mathbb{F}_2$. For the multiplication part, we use the generalization of the HPC1 multiplication gadget to work in $\mathbb{F}_p$ (see DOM-*indep* gadget in Alg. 13). Neither extension from $\mathbb{F}_2$ to $\mathbb{F}_p$ breaks the PINI security notion—the extensions do not change the indices of the shares in the probed expressions—the output numerator is then d -PINI. Since it is divided by a public value, this adapted version remains d -PINI. $\square$

Algorithm 8: d th-order PINI Multiplication Gadget for Additively Shared Numerator according to HPC1.

Input: $\mathbf{A} = (A_i)_{0 \leq i \leq d}, \mathbf{U} = (U_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$ and $B, V \in \mathbb{F}_p$ such that
 $\sum_i A_i \cdot B^{-1} = x \bmod p$ and $\sum_i U_i \cdot V^{-1} = y \bmod p$

Output: $\mathbf{C} = (C_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$ and $B, V \in \mathbb{F}_p$ such that
 $\sum_i C_i \cdot (BV)^{-1} = xy \bmod p$

1 $\mathbf{W} \leftarrow \text{Refresh}(\mathbf{U})$ SNI-Refresh (Alg. 12)
 2 $\mathbf{C} \leftarrow \text{Multiply}(\mathbf{A}, \mathbf{W})$ DOM-Indep in $\mathbb{F}_p$ (Alg. 13)
 3 **return** $\mathbf{C} \cdot B^{-1} \cdot V^{-1}$

Squaring Gadget. We adapt the PINI squaring gadget from [CMM⁺23] to fit a d th-order-additively shared numerator from quotient masking, which we detail in Alg. 9 for one input. Our modifications do not interfere with the compositional security notion.

Proposition 4. *The squaring gadget for an additively shared numerator in quotient masking (Alg. 9) is d -PINI.*

Proof. We simply perform the original squaring gadget on the secret numerator, which is proven d -PINI in [CMM⁺23, Proposition 1], and the public denominator is squared. Then, our adapted squaring gadget is still d -PINI. $\square$

CSIDH Point Selection Gadget. We can easily mask Line 3 from Alg. 3 with an additively shared numerator in Alg. 10, where the algorithm chooses the point P_0 or P_1 according to a secret bit b . We adapt Eq. (6) with our choice of quotient masking for CSIDH, to have

$$\frac{(b_0 + \dots + b_d)(X_0^{(0)} + \dots + X_d^{(0)})}{Z_0} + \frac{(1 - b_0 - \dots - b_d)(X_0^{(1)} + \dots + X_d^{(1)})}{Z_1} \left(= \frac{X_P}{Z_P} \right).$$

However, note that probing Z_0 , Z_1 and Z_P leaks information about the chosen point P , even if those are refreshed. Indeed, Z_P is equal to either Z_0 or Z_1 in this expression; and probing the second projective coordinates make it possible to perform a value comparison. To tackle this problem, we put the quotients under a common denominator and we obtain

$$\frac{(b_0 + \dots + b_d)(X_0^{(0)} + \dots + X_d^{(0)})Z_1 + (1 - b_0 - \dots - b_d)(X_0^{(1)} + \dots + X_d^{(1)})Z_0}{Z_1 Z_0}. \quad (7)$$

Algorithm 9: d th-order PINI Squaring Gadget for Additively Shared Numerator
(adapted from [CMM⁺23, Algorithm 1]).

Input: $\mathbf{A} = (A_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$ and $B \in \mathbb{F}_p$ such that $\sum_i A_i \cdot B^{-1} = x \bmod p$
Output: $\mathbf{C} = (C_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$ and $B^2 \in \mathbb{F}_p$ such that $\sum_i C_i \cdot B^{-2} = x^2 \bmod p$

```

1 for  $i = 0$  to  $d$  do
2   for  $j = i + 1$  to  $d$  do
3      $r_{ij} \xleftarrow{\$} \mathbb{F}_p$ 
4      $s_{ij} \xleftarrow{\$} \mathbb{F}_p$ 
5      $\alpha_{ij} \leftarrow 2A_j + r_{ij} \bmod p$ 
6      $\beta_{ij} \leftarrow A_i \alpha_{ij} + s_{ij} \bmod p$ 
7   end
8 end
9 for  $i = 0$  to  $d$  do
10   $\gamma_i \leftarrow A_i \left( A_i - \sum_{j=i+1}^d r_{ij} \right) \bmod p$ 
11   $C_i \leftarrow \gamma_i + \sum_{j=i+1}^d \beta_{ij} - \sum_{j=0}^{i-1} s_{ji} \bmod p$ 
12 end
13 return  $\mathbf{C} \cdot B^{-2}$ 

```

Each i -th output share of the numerator $X_i^{(P)}$ is computed by arranging the input shares such that it contains only with the i -th shares $X_0^{(i)}$ and $X_1^{(i)}$ of X_0 and X_1 , respectively, and all the shares of b . The linear and non-linear operations in the numerator of the above expression are now performed according to existing d -th order PINI gadgets; thus the overall computation on the shares is PINI, and the denominator remains a public value, as stated in the following proposition.

Proposition 5. *The CSIDH Point Selection gadget for an additively shared numerator in quotient masking (Alg. 10) is d -PINI.*

Proof. The reasoning is similar to the proof of Proposition 1. The secrets are the input numerators, b and $(1-b)$. The denominators Z_0, Z_1 are public values, which do not interfere with the PINI property. In the numerator of Eq. (7), we perform two multiplications and one addition of shared secret values with d -PINI gadgets. Concerning the multiplication $(1-b)X_1$, as in Section 4.2, the shares of b rearranged by defining $(b'_0, b'_1, \dots, b'_d) = (1-b_0, -b_1, \dots, -b_d)$. Then, this overall computation is a composite gadget made of d -PINI ones, and [CS20, Proposition 3] ensures that it remains d -PINI secure. Since the denominators Z_0 and Z_1 are public values, the gadget is d -PINI secure. $\square$

Quotient Additive Refactor Gadget. In our masked algorithm, we may have the numerator (first coordinate) and the denominator (second coordinate) already shared from former computations and treated as secret separately, and we need to realign them with quotient masking requirements for following operations, where one part of the quotient is made non-secret. For that purpose, we define an additive quotient refactor gadget to fit our case with two shares: It turns $\sum_{i=0}^1 A_i$ and $\sum_{i=0}^1 B_i$ into $\sum_{i=0}^1 A_i \cdot B^{-1}$ in Alg. 11. It is used in Alg. 5 and Alg. 6 at the end of the computations for X_R and Z_R to output the new point. We defined it for two shares only ($d = 1$) to take security precautions. We discuss the latter point in the following section and with a first-order security justification.

Algorithm 10: d th-order CSIDH Point Selection Gadget for Additively Shared Numerator.

Input: $\mathbf{b} = (b_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$ such that $\sum_i b_i = b \in \{0, 1\}$,
 $\mathbf{A}^{(0)} = (A_i^{(0)})_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$ and $B_0, B_1 \in \mathbb{F}_p$ such
that $\sum_i A_i^{(0)} \cdot B_0^{-1} = x_0 \bmod p$ and $\sum_i A_i^{(1)} \cdot B_1^{-1} = x_1 \bmod p$
Output: $\mathbf{C} = (C_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$ and $B_0, B_1 \in \mathbb{F}_p$ such that
 $\sum_i C_i \cdot (B_0 B_1)^{-1} = x_b \bmod p$

- 1 $\mathbf{A}^{(0)} \leftarrow B_1 \cdot \mathbf{A}^{(0)}$
- 2 $\mathbf{A}^{(1)} \leftarrow B_0 \cdot \mathbf{A}^{(1)}$
- 3 $\alpha \leftarrow \text{Multiply}(\mathbf{b}, \mathbf{A}^{(1)})$ PINI multiplication (Alg. 8)
- 4 $b_0 \leftarrow 1 - b_0 \bmod p$
- 5 **for** $i = 1$ to d **do**
- 6 $b_i \leftarrow -b_i \bmod p$
- 7 **end**
- 8 $\beta \leftarrow \text{Multiply}(\mathbf{b}, \mathbf{A}^{(0)})$ PINI multiplication (Alg. 8)
- 9 **for** $i = 0$ to d **do**
- 10 $C_i \leftarrow \alpha_i + \beta_i \bmod p$
- 11 **end**
- 12 **return** $\mathbf{C} \cdot B_0^{-1} \cdot B_1^{-1}$

Algorithm 11: First-order Additive Refactor for Numerator Sharing

Input: $\mathbf{A} = (A_0, A_1) \in \mathbb{F}_p^2$ and $\mathbf{B} = (B_0, B_1) \in \mathbb{F}_p^2$ such that $B_0 + B_1 \neq 0$ and
 $(A_0 + A_1)/(B_0 + B_1) = x \bmod p$
Output: $\mathbf{A} = (A_0, A_1) \in \mathbb{F}_p^2$ and $D \in \mathbb{F}_p$ such that $(A_0 + A_1) \cdot D^{-1} = x \bmod p$

- 1 $r_0 \xleftarrow{\$} \mathbb{F}_p \setminus \{0\}$
- 2 **for** $i = 0$ to 1 **do**
- 3 $A_i \leftarrow A_i r_0 \bmod p$
- 4 $B_i \leftarrow B_i r_0 \bmod p$
- 5 **end**
- 6 $D \leftarrow B_0 + B_1 \bmod q$
- 7 **return** $\mathbf{A} \cdot D^{-1}$

4.3.1 Insecure Higher-Order Quotient Additive Refactor Gadget

In this section, we explain why, unfortunately, CSIDH does not use quotient masking for higher masking orders ($d > 1$). This is due to the refactor gadget that is used in the Montgomery Ladder in Alg. 5 and Alg. 6.

For $d = 1$, the output of the refactor gadget (cf. Alg. 11) are the shared numerator $(A_0 r_0, A_1 r_0)$ and the now public denominator $D = (B_0 r_0) + (B_1 r_0)$. There is no attack with one probe to retrieve either the original value A or B because each share is refreshed.

We can trivially extend this gadget for $d = 2$ and obtain as output shares for the numerator $(A_0 r_0 r_1, A_1 r_0 r_1, A_2 r_0 r_1)$ and have the now public value $D = \sum_{i=0}^2 B_i r_0 r_1$. Since $d = 2$, an adversary may put two probes on $(A_i, A_i r_0 r_1)$ or $(A_i r_0 r_1, D = B r_0 r_1)$ for a chosen i ($i \in \{0, 1, 2\}$). We do not find how to recover an origin secret in this setting. However, we choose to be cautious and not use quotient masking for $d = 2$.

However, for the third order, the output of the gadget for the numerator are

$$(A_0 r_0 r_1 r_2, A_1 r_0 r_1 r_2, A_2 r_0 r_1 r_2, A_3 r_0 r_1 r_2)$$

Table 2: Performance comparison of selected masked PQC algorithms. All values are measured in megacycles (10^6 clock cycles). The value for unmasked Kyber was taken from [KPR⁺], which uses the same setup as [BC22].

	CSIDH-2048 (this work)		BIKE-I [DR24]	Kyber768 [BC22]
tested on	x86		x86	Cortex M4
tested function	derive		decaps	Kyber.CCAKEM.Dec
masking variant	standard	quotient		
unmasked	24 308	–	1.07	0.71
$d = 1$	373 320	334 596	329	10.02
$d = 2$	1 240 780	–	723	16.75
$d = 3$	2 239 883	–	1211	24.71
$d = 4$	3 434 419	–	1873	34.68

and the output public denominator is $D = \sum_{i=0}^3 B_i r_0 r_1 r_2$. An adversary can put three probes on the values $(A_i, A_i r_0 r_1 r_2, D)$. Then, the attacker computes $A_i r_0 r_1 r_2 / A_i = r_0 r_1 r_2$ and then $D / (r_0 r_1 r_2) = B$ to recover the original secret B . It leads to a security breach. Because of the required refactoring, i.e., making the denominator of Alg. 5 and Alg. 6 public again after being computed from secret shares, quotient masking is not used for higher-order masking.

5 Implementation and Performance

To justify our theoretical contribution we provide the first arbitrary-order masked implementation of CSIDH. The code is available at <https://github.com/maskedcsidhteam/masked-csidh>.

IMPLEMENTATION DETAILS Our code is based on the public CSIDH implementation by [CCSC⁺24] which runs in constant time and features highly optimized $\mathbb{F}_p$ arithmetic operations (in particular multiplication, addition/subtraction and inversion). For our proof of concept tests, we masked the lowest parameter set (CSIDH-2028) but the generalization to further security levels is straightforward.

To see the impact of quotient masking approach, we implemented two variants of CSIDH: one protected by quotient masking (cf. Section 4.3) and one that is protected by standard masking (cf. Section 4.2). Both versions are structurally the same and share most of the code, they differ only in the Montgomery ladder subroutine. We change from standard to quotient masking at the beginning of this function and convert back once the Montgomery ladder is finished. Since the vast majority of the computation time of CSIDH is spent in this function, this is a common way to generate a meaningful lower bound of our proposed method’s speed-up.

PERFORMANCE AND COMPARISON The results of our tests are shown in Table 2. It displays the number of cycles needed in average over 40 runs of the key derivation and for masking order up to $d = 4$. The experiments were run on a 11th Gen Intel Core i7-1165G7 (single core) with deactivated Turbo Boost.

For comparison, the current state-of-the-art numbers are given for the lattice-based PQC-KEM *Kyber* (now turned into the standard *ML-KEM*) and the code-based PQC-KEM *BIKE*. *Kyber* has a masked implementation for the embedded ARM Cortex-M4 [BC22] to which we can compare to. The unmasked baseline for *Kyber* is the provided by the highly optimized PQM4 implementation of *ML-KEM*. For the comparison with *BIKE*, we cite the benchmarks of the masked implementation for x86 in [DR24] and put them in relation to the currently best (unmasked) cycle count as listed in [BL].

While both *Kyber* and *BIKE* outperform CSIDH in terms of absolute cycle counts, the overhead due to masking is similar or even worse. Figure 1 shows that the *Kyber* overhead

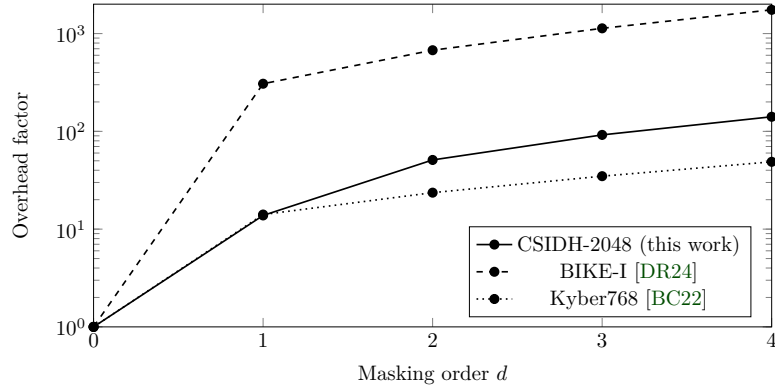


Figure 1: Masking overhead comparison of selected PQC algorithms. The overhead factor is measured relative to the unmasked state of the art cycle count.

factor is close to the CSIDH overhead factor, whereas the BIKE overhead is constantly almost an order of magnitude worse than CSIDH. Interestingly, all three primitives show a similar growth rate after $d = 1$, indicating that the cost of increasing the masking order grows similarly across all three primitives.

Acknowledgments

Jonas Meers, Anna Guinet and Tim Güneysu have been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy - EXC 2092 CASA - 390781972.

References

- [ABC⁺23] Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Tobias Schneider, Markus Schönauer, François-Xavier Standaert, and Christine van Vredendaal. Protecting Dilithium against leakage revisited sensitivity analysis and improved implementations. *IACR TCHES*, 2023(4):58–79, 2023. doi:10.46586/tches.v2023.i4.58–79.
- [ABE⁺24] Bill Allombert, Jean-François Biasse, Jonathan Komada Eriksen, Péter Kutas, Chris Leonardi, Aurel Page, Renate Scheidler, and Márton Tot Bagi. PEARL-SCALLOP: Parameter extension applicable in real-life SCALLOP. Cryptology ePrint Archive, Report 2024/1744, 2024. URL: <https://eprint.iacr.org/2024/1744>.
- [AVL21] Rodrigo Abarzúa, Claudio Valencia, and Julio López. Survey on performance and security problems of countermeasures for passive side-channel attacks on ECC. *Journal of Cryptographic Engineering*, 11(1):71–102, March 2021. URL: <http://dx.doi.org/10.1007/s13389-021-00257-8>, doi:10.1007/s13389-021-00257-8.
- [BBC⁺21] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. CTIDH: faster constant-time CSIDH. *IACR TCHES*, 2021(4):351–387, 2021. URL: <https://tches.iacr.org/index.php/TCHES/article/view/9069>, doi:10.46586/tches.v2021.i4.351–387.

- [BBD⁺15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 457–485. Springer, Berlin, Heidelberg, April 2015. doi:[10.1007/978-3-662-46800-5_18](https://doi.org/10.1007/978-3-662-46800-5_18).
- [BBD⁺16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016. doi:[10.1145/2976749.2978427](https://doi.org/10.1145/2976749.2978427).
- [BBJ⁺08] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted Edwards curves. In Serge Vaudenay, editor, *AFRICACRYPT 08*, volume 5023 of *LNCS*, pages 389–405. Springer, Berlin, Heidelberg, June 2008. doi:[10.1007/978-3-540-68164-9_26](https://doi.org/10.1007/978-3-540-68164-9_26).
- [BC22] Olivier Bronchain and Gaëtan Cassiers. Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based KEMs. *IACR TCHES*, 2022(4):553–588, 2022. doi:[10.46586/tches.v2022.i4.553-588](https://doi.org/10.46586/tches.v2022.i4.553-588).
- [BCH⁺22] Lejla Batina, Łukasz Chmielewski, Björn Haase, Niels Samwel, and Peter Schwabe. SoK: SCA-secure ECC in software – mission impossible? *IACR Transactions on Cryptographic Hardware and Embedded Systems*, page 557–589, November 2022. URL: <http://dx.doi.org/10.46586/tches.v2023.i1.557-589>, doi:[10.46586/tches.v2023.i1.557-589](https://doi.org/10.46586/tches.v2023.i1.557-589).
- [BDFLS20] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *Open Book Series*, 4(1):39–55, December 2020. URL: <http://dx.doi.org/10.2140/obs.2020.4.39>.
- [BGR⁺21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking Kyber: First- and higher-order implementations. *IACR TCHES*, 2021(4):173–214, 2021. URL: <https://tches.iacr.org/index.php/TCHES/article/view/9064>, doi:[10.46586/tches.v2021.i4.173-214](https://doi.org/10.46586/tches.v2021.i4.173-214).
- [BL] Daniel J. Bernstein and Tanja Lange. ECRYPT benchmarking of asymmetric systems project. "<https://bench.cr.yp.to/results-kem/amd64-hertz.html>". Accessed: 04-May-2026.
- [BR23] Sonia Belaïd and Matthieu Rivain. High order side-channel security for elliptic-curve implementations. *IACR TCHES*, 2023(1):238–276, 2023. doi:[10.46586/tches.v2023.i1.238-276](https://doi.org/10.46586/tches.v2023.i1.238-276).
- [BS01] Dan Boneh and Igor Shparlinski. On the unpredictability of bits of the elliptic curve Diffie-Hellman scheme. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 201–212. Springer, Berlin, Heidelberg, August 2001. doi:[10.1007/3-540-44647-8_12](https://doi.org/10.1007/3-540-44647-8_12).
- [BV96] Dan Boneh and Ramarathnam Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 129–142. Springer, Berlin, Heidelberg, August 1996. doi:[10.1007/3-540-68697-5_11](https://doi.org/10.1007/3-540-68697-5_11).

- [CCC⁺19] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. Stronger and faster side-channel protections for CSIDH. In Peter Schwabe and Nicolas Thériault, editors, *LATINCRYPT 2019*, volume 11774 of *LNCS*, pages 173–193. Springer, Cham, October 2019. doi:10.1007/978-3-030-30530-7_9.
- [CCSC⁺24] Fabio Campos, Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Michael Meyer, Krijn Reijnders, Francisco Rodríguez-Henríquez, Peter Schwabe, and Thom Wiggers. Optimizations and practicality of high-security CSIDH. *CiC*, 1(1):5, 2024. doi:10.62056/anjbkstdja.
- [CDV20] Wouter Castryck, Thomas Decru, and Frederik Vercauteren. Radical isogenies. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 493–519. Springer, Cham, December 2020. doi:10.1007/978-3-030-64834-3_17.
- [CGLS21] Gaetan Cassiers, Benjamin Gregoire, Itamar Levi, and Francois-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 70(10):1677–1690, October 2021. URL: <http://dx.doi.org/10.1109/tc.2020.3022979>, doi:10.1109/tc.2020.3022979.
- [CGTZ23a] Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. High-order masking of NTRU. *IACR TCHES*, 2023(2):180–211, 2023. doi:10.46586/tches.v2023.i2.180-211.
- [CGTZ23b] Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. Improved gadgets for the high-order masking of Dilithium. *IACR TCHES*, 2023(4):110–145, 2023. doi:10.46586/tches.v2023.i4.110-145.
- [CH17] Craig Costello and Hüseyin Hisil. A simple and compact algorithm for SIDH with arbitrary degree isogenies. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 303–329. Springer, Cham, December 2017. doi:10.1007/978-3-319-70697-9_11.
- [CHMR25] Fabio Campos, Andreas Hellenbrand, Michael Meyer, and Krijn Reijnders. dCTIDH: Fast & deterministic CTIDH. *IACR TCHES*, 2025(3):516–541, 2025. doi:10.46586/tches.v2025.i3.516-541.
- [CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO’99*, volume 1666 of *LNCS*, pages 398–412. Springer, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_26.
- [CLM⁺18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 395–427. Springer, Cham, December 2018. doi:10.1007/978-3-030-03332-3_15.
- [CLP24] Mingjie Chen, Antonin Leroux, and Lorenz Panny. SCALLOP-HD: Group action from 2-dimensional isogenies. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14603 of *LNCS*, pages 190–216. Springer, Cham, April 2024. doi:10.1007/978-3-031-57725-3_7.

- [CMM⁺23] Gaëtan Cassiers, Loïc Masure, Charles Momin, Thorben Moos, and François-Xavier Standaert. Prime-field masking in hardware and its soundness against low-noise SCA attacks. *IACR TCHES*, 2023(2):482–518, 2023. doi:[10.46586/tches.v2023.i2.482-518](https://doi.org/10.46586/tches.v2023.i2.482-518).
- [Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin Kaya Koç and Christof Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 292–302. Springer, Berlin, Heidelberg, August 1999. doi:[10.1007/3-540-48059-5_25](https://doi.org/10.1007/3-540-48059-5_25).
- [CPRR14] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 410–424. Springer, Berlin, Heidelberg, March 2014. doi:[10.1007/978-3-662-43933-3_21](https://doi.org/10.1007/978-3-662-43933-3_21).
- [CS20] Gaetan Cassiers and Francois-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020. URL: <http://dx.doi.org/10.1109/tifs.2020.2971153>, doi:[10.1109/tifs.2020.2971153](https://doi.org/10.1109/tifs.2020.2971153).
- [CSCJR22] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. The SQALE of CSIDH: sublinear Vélú quantum-resistant isogeny action with low exponents. *Journal of Cryptographic Engineering*, 12(3):349–368, September 2022. doi:[10.1007/s13389-021-00271-w](https://doi.org/10.1007/s13389-021-00271-w).
- [DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, Berlin, Heidelberg, May 2014. doi:[10.1007/978-3-642-55220-5_24](https://doi.org/10.1007/978-3-642-55220-5_24).
- [DEF⁺25] Pierrick Dartois, Jonathan Komada Eriksen, Tako Boris Fouotsa, Arthur Herlédan Le Merdy, Riccardo Invernizzi, Damien Robert, Ryan Rueger, Frederik Vercauteren, and Benjamin Wesolowski. PEGASIS: Practical effective class group action using 4-dimensional isogenies. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part I*, volume 16000 of *LNCS*, pages 67–99. Springer, Cham, August 2025. doi:[10.1007/978-3-032-01855-7_3](https://doi.org/10.1007/978-3-032-01855-7_3).
- [DFK⁺23] Luca De Feo, Tako Boris Fouotsa, Péter Kutas, Antonin Leroux, Simon-Philipp Merz, Lorenz Panny, and Benjamin Wesolowski. SCALLOP: Scaling the CSI-FiSh. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 345–375. Springer, Cham, May 2023. doi:[10.1007/978-3-031-31368-4_13](https://doi.org/10.1007/978-3-031-31368-4_13).
- [DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. doi:[10.1109/TIT.1976.1055638](https://doi.org/10.1109/TIT.1976.1055638).
- [DR24] Loïc Demange and Mélissa Rossi. A provably masked implementation of BIKE key encapsulation mechanism. *CiC*, 1(1):23, 2024. doi:[10.62056/aesgvua5v](https://doi.org/10.62056/aesgvua5v).
- [FGDM⁺10] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. State-of-the-art of secure ECC implementations: A survey on known side-channel attacks and countermeasures. In *2010 IEEE*

- International Symposium on Hardware-Oriented Security and Trust (HOST)*, page 76–87. IEEE, June 2010. URL: <http://dx.doi.org/10.1109/hst.2010.5513110>, doi:10.1109/hst.2010.5513110.
- [FHKP13] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 254–271. Springer, Berlin, Heidelberg, February / March 2013. doi:10.1007/978-3-642-36362-7_17.
- [FV12] Junfeng Fan and Ingrid Verbauwhede. *An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost*, page 265–282. Springer Berlin Heidelberg, 2012. URL: http://dx.doi.org/10.1007/978-3-642-28368-0_18, doi:10.1007/978-3-642-28368-0_18.
- [GdKQ⁺24] Phillip Gajland, Bor de Kock, Miguel Quaresma, Giulio Malavolta, and Peter Schwabe. SWOOSH: Efficient lattice-based non-interactive key exchange. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024. URL: <https://www.usenix.org/conference/usenixsecurity24/presentation/gajland>.
- [GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security, CCS’16*, page 3–3. ACM, October 2016. URL: <http://dx.doi.org/10.1145/2996366.2996426>, doi:10.1145/2996366.2996426.
- [GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the “duplication” method). In Çetin Kaya Koç and Christof Paar, editors, *CHES’99*, volume 1717 of *LNCS*, pages 158–172. Springer, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48059-5_15.
- [HR23] Nadia Heninger and Keegan Ryan. The hidden number problem with small unknown multipliers: Cryptanalyzing MEGA in six queries and other applications. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 147–176. Springer, Cham, May 2023. doi:10.1007/978-3-031-31368-4_6.
- [ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Berlin, Heidelberg, August 2003. doi:10.1007/978-3-540-45146-4_27.
- [JJV07] David Jao, Dimitar Jetchev, and Ramarathnam Venkatesan. On the bits of elliptic curve Diffie-Hellman keys. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *INDOCRYPT 2007*, volume 4859 of *LNCS*, pages 33–47. Springer, Berlin, Heidelberg, December 2007. doi:10.1007/978-3-540-77026-8_4.
- [KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO’99*, volume 1666 of *LNCS*, pages 388–397. Springer, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_25.
- [KLRBG22] Markus Krausz, Georg Land, Jan Richter-Brockmann, and Tim Güneysu. Efficiently masking polynomial inversion at arbitrary order. In Jung Hee Cheon and Thomas Johansson, editors, *PQCrypto 2022*, pages 309–326. Springer, Cham, September 2022. doi:10.1007/978-3-031-17234-2_15.

- [KO63] A. Karatsuba and Yu. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595, January 1963.
- [Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitiz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Berlin, Heidelberg, August 1996. doi:10.1007/3-540-68697-5_9.
- [KPR⁺] Matthias J. Kannwischer, Richard Petri, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. <https://github.com/mupq/pqm4>. Accessed: 04-May-2026.
- [KW16] Hugo Krawczyk and Hoeteck Wee. The OPTLS protocol and TLS 1.3. In *2016 IEEE European Symposium on Security and Privacy*, pages 81–96. IEEE Computer Society Press, March 2016. doi:10.1109/EuroSP.2016.18.
- [MBA⁺21] Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk. Raccoon attack: Finding and exploiting most-significant-bit-oracles in TLS-DH(E). In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 213–230. USENIX Association, August 2021. URL: <https://www.usenix.org/conference/usenixsecurity21/presentation/merget>.
- [MCR19] Michael Meyer, Fabio Campos, and Steffen Reith. On lions and elligators: An efficient constant-time implementation of CSIDH. In Jintai Ding and Rainer Steinwandt, editors, *PQCrypto 2019*, pages 307–325. Springer, Cham, 2019. doi:10.1007/978-3-030-25510-7_17.
- [Mes01] Thomas S. Messerges. Securing the AES finalists against power analysis attacks. In Bruce Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 150–164. Springer, Berlin, Heidelberg, April 2001. doi:10.1007/3-540-44706-7_11.
- [MGTF19] Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking Dilithium - efficient implementation and side-channel evaluation. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 2019*, volume 11464 of *LNCS*, pages 344–362. Springer, Cham, June 2019. doi:10.1007/978-3-030-21568-2_17.
- [MN23] Jonas Meers and Julian Nowakowski. Solving the hidden number problem for CSIDH and CSURF via automated coppersmith. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part IV*, volume 14441 of *LNCS*, pages 39–71. Springer, Singapore, December 2023. doi:10.1007/978-981-99-8730-6_2.
- [Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985. URL: <http://dx.doi.org/10.1090/s0025-5718-1985-0777282-x>, doi:10.1090/s0025-5718-1985-0777282-x.
- [MOT20] Tomoki Moriya, Hiroshi Onuki, and Tsuyoshi Takagi. How to construct CSIDH on Edwards curves. In Stanislaw Jarecki, editor, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 512–537. Springer, Cham, February 2020. doi:10.1007/978-3-030-40186-3_22.

- [MP] Moxie Marlinspike and Trevor Perrin. The X3DH key agreement protocol. <https://signal.org/docs/specifications/x3dh/>. Accessed: 04-May-2026.
- [MR18] Michael Meyer and Steffen Reith. A faster way to the CSIDH. In Debrup Chakraborty and Tetsu Iwata, editors, *INDOCRYPT 2018*, volume 11356 of *LNCS*, pages 137–152. Springer, Cham, December 2018. doi:10.1007/978-3-030-05378-9_8.
- [OAYT19] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. (Short paper) A faster constant-time algorithm of CSIDH keeping two points. In Nuttapong Attrapadung and Takeshi Yagi, editors, *IWSEC 19*, volume 11689 of *LNCS*, pages 23–33. Springer, Cham, August 2019. doi:10.1007/978-3-030-26834-3_2.
- [PGMP19] Thomas Prest, Dahmun Goudarzi, Ange Martinelli, and Alain Passelègue. Unifying leakage models on a Rényi day. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 683–712. Springer, Cham, August 2019. doi:10.1007/978-3-030-26948-7_24.
- [PPS25] Lorenz Panny, Christophe Petit, and Miha Stopar. KLaPoTi: An asymptotically efficient isogeny group action from 2-dimensional isogenies. *CiC*, 2(3):5, 2025. doi:10.62056/ahp2wakrz.
- [Ren18] Joost Renes. Computing isogenies between Montgomery curves using the action of $(0, 0)$. In Tanja Lange and Rainer Steinwandt, editors, *PQCrypto 2018*, pages 229–247. Springer, Cham, 2018. doi:10.1007/978-3-319-79063-3_11.
- [Rob24] Damien Robert. Fast pairings via biextensions and cubical arithmetic. Cryptology ePrint Archive, Report 2024/517, 2024. URL: <https://eprint.iacr.org/2024/517>.
- [SG18] Pascal Sasdrich and Tim Güneysu. Exploring RFC 7748 for hardware implementation: Curve25519 and curve448 with side-channel protection. *Journal of Hardware and Systems Security*, 2(4):297–313, Dec 2018. doi:10.1007/s41635-018-0048-z.
- [Sha17] Barak Shani. On the bit security of elliptic curve Diffie-Hellman. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 361–387. Springer, Berlin, Heidelberg, March 2017. doi:10.1007/978-3-662-54365-8_15.
- [Sil09] Joseph H Silverman. *The Arithmetic of Elliptic Curves*. Graduate texts in mathematics. Springer, Dordrecht, 2009. URL: <https://cds.cern.ch/record/1338326>.
- [TJ10] Michael Tunstall and Marc Joye. Coordinate blinding over large prime fields. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 443–455. Springer, Berlin, Heidelberg, August 2010. doi:10.1007/978-3-642-15031-9_30.
- [WPH⁺22] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 679–697. USENIX Association, August 2022. URL: <https://www.usenix.org/conference/usenixsecurity22/presentation/wang-yingchen>.

- [XHS20] Jun Xu, Lei Hu, and Santanu Sarkar. Cryptanalysis of elliptic curve hidden number problem from PKC 2017. *Designs, Codes and Cryptography*, 88(2):341–361, Feb 2020. doi:10.1007/s10623-019-00685-y.

A Gadgets from Algorithm 8

Alg. 12 and Alg. 13 describe respectively how to perform the the SNI refreshing and DOM-*indep* multiplication from Alg. 8, i.e., the HPC1 gadget-variant for quotient masking where the numerator is additively shared. The SNI refreshing gadget is adapted from [CGLS21] to operate in $\mathbb{F}_p$ for $d = 1, \dots, 4$ (for higher orders, one can adapt in a similar fashion the original gadget). The DOM-*indep* gadget was firstly introduced by Groß et al. in [GMK16] to operate in $\mathbb{F}_2$; but we take its generalization in $\mathbb{F}_p$ from [CMM⁺23, Algorithm 4].

Algorithm 12: d th-order Optimized SNI-Refresh in $\mathbb{F}_p$ for $d \in [1, 4]$ (adapted from [CGLS21]).

Input: Order $d \in \mathbb{N}$, $\mathbf{U} = (U_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$
Output: $\mathbf{W} = (W_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$

```

1 if  $d = 1$  then
2    $r_0 \xleftarrow{\$} \mathbb{F}_p$ 
3    $W_0 \leftarrow U_0 + r_0 \bmod p$ 
4    $W_1 \leftarrow U_1 - r_0 \bmod p$ 
5 end
6 if  $d = 2$  then
7    $r_0, r_1 \xleftarrow{\$} \mathbb{F}_p$ 
8    $W_0 \leftarrow U_0 + r_0 \bmod p$ 
9    $W_1 \leftarrow U_1 + r_1 \bmod p$ 
10   $W_2 \leftarrow U_2 - r_0 - r_1 \bmod p$ 
11 end
12 if  $d = 3$  or  $d = 4$  then
13   for  $i = 0$  to  $d$  do
14      $r_i \xleftarrow{\$} \mathbb{F}_p$ 
15   end
16   for  $i = 0$  to  $d$  do
17      $W_i \leftarrow U_i + r_i - r_{i-1 \bmod (d+1)} \bmod p$ 
18   end
19 end
20 return  $\mathbf{W}$ 

```

Algorithm 13: d th-order DOM-Indep in $\mathbb{F}_p$ [CMM⁺23, Algorithm 4].

Input: $\mathbf{A} = (A_i)_{0 \leq i \leq d}$, $\mathbf{W} = (W_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$

Output: $\mathbf{C} = (C_i)_{0 \leq i \leq d} \in \mathbb{F}_p^{d+1}$ such that $\sum_i C_i = \sum_i A_i \cdot W_i \bmod p$

```

1 for  $i = 0$  to  $d$  do
2   for  $j = i + 1$  to  $d$  do
3      $r_{ij} \xleftarrow{\$} \mathbb{F}_p$ 
4      $r_{ji} \leftarrow -r_{ij} \bmod p$ 
5   end
6 end
7 for  $i = 0$  to  $d$  do
8    $C_i \leftarrow A_i \cdot W_i + \sum_{j=0, j \neq i}^d (A_i \cdot W_j + r_{ij}) \bmod p$ 
9 end
10 return  $\mathbf{C}$ 

```
